2018

OPTIMIZATION FOR STRUCTURAL EQUATION MODELING: APPLICATIONS TO SUBSTANCE USE DISORDERS

Mahsa Zahery
Virginia Commonwealth University

OPTIMIZATION FOR STRUCTURAL EQUATION MODELING: APPLICATIONS TO SUBSTANCE USE DISORDERS

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

MAHSA ZAHERY

Master of Science from Chalmers University of Technology in Sweden, 2010

Bachelor of Science from Azad University of Central Tehran in Iran, 2007

Director: Dr. Tomasz Arodz,

Associate Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

February, 2018

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Abstract

OPTIMIZATION FOR STRUCTURAL EQUATION MODELING:
APPLICATIONS TO SUBSTANCE USE DISORDERS

By Mahsa Zahery

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2018.

Director: Dr. Tomasz Arodz,
Associate Professor, Department of Computer Science

Substance abuse is a serious issue in both modern and traditional societies. Besides health complications such as depression, cancer and HIV, social complications such as loss of concentration, loss of job, and legal problems are among the numerous hazards substance use disorder imposes on societies. Understanding the causes of substance abuse and preventing its negative effects continues to be the focus of much research.

Substance use behaviors, symptoms and signs are usually measured in form of ordinal data, which are often modeled under threshold models in Structural Equation Modeling (SEM). In this dissertation, we have developed a general nonlinear optimizer for the software package OpenMx, which is a SEM package in widespread use in the fields of psychology and genetics. The optimizer solves nonlinearly constrained optimization problems using a Sequential Quadratic Programming (SQP) algorithm. We have tested the performance of our optimizer on ordinal data and compared the results with two other optimizers (implementing SQP algorithm) available in the

OpenMx package. While all three optimizers reach the same minimum, our new optimizer is faster than the other two.

We then applied OpenMx with our optimization engine to a very large population-based drug abuse dataset, collected in Sweden from over one million pairs, to investigate the effects of genetic and environmental factors on liability to drug use. Finally, we investigated the reasons behind better performance of our optimizer by profiling all three optimizers as well as analyzing their memory consumption. We found that objective function evaluation is the most expensive task for all three optimizers, and that our optimizer needs fewer number of calls to this function to find the minimum. In terms of memory consumption, the optimizers use the same amount of memory.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Substance abuse is a major health problem in today's societies. Serious medical problems including cancer, heart disease, and AIDS are directly linked to substance abuse. According to the 2014 World Drug Report of the United Nations Office on Drugs and Crime, between 3.5 and 7% of the worlds population aged 1564 used an illicit drug in 2012. Approximately 0.6% of the population can be categorized as problem drug users (United Nations Office on Drugs and Crime [1]). In the US, the lifetime prevalence of drug use disorder (drug abuse or dependence) has been estimated at 10.3% [2].

Substance abuse does not only impact people at personal levels, many social problems such as driving under the influence, violence, stress and child abuse are direct consequences of substance abuse. Besides the personal and social costs it imposes on a society, the financial cost associated with this problem is enormous. According to the National Institute on Drug Abuse (NIDA), only the health care cost for using tobacco, alcohol and illicit drugs is around $165 billion in US. Including the costs associated with crimes and lost productivity, the overall cost is around $700 billion [3]. Yet, despite the enormous health, social and financial costs, substances continue to be commonly used all around the world.

The etiology of substance use, abuse and addiction with the aim of preventing its adverse effects has been the focus of research for many years. Numerous studies confirm that both genetic and environmental factors make an individual susceptible

to substance abuse [4, 5, 6, 7, 8, 9]. Environmental factors are the factors in an individual's surrounding that increase the person's liability to drug abuse. Genetic factors make up for the inherited component in such liability.

The human genome contains 3 billion of nucleotide base pairs. These base pairs are building blocks of our DNA. They are located in the 23 pairs of chromosomes within the nucleus of our cells. Chromosomes store our genetic information. They contain thousands of genes. Most of these genes are polymorphic meaning that there are two or more alternative forms of the same gene. Single nucleotide polymorphisms (SNP) are the most common type of genetic variation in humans, but for complex traits such as substance use, it is unlikely that one gene is responsible for the large portion of variation. Instead multiple genes control such complex traits. Such traits are called polygenic traits. Twin studies are helpful in studying polygenic traits.

As for environmental factors, There are two sources for total environmental variation in a trait: shared and specific. Shared environmental influences are those that are shared between twins. For example, a family's Socioeconomic status is the same for both twins in the family. Specific or nonshared environmental factors are those factors that affects one twin but not the other. For example, twins can have unique experiences at school. The individual shocks that either twin might receive independently of their co-twin account for unique or specific environmental factors.

Twin studies are favored by scientists who seek to study the effects of genetics and environment on human development [10, 11, 12, 13]. Identical twins (monozygotic) who share 100% of their genes can only be different due to the environmental factors. Additionally, if the twins are raised in the same house, many of the environmental factors are the same for them. For such twins, the differences between the individuals are caused by their unique experiences. On the other hand, identical twins might be raised separately. Such twins can be compared against each other to find the effect

of environmental factors in human development. The same classification is true for non-identical twins (dizygotic) who share 50% of their genes. They can be raised together or separately. Non-identical twins can also be studied to test the impacts between genes and environment. Twin studies have contributed immensely to finding the reasons why individuals are susceptible to substance abuse [14, 15, 16, 17, 12, 18, 19, 20, 21, 22, 23].

Substance use behavior can be measured in form of ordinal data. Behavioral data are often of binary (yes/no responses) or ordinal (none/some/a lot) type, which are inherently less accurate than continuous measures. One common approach with binary/ordinal data is to assume that there is an underlying, normally distributed continuous variable for each binary/ordinal variable [24]. For example, in substance use behavior, sensitivity to the rewarding experience of drug use may form part of the underlying propensity to use substances frequently.

Such liability is typically thought of as being due to the additive effects of a large number of factors each of small effect, which the central limit theorem predicts will generate a normal distribution of liability [24, 25]. Thresholds on this liability distribution delimit binary or ordinal response categories. For binary data, "yes" responses are observed above a threshold, while "no" responses are observed below that. For ordinal data, the number of thresholds is one fewer than the number of categories in the data. For example, subjects with liability below the first threshold have observed response value of none. Subjects with liability in between the first and second thresholds have observed response value of some and those with liability above the second threshold have observed value of a lot.

Assuming an underlying threshold model, substance use behavior can be studied by modeling likelihood of the observed drug use response patterns in the data. To do this, Structural Equation Modeling (explained in details in background chapter) is a

useful analytical framework.

1.2 Contributions of the Dissertation

In this study, we developed an optimizer for the software package OpenMx, which is a free, open-source, widely popular SEM package that runs inside R, and is available as an R package on the Comprehensive R Archive Network (CRAN) [26, 27]. Our optimizer solves general nonlinearly constrained problems using a Sequential Quadratic Programming (SQP) algorithm. We have compared the performance of our new optimizer with its peer optimizers in OpenMx, and obtained favorable results for ordinal datasets: our developed optimizer runs faster and is more consistent than the other two.

We then applied this newly developed OpenMx optimizer to a Swedish substance abuse dataset collected from the medical and criminal registries, and studied the quantitative and qualitative contributions of genetic and environmental components ascertained through these different sources. The results show a significant contribution of genetic factors, and a moderate contribution of environmental factors in susceptibility to drug abuse. Both components are higher in males than in females, and higher through criminal records than medical records.

Next, we compared the performance of the optimizers on the Swedish data set and found results consistent with those of the synthetic datasets. We then profiled all three optimizers with Valgrinds' profiling tool Callgrind. We also used Valgrind's heap profiler tool Massif to compare memory usage of the optimizers [28].

Finally, several important features were added to our optimizer to enhance its performance in the presence of analytical gradients and Jacobians. We also implemented the central difference approximation method for our optimizer to enable the user to choose between forward and central approximation methods for calculating

numerical gradients/Jacobians when the analytical gradients/Jacobians are not manually provided to the optimizer. Moreover, we improved the optimizer's performance by guiding the optimizer to find a feasible search direction when the initial points are infeasible.

1.3 Structure of the Proposal

The rest of this proposal is organized as follows. Chapter 2 provides a background on the research conducted in this dissertation. Structural Equation Modeling (SEM) is covered in section 2.1. Section 2.2 describes OpenMx as one of the most popular SEM software packages, with discussions of factor analysis and threshold models in subsections 2.2.1 and 2.2.2 respectively. Section 2.3 highlights the role of optimization in SEM. Section 2.4 covers standard maximum likelihood and full information maximum likelihood fit functions in OpenMx.

Chapter 3 discusses optimization methods used in our research briefly. Gradient-based algorithms are discussed in subsection 3.1. Unconstrained nonlinear algorithms are explained in section 3.1.1 with steepest descent in section 3.1.1.1, conjugate gradient in section 3.1.1.2, Newton's method in section 3.1.1.3 and BroydenFletcher-GoldfarbShanno algorithm in section 3.1.1.4. Constrained nonlinear algorithms are explained in section 3.1.2 with subsections primal methods in section 3.1.2.1 and Sequential Quadratic Programming and Quadratic Programming in sections 3.1.2.2 and 3.1.2.3 respectively. The chapter ends with section 3.2 which discusses the other two optimizers within the OpenMx package (NPSOL and SLSQP) in subsections 3.2.1 and 3.2.2.

Chapter 4 discusses the newly developed optimizer (CSOLNP) for the OpenMx package. Section 4.1 provides an introduction to the optimizer CSOLNP. Section 4.2 explains CSOLNP's algorithm in detail, including the optional arguments that can

affect the speed and accuracy with which the solution may be found.

Using simulated data, chapter 5 provides performance comparison between three optimizers on both threshold and continuous models. Section 5.1.1 provides the comparison results of the optimizers' runtime averaged over 250 different replications of data for threshold models. Section 5.1.2 provides the comparison results of the optimizers' runtime averaged over 250 replications of different threshold models with different starting values. Section 5.2 analyzes the performance of the optimizers on continuous models, with the same simulation setup as threshold models explained in sections 5.2.1 and 5.2.2. A conclusion section on the performance of the optimizers on threshold and continuous models is presented in section 5.3.

Chapter 6 describes the application of the CSOLNP optimizer to data on drug use. Section 6.1 discusses the statistical models to be used. Section 6.2 presents the conclusion of our research on drug use, and section 6.3, provides a figure of the performance comparison of the optimizers' runtime on the drug abuse dataset collected from Sweden.

We discuss the optimizers' profiling results in chapter 7. Section 7.1 explains Valgrind [28] as a popular profiling suite and a summary of the tools available within this program. Section 7.2 explains Valgrind's profiling tool Callgrind in details, and is followed by subsection 7.2.1 where the results of running Callgrind on the three optimizers are presented. In section 7.3, we explain Valgrind's heap profiler Massif and present the results of comparing memory usage of the optimizers. In section 7.4, the chapter is summarized.

In chapter 8, we present several features that we added to CSOLNP to improve its performance. In section 8.1, analytical gradients are discussed, followed by section 8.2 where we explained analytical Jacobians. Section 8.3 discusses central difference approximation, and finally in section 8.4, we explain how CSOLNP solves the problem

of infeasible initial point. Finally, in chapter 9 we summarize all the chapters and present the conclusions from the dissertation.

CHAPTER 2

BACKGROUND

2.1 Structural Equation Modeling

Structural Equation Modeling (SEM) is a set of statistical methods to fit models to data. Users typically begin SEM with an explicit hypotheses which are then represented in a parametric model for variances, covariances and (optionally) means of measures of interest. The model is then fitted to data to see whether the model is rejected using various statistical criteria. [29, 30, 31].

There are two types of variables in SEM: latent variables and observed variables. Latent variables are the set of variables that are not directly measured. They are inferred from a set of variables that are directly observed and measured using tests and surveys, called observed or manifest variables. Latent variables may be referred to as factors or constructs, particularly when a set of similar items are used to measure a trait of interest, such as IQ, substance use or depression.

The history of SEM [32, 33] goes back to the development of linear regression models where dependent observed variables are predicted using a set of weighted independent observed variables that minimizes the sum of squared residual errors. A regression model consists of only observed variables. Sometimes, however, investigators will posit that there exists a variable which was not measured, but which generates covariances among the variables that were measured. Factor analysis [34] is an example of a SEM that includes latent variables.

With SEM, we can test hypothesized patterns of causal and correlational relationships among a set of observed and latent variables. Causal relationships between

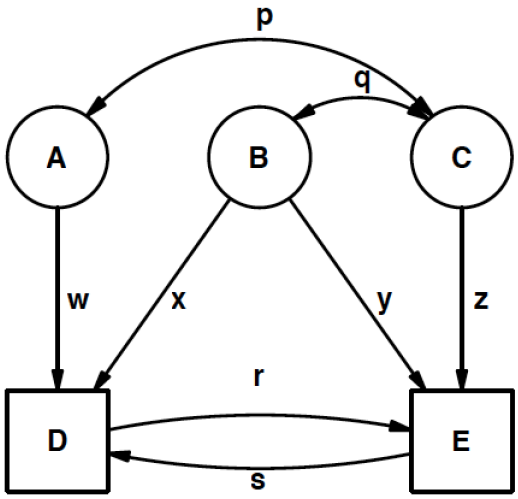


Fig. 1. Path diagram of three latent variables and two observed variables as well as correlations and path coefficients

variables indicate that one variable is the result of the occurrence of the other variable. This is also referred to as cause and effect. Correlational relationship between variables describes the size and direction of a relationship between two or more variables. With correlation, we can not automatically infer that the change in one variable is the cause of the change in the other variable. In Fig. 1, we illustrate a path diagram of three latent variables, and two observed variables and the causal and correlational relationships between them [35].

In Fig. 1, D and E are observed variables shown with rectangles. A, B and C are latent variables enclosed in circles. Another distinction between variables are dependent and independent variables. In this figure, D and E are dependent variables. These are the variables that we try to predict. A, B and C are independent variables. They help in predicting the dependent variables D and E.

The single-headed arrows from A to D, B to D and E, and C to E define causal relationships in the model. A, B and C at the head of the single-headed arrows are the *causes*, while D and E at the tail of the arrows are the *effects*. There are

two single-headed arrows between D and E. There is a feedback-loop or reciprocal causation between these two observed variables. This means D and E cause each other.

Double headed arrows between correlations. In Fig. 1, A and C are correlated. C also correlates with B. But A and B are not correlated. So, although A correlates with C and C correlates with B, we cannot necessarily conclude that A is correlated with B. Also, although A and B both cause dependent variable D, this does not imply that they are correlated.

p and q in Fig. 1 are correlation coefficients. r, s, w, x, y and z correspond to causal paths between latent and observed variables and are called path coefficients. These are simply the weights from the regression analyses.

Regardless of the data type, SEM involves four main steps: model specification; model identification; model estimation; and model testing. Each of these steps are explained in the context of one of the most popular SEM software packages called OpenMx in section 2.2.

2.1.1 Popularity of SEM

SEM is popular for the following reasons:

- In early work fitting SEMs, data were usually summarized as covariance or correlation matrices, because the likelihood (the quantity we wish to maximize) can be evaluated very rapidly, regardless of sample size. Today, models are more likely to be fitted to the raw data because it is convenient when there are missing observations, and certain models (such as mixture distributions) can only be distinguished by using the likelihood of each observation separately.

- Datasets have become larger and the statistical models have become more so-

phisticated. SEM is capable of testing complex theoretical models as well as handling a large number of variables to deal with such complex models.

- There is always a measurement error associated with observed variables. Before SEM, measurement errors and data analysis were handled separately. In addition to being capable of analyzing latent and observed variables together, SEM can model measurement errors associated with them as well.

- SEM software packages have become more and more user-friendly, and hence they are easier to use and understand.

One of the most famous SEM packages used widely in the field of psychology and genetics is OpenMx, explained in the next section.

2.2 OpenMx

OpenMx is a free, open source, platform-independent SEM package that is available as an R package on the Comprehensive R Archive Network (CRAN). OpenMx designs a statistical model to test a hypothesis using a dataset [26, 27]. The front end of OpenMx is written in R, providing users with better data management, statistics and graphics, while the backend is written in C++ for its superior computational performance.

- Model specification: OpenMx's approach to any SEM or statistical modeling analysis starts with model specification. Model specification involves determining every relationship and parameter in the model that is of interest. During this step, a theoretical model is developed mathematically so that it can be tested with observed data. In SEM, a model is said to fit perfectly when its expected covariances and expected means exactly match those in the data.

Model specification in OpenMx can be done in both matrix-style and path-style [36]. Matrix-style method specifies the matrices that are used to compute the expected covariance and mean structure of the observed and latent variables. With matrix-style method, models are specified in terms of matrix algebra. Path-style method uses functions such as mxPath() to specify the model's regression and correlation paths between observed and latent variables. A valuable feature of the path approach is that a path diagram can be generated, and this may be a mathematically complete model description if certain rules are followed [37, 38].

- Model identification: The next step is model identification. In SEM, the knowns are mainly the variances and covariances of the observed variables and the unknowns are model parameters. A model is considered identified if one and only one best value can be implied for each unknown parameter, using known information.

- Model estimation: Next is model estimation. In this step, a fit function is used to minimize the difference between model-implied and observed covariance matrices. Maximum likelihood and several variants of least squares are the most popular fit functions. However, in OpenMx users have the freedom to define their own objective functions.

- Model testing: The final step is model testing, where the model is tested to see how well the data fit the model. Different fit indices are reported in OpenMx for goodness of fit including Chi Square, Root Mean Square Error of Approximation (RMSEA), Akaike Information Criterion (AIC) and Bayes Information Criterion (BIC) [39, 40, 41, 42, 43]

In the next section, OpenMx's approach to analyze factor models is explained in detail.

2.2.1 Factor analysis

Factor analysis attempts to find a smaller set of variables (latent variables) to represent the variance-covariance of the observed variables. The aim is to preserve the relationships between the observed variables using another set of variables that are not observable, but generate the observed covariances. There are two different approaches to factor analysis:

- Exploratory factor analysis (EFA): EFA attempts to *explore* the relationship between observed variables and factors, with no specific hypothesis. The aim is to find a model that fits the data. EFA typically proceeds by eigenvalue decomposition of the observed correlation matrix. The number of factors is usually taken to be the number of eingenvalues greater than unity.

- Confirmatory factor analysis (CFA): CFA *confirms* that a subset of observed variables are represented by each factor. In CFA, specific hypotheses are modeled, and then tested to see whether the sample data confirm that model.

As an example of factor analysis, a path diagram of a 1-factor, 3-variate factor model is represented in Fig. 2.

The latent variable F represents the common variation among the observed variables V1, V2 and V3. The regression of an observed variable on a factor is interpreted as factor loading, denoted by λ_1, λ_2, λ_3. Correlations between variables are represented with two-headed arrows. Here, the only two-headed arrows are autocorrelational (from the same variable to itself); they represent the variables' variances, denoted by σ^2_{V1}, σ^2_{V2}, σ^2_{V3} and σ^2_F.

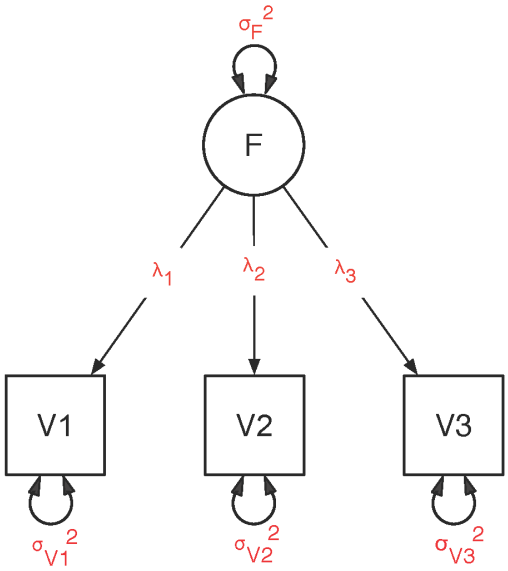


Fig. 2. Path diagram of a 1-factor, 3-variate factor model

To represent the model in matrix form, we can describe the model algebraically. Considering p observed variables, m factors and n subjects, a factor model can be written as:

$$Y_{ij} = b_i X_j + E_{ij}$$

Where $i = 1, ..., p$ variables and $j = 1, ..., n$ subjects. Y_{ij}'s represent observed variables for each subject. X_j are factor scores, which are values of each factor for each of the subjects j in the sample. b_i's are factor loadings. E_{ij} is unique for each observed variable, and explains the variability beyond that explained by common factors. Factor loadings are estimated as:

$$\Sigma_{YY} = BPB' + E$$

Where Σ_{YY} is a p by p covariance matrix of observed variables, B is a p by m matrix of factor loadings, P is an m by m covariance matrix of the common factors, and E is a p by p matrix of specific variances.

2.2.2 Ordinal data and threshold models

Variables that are not measured on continuous scale, but have limited number of categories are called categorical variables. Categorical variables that have logical ordering are referred to as ordinal variables. Ordinal data are modeled under threshold models [24, 44]. Each ordinal variable is assumed to be predicted by an underlying normally distributed latent continuous variable. Thresholds on the latent continuous variables delineate the proportion of the observations in each category. These proportions are obtained by calculating the integral over the relevant segment of the distribution. Fig. 3 illustrates thresholds on a normally distributed continuous variable underlying a 3-category ordinal variable. Because the normal distribution has no explicit integral, it has to be evaluated numerically, but this comes with limited precision. The higher the precision, the more CPU time is required, but insufficient precision can cause issues during optimization. In chapter 7, we report results of investigating precision and CPU time.

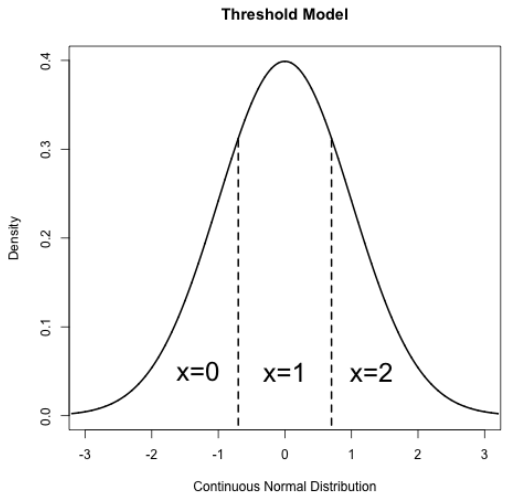


Fig. 3. Threshold model for ordinal data with three categories 0, 1, and 2.

2.3 Optimization in SEM

To highlight the role of optimization in SEM, we provide an overview of SEM approach to test a statistical model. SEM tests a hypothesis of interest in the following steps:

1. The researcher develops a theory.
2. Supporting data is gathered.
3. Alternative models are designed to test the theory.
4. Alternative models are fitted to data.
5. Fit statistics of the models are compared to find the model that best fits data.

Fitting a model to data involves calculating the model's expectation for the data which is in the form of expected covariances and means. To determine how well the data fit the model, model's expectation is compared to the data using a fit function. This comparison is done by calculating a measure of the difference between the observed (data covariance) and expected covariance matrices and means. This measure is called the function value. SEM aims at finding the values of the model's free parameters that minimize the function value. In other words, an optimizer is used to find the best set of parameters to minimize the misfit between data and model.

2.4 Fit functions in OpenMx

OpenMx offers several fit functions; in this dissertation, maximum likelihood is used almost exclusively, so these are defined in this section;

- Maximum Likelihood fit function:

Given a data set with N rows and p variables with an observed covariance matrix S, and a model with an expected covariance matrix Σ, and assuming a multivariate normal distribution, the maximum likelihood function value is defined as:

$$G^2 = (N-1)(\ln|\Sigma| - \ln|S| + tr(S\Sigma^{-1}) - p)$$

This function is effectively the difference in log-likelihood between a saturated model where the observed covariance is the expected matrix, and the expected covariance matrix. If there is a model for the means, then the maximum likelihood function value is defined as:

$$G_M^2 = (N-1)(\ln|\Sigma| - \ln|S| + tr(S\Sigma^{-1}) - p + \frac{N}{N-1}(x-\mu)'\Sigma^{-1}(x-\mu) + 1)$$

here x is the observed vector of means x, and μ is the expected mean vector. Under certain regularity conditions [45], G^2 and G_M^2 are asymptotically distributed as chi^2 with degrees of freedom equal to the difference between the number of observed statistics and the number of parameters in the model.

- Full Information Maximum Likelihood (FIML)

  FIML is used to specify a separate likelihood calculation for each row in the data matrix. This is very useful when the data has missing values.The reason is that FIML uses the raw data as input and hence can use all the available information in the data. Therefore if some of the measures are missing at a row, that row of data contributes less to the misfit, while a row with all the entries available contributes fully to the misfit computation.

  Having k variables measured for an individual i in a vector x_i, the full informa-

tion maximum likelihood function value is defined as:

$$-2\ln\mathcal{L} = \sum_{i=1}^{N}(-k\ln 2\pi + \ln|\Sigma_i| + (x_i - \mu_i)'\Sigma_i^{-1}(x_i - \mu_i))$$

The expected covariance matrix Σ and expected mean vector μ have their size adapted to the rows and columns that exist in the k variables which exist for individual i. Note that this function is not a difference between the log-likelihood of a saturated model and the fitted one, but simply the log of the likelihood itself.

CHAPTER 3

OPTIMIZATION METHODS

The standard form for a nonlinearly constrained optimization is:

$$\begin{aligned} &\min\ f(x) \\ &\text{subject to :} \\ &g(x) = 0 \\ &l_h \leq h(x) \leq u_h \\ &l_x \leq x \leq u_x\ , \end{aligned} \tag{3.1}$$

where:

$f(x) : R^n \rightarrow R$ is the objective function,

$g(x) : R^n \rightarrow R^{m_e}$ are the equality constraint functions,

$h(x) : R^n \rightarrow R^{m_i}$ are the inequality constraint functions,

l_h, u_h are the lower and upper bounds for inequalities, and

l_x and u_x are the bounds for decision variables.

The objective function and the constraints can be linear or nonlinear. The aim is to find a combination of decision variables that results in the lowest objective function value in case of minimization or highest in case of maximization, while satisfying all the constraints.

Optimization algorithms can be classified as constrained and unconstrained algorithms. Since most optimization algorithms are gradient-based, an overview of these methods are provided as well as the most popular algorithms for constrained and unconstrained methods that use gradient information for finding the solution.

3.1 Gradient-based Algorithms

Gradient-based algorithms perform in two steps: finding 1) a search direction and 2) a step size to move along this direction. This two-step process is known as line search. Using gradient information, the first step involves finding a search direction that improves the objective function. The second step involves the decision of how far to proceed along this direction before hitting the constraints. Eventually, the direction and the step size that improve the objective value, while satisfying the constraints would be chosen [46, 47].

3.1.1 Unconstrained Nonlinear Algorithms

Unconstrained optimization refers to the problem of optimizing an objective function without any constraints on the decision variables. We discuss gradient-descent and conjugate gradient methods as two gradient-based methods for unconstrained nonlinear problems in the next two sections.

3.1.1.1 Steepest Descent (Gradient descent)

This method uses the gradient vector at each point as the search direction for each iteration. The idea is that the objective function $f(x)$ decreases with the fastest rate in the direction of negative gradient:

$$d_k = -\nabla f(x_k)$$

The rate of change is given by the norm of gradient vector $||\nabla f(x)||$.

Steepest descent method zigzags in the feasible region of the problem. This behavior means that the algorithm does not choose the fastest path towards the solution. Zigzaging towards the solution results in the algorithm to faster find the

descent direction in the few first iterations, but it gets very slow towards the end of the routine when the problem is close to convergence.

To overcome this problem, the method of conjugate gradient was developed, which takes into account the information from the previous descent directions in order to avoid redundant steps in the next iterations. At each iteration k, the negative gradient vector at current iteration is combined with the previous iterations' descent directions to find a new conjugate direction for the next iteration [48, 49].

3.1.1.2 Conjugate Gradient

The method starts with the starting point x_0. The negative gradient descent direction is computed: $d_0 = -\nabla f(x_0)$. Next, line search is performed to find the step length α_0 alongside d_0. The current point is updated by $x_1 = x_0 + \alpha d_0$. At this iteration we have enough information to find a conjugate direction: $d_k = s_k + \beta_k d_k - 1$ where s_k is the descent direction at point x_k where $k > 0$: $s_k = -\nabla f(x_k)$ and β_k is:

$$(\frac{||\nabla f(x_k)||}{||\nabla f(x_{k-1})||})^2$$

The next point is then updated by $x_{k+1} = x_k + \alpha_k d_k$. The algorithm iterates until $\nabla f(x) \approx 0$ [50, 51].

3.1.1.3 Newton's Method

Newton's method is derived from a second-order Taylor series expansion of the objective function about an initial point x^0 in the following form:

$$f(x) = f(x^0) + \nabla f(x^0)^T (x - x^0) + \frac{1}{2}(x - x^0)^T H(x^0)(x - x^0) \ ,$$

where $H(x^0)$ is the Hessian matrix. Newton's method updates point estimates using the following formula:

$$x = x^0 - H(x^0)^{-1}\nabla f(x^0) \ ,$$

which is obtained by setting the differentiation of Taylor series expansion with respect to x equal to 0. Here, the search direction is obtained by $H(x^0)^{-1}\nabla f(x^0)$ and the step size is 1 [52, 52, 53].

3.1.1.4 Broyden-Fletcher-Goldfarb-Shanno

The most popular method for unconstrained optimization is Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. BFGS is an iterative algorithm where it uses the information from the previous iterations to find a new search direction that improves the objective value. Instead of calculating the exact inverse of the Hessian $H(x^0)^{-1}$, BFGS provides an approximation to the inverse of the Hessian matrix, and hence is considered a quasi-newton algorithm [54, 55, 56]. This algorithm is explained in more details in the next chapter.

3.1.2 Constrained Nonlinear Algorithms

For general nonlinear optimization with constraints, gradient-based algorithms are considered converged when the Karush-Kuhn-Tucker (KKT) conditions are satisfied. KKT conditions are first order necessary conditions to determine if a constrained local optimum has been found [57, 58]. These conditions are summarized as:

1. The optimal point (x^*) must be feasible.

2. The gradient of the Lagrangian must be zero at the optimal point:

$$\nabla f(x^*) + \sum_{i=1}^{m_i} \lambda_i \nabla h_i(x^*) + \sum_{j=1}^{m_e} \lambda_j \nabla g_j(x^*) = 0$$

3. Complementary slackness: for each inequality constraint:

$$\lambda_i h_i(x) = 0$$

Now to make sure the optimal point is not a saddle point or a maximum point, second order optimality conditions should be imposed:

$$w^T \nabla \mathcal{L}^2(x^*, \lambda^*) w > 0 \quad \forall w \in T', w \neq 0 \ ,$$

where T' is the tangent space for feasible points:

$$T' = \{v : \nabla h_j(x^*) v = 0 \quad \forall j \in \{j : \lambda_j > 0\}, \nabla g(x^*) v = 0\}$$

The tangent space gives us the set of directions in which one we move from x^* while still staying within the feasible set [59, 60].

We discuss Sequential Quadratic Programming (SQP) and Quadratic Programming (QP) as well as barrier methods (interior point methods) and primal methods as constrained optimization techniques in the next sections.

3.1.2.1 Primal methods

Primal methods start with a candidate point in the feasible region and remain in the feasible region to find an optimal solution. The aim is to find candidate point estimates that decrease the objective function at each iteration, while staying in the feasible region. So the objective function is constantly decreased and the constraints are always satisfied at each iteration. This is achieved by finding a search direction

d_k that is both descending and feasible. this means the following should hold for d_k:

$$\nabla f(x)^T d_k < 0$$

$$\nabla h_i(x)^T d_k < 0$$

$$\nabla g_i(x)^T d_k = 0$$

$f(x)^T d_k < 0$ implies descending direction for the objective function $f(x)$. $\nabla h_i(x)^T d_k < 0$ and $g_i(x)^T d_k = 0$ imply that feasibility is increased by moving in the direction tangential to the active set for $h_i(x)$'s and parallel for the $g_i(x)$'s.

Gradient projection methods are a category of primal methods. They are motivated from steepest descent algorithms. The algorithm works by moving in the direction of the negative gradient, and then projects it onto working surface formed by active constraints to find the direction of movement. This algorithm is explained more in the next chapter in the context of our developed optimizer.

3.1.2.2 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is the most popular constrained optimization technique. SQP optimizes a quadratic approximation of the objective function subject to linearly approximated constraints. Such problem is called Quadratic Programming (QP). The objective function is often converted to a merit function which is used as a criterion to determine whether a candidate point is better than the previous point estimate where the objective function is improved and the constraints are satisfied [61, 47].

Since an SQP method solves a QP at each iteration, we explain QP in more details. This will help in understanding the basis of all the three optimizers available in OpenMx.

3.1.2.3 Quadratic Programming

QP is an special case of Nonlinear Programming where the objective function is a quadratic function of decision variables and the constraints are linear functions of these variables [62, 63].

QP is the problem of finding a vector x of decision variables such that the following quadratic function is minimized:

$$\min \tfrac{1}{2}x^T Hx + f^T x$$

$$\text{subject to:}$$

$$g(x) = 0$$

$$h(x) \leq 0$$

$$x \geq 0 \ ,$$

where: $H \in R^{n \times n}$, $f \in R^n$, and $g(x)$ and $h(x)$ are respectively equality and inequality constraints where $g(x) : R^n \rightarrow R^{m_e}$ and $h(x) : R^n \rightarrow R^{m_i}$.

There are several methods to solve a QP problem, some of them being: Interior point methods, augmented Lagrangian methods, active set methods and conjugate gradient methods. An unconstrained quadratic programming problem is most straightforward to solve. Simply setting the derivative of the objective function equal to zero and solving the problem would give us the solution to the QP problem. The conjugate gradient method is the most common approach to a QP problem when the objective function is convex and there are only equality constraints. If inequality constraints are involved, interior point, augmented Lagrangian and active set methods are mostly preferred. We now describe the most general approaches for solving QP problems and explain them in more details in the remainder of this section.

Interior point or barrier methods try to reach to the optimum by passing through

the interior of the feasible region. This class of methods attempts to restrict the constraints into the objective function by creating a barrier function which limits the optimizer to iterate only in the feasible region. The algorithm starts by adding slack variables to any inequality constraints and covert them to equality constraints as the following:

$$h(x) \leq b \quad \rightarrow \quad h(x) + s = 0$$

$$s \geq 0$$

It then continues by converting the $x \geq 0$ constraint to a barrier term that is added to the objective function. The objective function $f(x)$ is modified to the following after a barrier term is added:

$$\min f(x) - \mu \sum_{i=1}^{n} \ln x_i$$

The term $\ln x_i$ is undefined for $x_i < 0$. A large value of μ gives the analytic center of the feasible region. As μ gets smaller and approaches 0, the optimal solution is found. To find the solution to the barrier problem, KKT conditions are formed and then solved by Newton's method iteratively. At each iteration, a new search direction is found. Next step would be finding the steplength α which is found using backtracking approach [64, 65].

The following section provides brief explanation of two different implementations of the SQP algorithm within OpenMx software: NPSOL and SLSQP. Our developed optimizer CSOLNP is explained in details in the methodology section.

3.2 Optimizers within OpenMx

Apart from our developed optimizer CSOLNP, two other optimization engines are available in OpenMx: NPSOL and SLSQP. These optimizers are briefly reviewed

in the following sections.

3.2.1 NPSOL

NPSOL (short for Nonlinear Programming at Systems Optimization Laboratory at Stanford ([66]) was the only available optimizer either in Mx (OpenMx's predecessor package developed in Fortran) or OpenMx since the early 1990's. NPSOL is a SQP method which finds the solution by applying line search on the augmented Lagrangian merit function. The optimizer solves a QP subproblem at each iteration. The QP subproblem is solved using subroutines from the LSSOL package which is a FORTRAN package developed at the same lab for solving constrained linear least-squares and convex quadratic programming. The solution to the QP subproblem determines the search direction. The Hessian matrix for QP subproblem is updated using BFGS. Finally the iteration ends by specifying a step length which provides sufficient decrease in the merit function. NPSOL is written in Fortran, and can be called from Fortran, C and C++ programs. NPSOL is licensed under Stanford Business Software Inc.

3.2.2 SLSQP

SLSQP (short for Sequential Least-Squares Quadratic Programming) is a SQP method which solves the QP subproblem by solving its equivalent linear least squares problem. SLSQP is available through NLopt collection [67] which is an open source library for nonlinear optimization, and is developed in C.

CHAPTER 4

APPLICATION OF CSOLNP WITHIN OPENMX ON DRUG USE [1]

CSOLNP, short for C++-based optimizer for Solving Nonlinear Programs, is a C++ implementation of the R package RSOLNP [69] and is developed as one of the optimizers available in the OpenMx package [26, 27]. Mx and OpenMx used to perform optimization using only NPSOL [66]. Recently, SLSQP from NLopt collection [67] has been added to OpenMx. Similar to NPSOL and SLSQP, CSOLNP solves nonlinear problems by applying the SQP method to a linearly constrained augmented Lagrangian objective function. While all three optimizers use SQP method, the implementations are different.

4.1 Introduction to CSOLNP

CSOLNP solves general nonlinear problems of the form presented in formula 3.1. It is an iterative algorithm which solves a QP subproblem at each major iteration. QP is a special case of Nonlinear Programming optimization where the objective function is quadratic and the constraints are linear. Each major iteration starts by solving a linearly constrained problem with an augmented Lagrangian objective function of the following form:

[1] Based on the publication [68]: Zahery, Mahsa, Hermine H. Maes, and Michael C. Neale. "CSOLNP: Numerical Optimization Engine for Solving Non-linearly Constrained Problems." Twin Research and Human Genetics 20, no. 4 (2017): 290-297.

$$\min\ f(x) - y^k g(x) + (\tfrac{\rho}{2})||g(x)||^2$$

$$\text{subject to :}$$

$$J^k(x - x^k) = -g(x^k)$$

$$l_x \leq x \leq u_x$$

The inequality constraints are converted to equality constraints by adding slack variables:

$$h(x) + s = 0, \quad s \geq 0$$

for $h(x) \leq 0$ constraints, and:

$$h(x) - s = 0, \quad s \geq 0$$

for $h(x) \geq 0$ constraints. The superscript k denotes the iteration number, and J^k is the Jacobian matrix:

$$J^k = \frac{\partial g}{\partial x}|_{x^k}$$

The original objective function is converted to an augmented Lagrangian function which incorporates a penalty term (ρ), as well as a Lagrange multiplier term (y). The penalty term is used to penalize the objective function if the current point estimation is violating the constraints, while the Lagrange multiplier is used to reduce the computational cost imposed by updating the penalty term at each iteration. The augmented Lagrangian objective function plays the role of a merit function measuring the quality of each iteration for finding a better point estimate.

The augmented Lagrangian objective function is a very common choice for a merit function ([70]). This method does not have the drawback of penalty methods in terms of having to face an ill-conditioned unconstrained problem with huge gradients.

For penalty methods, the penalty parameter is increased at each iteration to ensure that the unsatisfied constraints are penalized more severely, which eventually helps the optimizer stay close to a feasible region. Hence, optimization is achieved when the penalty parameter is increased to infinity while the term $||g(x)||^2$ is close to zero suggesting that no constraints are violated. But increasing the penalty parameter to infinity can result in increasing the condition number of the problem to infinity as the algorithm proceeds. Condition number is the sensitivity of the output of a system with respect to small errors in the input. Hence a large condition number implies that small changes in the input data can make drastic changes in the solution of a system. A system with a large condition number is called ill-conditioned and its solution is not reliable. The augmented Lagrangian method uses a Lagrange multiplier term which avoids ill-conditioning by stopping the penalty parameter from approaching infinity.

After converting the problem to an augmented Lagrangian function with linearized constraints, CSOLNP continues with a feasibility check of the current point. The region that is bounded by the constraints of the problem is called the feasible region. Any point in this region is called feasible. If the current point is feasible, CSOLNP continues with finding an optimal solution in the feasible region. Otherwise, a phase 1 Linear Programming (LP) procedure is applied to find a feasible point.

A two-phase LP technique approaches the optimal solution of a system in two phases, feasibility seeking, and optimality seeking. In phase 1, an auxiliary problem is constructed by introducing artificial variables. Artificial variables do not have any physical meaning. They are only introduced to the problem to find a feasible solution.

In phase 1, one artificial variable is added for each inequality and equality constraints. The original problem is then replaced with the sum of these artificial variables. Since artificial variables should not become part of an optimal solution to the

original problem, they have to be zero at the feasible solution, and subsequently the sum of them should equal to zero. Hence, the goal is to minimize the new objective function subject to the constraints of the original problem. If the minimum objective value is zero, then the original problem has a feasible solution. This feasible solution is used as the starting point for phase 2 where the original objective function is minimized for finding the optimal solution.

After finding a feasible point, a QP subproblem is solved to find the search direction. By approximating the objective function quadratically, the QP subproblem preserves the nonlinearity of the original objective function. Also, the constraints make the original problem easy to solve. An obvious choice of a QP subproblem would have the following format:

$$\min \nabla f(x^k)^T(x - x^k) + \tfrac{1}{2}(x - x^k)^T H(x - x^k)$$

$$\text{subject to:}$$

$$\nabla g(x^k)^T(x - x^k) + g(x^k) = 0$$

$$\nabla h(x^k)^T(x - x^k) + h(x^k) = 0 \ ,$$

where:

∇ denotes the first derivative, and H is an approximation to the hessian of the Lagrangian of the objective function $\mathcal{L}(x^k, u^k)$.

Here, the objective function is obtained by the quadratic approximation of the original objective function at the current estimate x^k, and the constraints are the linear approximations of the actual constraints at the same point estimate. After the direction $d_x^k = x - x^k$ is found, CSOLNP proceeds by finding a step length (α) that satisfies all the constraints and provides sufficient decrease in the augmented Lagrangian merit function. The next iteration starts from the new point estimate

$x^{k+1} = x^k + \alpha d_x^k$.

The solution of each QP subproblem is a search direction towards a better point estimate. Eventually, after each iteration of SQP algorithm, a better approximation, x^{k+1}, is constructed. The sequence of these approximations are hoped to converge to a solution for the original constrained nonlinear problem. A flowchart of CSOLNP algorithm is provided in Fig. 4.

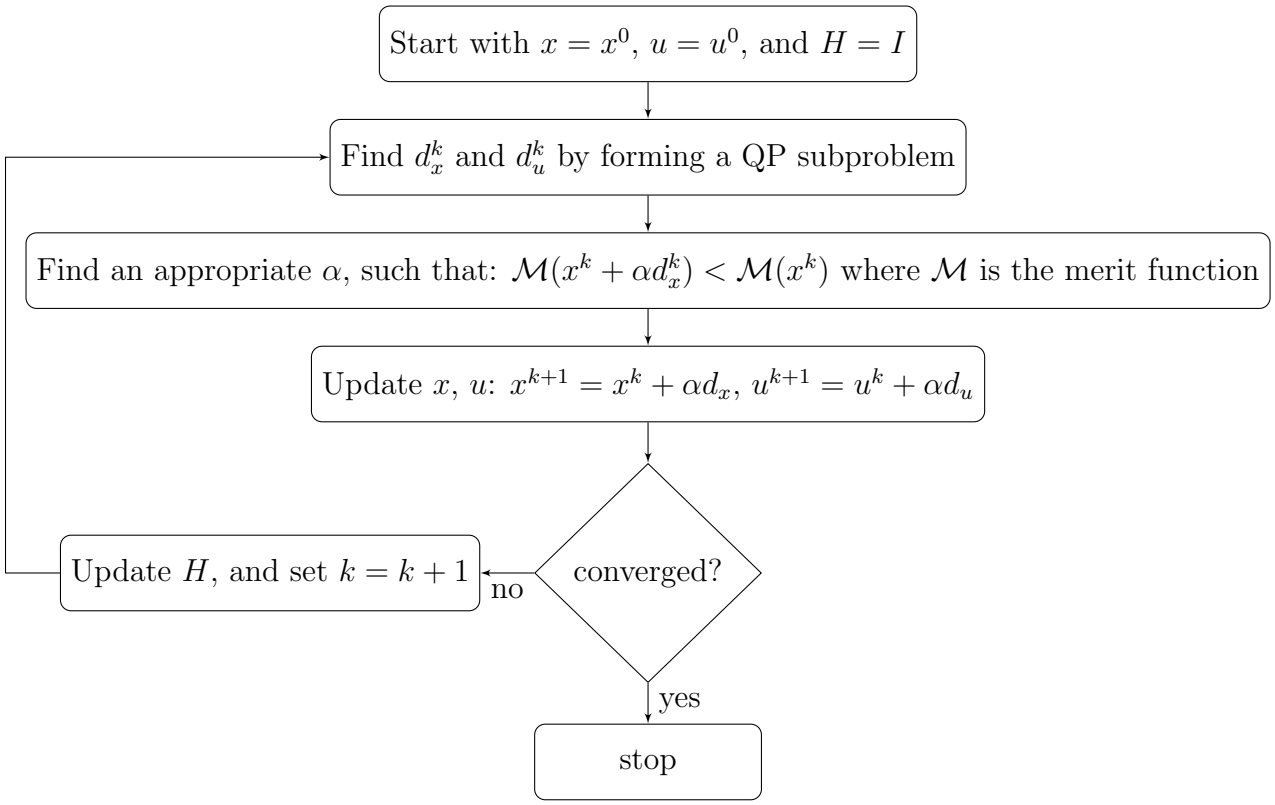


Fig. 4. A Flowchart of CSOLNP's algorithm. Starting with initial set of free variables x^0, and Lagrange multipliers u^0, the search directions d_x^k and d_u^k are found by solving a QP subproblem. Finding an appropriate step length α, CSOLNP updates the free variables and Lagrange multipliers. If the difference between the current objective value and the previous objective value is less than the optimality tolerance, the point estimates are considered to be converged, and CSOLNP reports the final set of free variables as the optimum. Otherwise, the Hessian matrix H is updated, and CSOLNP continues with the next iteration.

4.2 CSOLNP: Algorithm

We discuss details of CSOLNP's algorithm in sections ???. What occurs at each iteration of both the SQP algorithm and the QP subproblem will be explained. Next, the choice of step length α, and the convergence criteria will be discussed.

4.2.1 Initialization

Given the objective function and the constraints, CSOLNP starts with setting the following control parameters. All of these parameters except the penalty parameter can be user-defined. The default values for these parameters are provided in parentheses:

1. Maximum number of major iterations (iterations of the SQP algorithm = 400)
2. Maximum number of minor iterations (iterations of the QP subproblem = 800)
3. Penalty parameter ρ (= 1)
4. Perturbation parameter δ in finite differences method for finding the numerical gradient (= 1e-7)
5. Tolerance on feasibility and optimality (= 1e-12)

After setting these parameters, the objective function and the constraints are evaluated. Next, the Lagrange multipliers are initialized to a vector of zeros with length equal to the total number of constraints; in case there are no constraints, it is set to zero. An augmented parameter vector containing the inequality evaluations at the starting point as well as free variables' starting values is created. The corresponding Hessian matrix is initialized with the identity matrix for the first iteration.

4.2.2 Find a feasible direction

The first major iteration of the SQP algorithm starts by scaling the objective value, the constraints and the free variables. The gradients and the Jacobians are calculated numerically using the forward difference approximation method:

$$f'(x) \approx \frac{f(x+\delta) - f(x)}{\delta}$$

The default value for the perturbation parameter (δ) is 1e-7. The candidate point is then checked for feasibility. If it is not feasible, a phase 1 Linear Programming procedure is performed to start the QP algorithm with a feasible point. CSOLNP implements phase 1 with a combination of Affine Scaling Method and Gradient Projection Method in the sense that the feasibility direction is found using the Affine Scaling Method, while the step to move along this direction is found using the Gradient Projection Method.

The Affine Scaling Method is a simplified variant of Karmarkar's algorithm [71]. The basic idea behind this method is to start with a point lying in the interior (inside the boundaries) of the feasible region, and move in the direction of negative gradient descent to reduce (for minimization) the objective value at the fastest possible rate. Moving towards the direction of negative gradient descent, we might fall out of the feasible region. To have more space for reducing the objective value before hitting the boundaries of the feasible region, the Affine Scaling Method changes the coordinates of the feasible point to be placed at equal distance from the boundaries. In other words it transforms the feasible region to place the current point at its center. So for

a standard LP problem of the form:

$$\min c^T x$$

$$\text{subject to:}$$

$$Ax = b$$

$$x \geq 0$$

Where A is of size m by n. x and c are vectors of n elements and b is a vector of m elements, the Affine Scaling Method aims at moving in the direction of negative gradient of the objective function which is $-c$. Moving in this direction, the objective value is reduced, but we may violate $Ax = b$. To avoid this, $-c$ is projected into the null space of matrix A which is the set of all feasible direction vectors. This projection is:

$$P = I - A^T(AA^T)^{-1}A$$

hence, the projected gradient is Pc and the feasible direction would be $-Pc$. The last part of the Affine Scaling Method is to have the projected gradient near the center of the feasible region. This way, there is more room for further iterations of the algorithm. For this purpose, the current point x is rescaled to the point $X = D^{-1}x$, where D is a diagonal matrix of the elements of vector x. This changes the LP problem to minimizing $c^T DX$, subject to $ADX = b$ and $X \geq 0$. Subsequently the projection becomes $P = I - \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}\tilde{A}$, where $\tilde{A} = AD$. So, the projected gradient is $P\tilde{c}$ with $\tilde{c}$ being Dc. The new point is then $x^{k+1} = x^k - \alpha P\tilde{c}$, where α is the step length which is obtained by the Gradient Projection Method as the following:

$$\alpha = \begin{cases} \frac{x_i - u_i}{v_i}, & \text{if } v_i < 0 \\ \frac{x_i - l_i}{v_i}, & \text{if } v_i > 0 \end{cases}$$

Where v is the search direction obtained by the Affine Scaling Method, and l_i and u_i are respectively the lower and upper bounds for x_i. The objective value and the constraints are re-evaluated in case the starting point has been replaced with a feasible one. The merit function is also evaluated at the candidate point.

4.2.3 Solution to QP algorithm

At each iteration of the QP algorithm, a new search direction is found. this procedure starts by calculating the gradient of the merit function using the forward difference method. Next, the Hessian matrix is updated. If this is the first iteration of the QP algorithm, the algorithm considers the identity matrix as the Hessian approximation. Otherwise, a quasi-Newton approximation to the Hessian of the Lagrangian is calculated. In general, when the Hessian of the problem is dense (most of the matrix elements are nonzero), the quasi-Newton approximation can be a better choice, as it saves the computation time per iteration. CSOLNP uses the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton approximation to the Hessian at each iteration of the QP algorithm:

$$H_{k+1} = H_k + \frac{1}{y^T d} y y^T - \frac{H d d^T H}{d^T H d}$$

Where H is the Hessian matrix, d is $x^{k+1} - x^k$, and y is the change in the gradient: $g_{k+1} - g_k$.

The QP algorithm finds the search direction using the Newton's method. The search direction is obtained by Cholesky factorization of the Hessian matrix:

$$H_k d_k = -g_k$$

The Newton algorithm is considered to converge when all the constraints and free

variables' bounds are satisfied.

After finding the search direction, CSOLNP looks for a step length to move along this direction. The algorithm continues by finding a new temporary point: $x_{k+1} = x_k + d_k$. This point is not yet considered a new estimate for the next iteration. It is necessary to figure out the length of the step to move along the direction from the current point towards this temporary point. The step size is found using a binary search method: the interval between the current point and the temporary point is searched for a step size that results in the lowest merit function. The search continues until this interval is less than some tolerance.

Having the search direction and the step size, CSOLNP finds the next point estimate:

$$x_{k+1} = x_k + \alpha d_k$$

and a new iteration of the QP algorithm starts.

4.2.4 Convergence

The QP algorithm stops if the difference between the objective value at the current iteration of the QP algorithm and the previous iteration is less than the optimality tolerance. The vector of free variables, the Hessian matrix, as well as the objective value and the Lagrange multipliers (in case there are constraints) are updated, and a new iteration of the SQP algorithm (major iteration) is started.

Similar to the convergence of the QP subproblem, the problem is converged when the difference between the current objective value and the previous objective value is less than the optimality tolerance. Additionally the constraints are satisfied to within the feasibility tolerance.

4.3 Conclusion

We have developed a numerical optimization engine called CSOLNP for solving nonlinear constrained problems. CSOLNP is based on the R package RSOLNP. which implements a Sequential Quadratic Programming algorithm to solve general nonlinear problems with nonlinear objective functions and constraints.

Solving a Quadratic Programming subproblem at each iteration, CSOLNP finds a search direction that minimizes the objective function. The next step is to find an appropriate step length which best minimizes the merit function. The merit function is the augmented Lagrangian conversion of the original objective function. This is achieved by backtracking line search method where the points between the current point and a candidate point (obtained by taking a full-size step along the search direction) are tested for being the next point estimate. The point that has better minimized the merit function is considered for the next iteration of the SQP algorithm. The optimizer converges when both the optimality and feasibility tolerances are satisfied.

CHAPTER 5

PERFORMANCE COMPARISON OF CSOLNP, NPSOL AND SLSQP ON THRESHOLD AND CONTINUOUS MODELS

In all structural equation modeling (SEM) techniques, alternative models are designed to test a hypothesis of interest. These models are fitted to data to find the best set of parameters that minimizes the difference between models and data. To minimize the misfit between the models and data, SEM needs optimization, which unfortunately is not an exact science. Optimization problems have different landscapes to search, and the best search algorithm depends somewhat on the features of the landscape.

OpenMx offers three optimizers, which differ in their abilities to find the solution. We have compared the performances of CSOLNP, NPSOL and SLSQP on a variety of threshold models (this is a complex problem that is subject to local minima) as well as continuous data. Maximum Likelihood Estimation (MLE) is used as fit function for threshold models to estimate factor loadings and thresholds. The likelihood function is the joint probability of the latent continuous variables underlying the set of ordinal variables, and is defined as multivariate integration of the distribution over the intervals defined by the thresholds. For multivariate normal integration, we use Alan Genz's SADMVN routine [72]. The precision with which SADMVN computes the multivariate normal integration is varied between 1e-3 to 1e-6 in our simulations to compare the performances of CSOLNP, NPSOL and SLSQP. Other varying elements in our simulations are the number of latent variables (factors) and the sample sizes.

The models are run with 1 and 2 factors on datasets of size 1000, 10000 and 20000

samples. The performances of the three optimizers are tested on different threshold and continuous models. All the simulations in this dissertation are run on a Linux Beowulf cluster with the following computing resources:

- 42 Dell PE R630/R620 servers with CentOS/Redhat 6.7 64 bits Linux OS
- 800+ cores using Intel Xeon 56XX processors (2.67GHz to 3.4GHz)
- Total 5+TB RAM (80GB-128GB per node)
- 450TB network attached storage with 500TB backup storage
- 10TB internal disk storage (120GB-900GB per node)
- 40GB InfiniBand network connections to all nodes and storage
- Fail-over redundant master servers

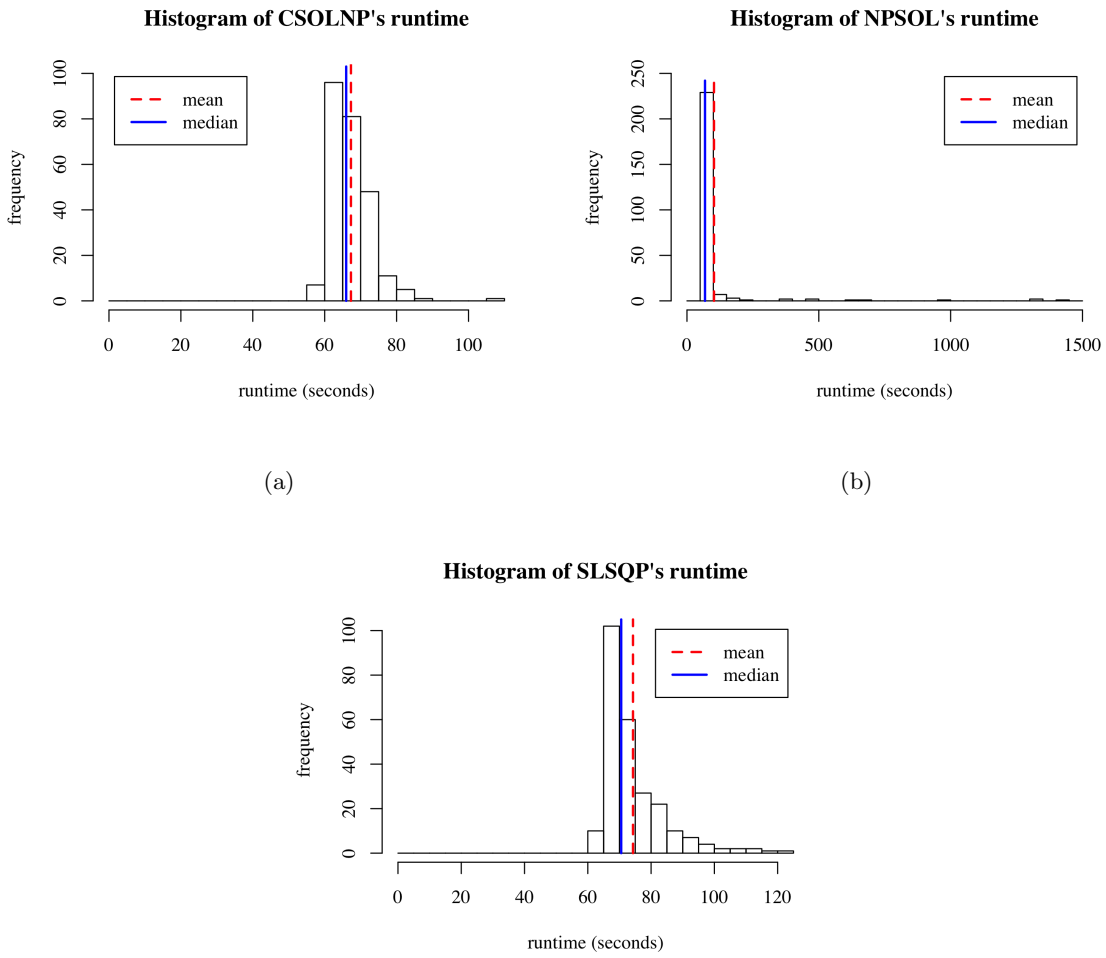
5.1 Comparison of the optimizers on threshold models

The performances of CSOLNP, NPSOL and SLSQP are compared on threshold models [44] with 5 variables and 1 to 2 factors. Performance was compared with respect to runtime of the optimizers when multivariate normal (mvn) integration absolute error tolerance is reduced from 1e-3 to 1e-7.

5.1.1 Comparison of optimizers' runtime averaged over 250 datasets

We ran threshold models with 1 and 2 factors on samples of 1000, 10000 and 20000 sizes respectively. Each model is run 250 times with different datasets. To provide a better understanding of the distribution of runtimes, we present histograms of runtime of the three optimizers on a 1 factor threshold model with sample of size 1000 and two extreme mvn absolute error tolerances, 1e-3 in Fig. 5 and 1e-7 in Fig. 6.

The blue line represents the median of the distribution and the red dashed line is the mean.



(a)

(b)

(c)

Fig. 5. Histogram of optimizers' runtime for (a) CSOLNP, (b) NPSOL and (c) SLSQP on a 1 factor threshold model with sample of size 1000 and mvn absolute error tolerance of 1e-3. The blue lines show the median of the distributions and the red dashed line represents the mean.

Looking at Fig. 5(a), we can see that for CSOLNP, the mean and median are very close to each other. The same holds true for SLSQP (Fig. 5(c)). In NPSOL

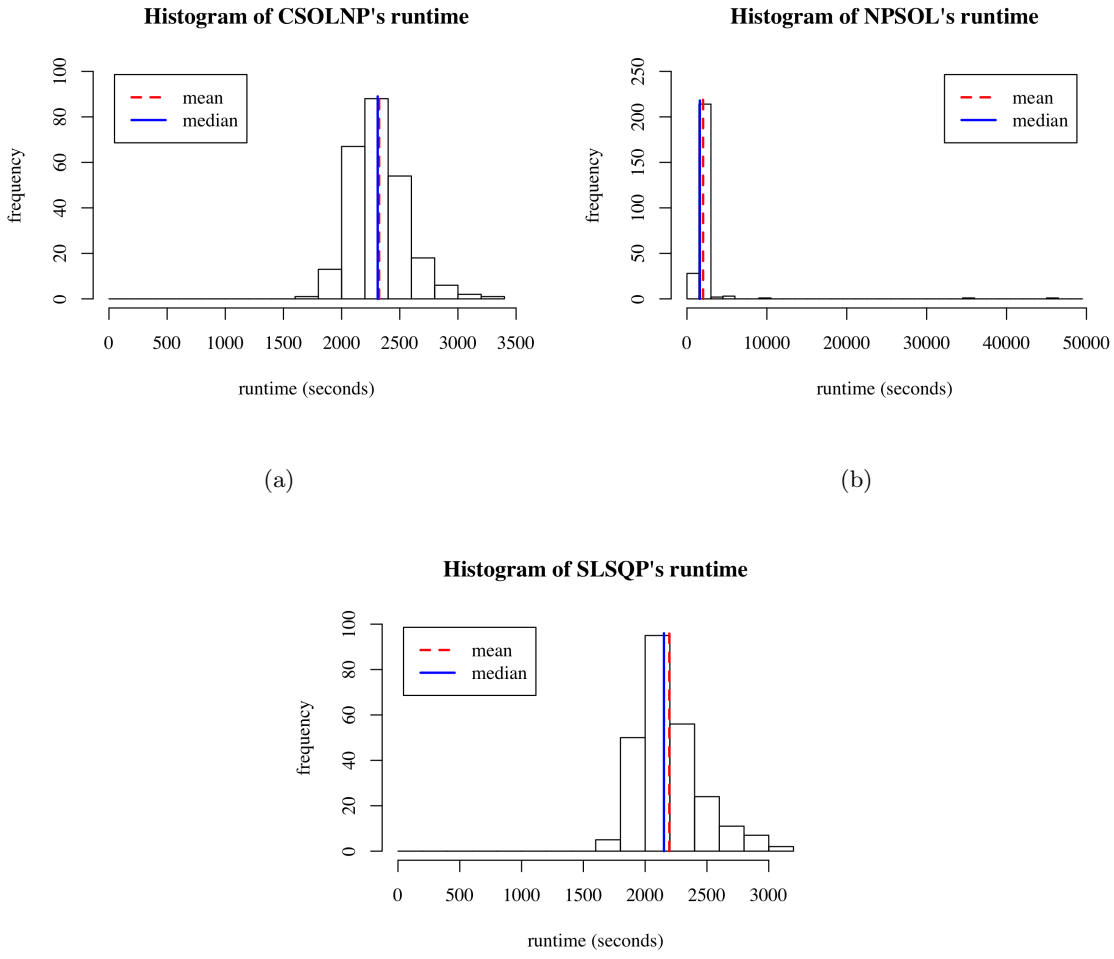


Fig. 6. Histogram of optimizers' runtime for (a) CSOLNP, (b) NPSOL and (c) SLSQP on a 1 factor threshold model with sample of size 1000 and mvn absolute error tolerance of 1e-7. The blue lines show the median of the distributions and the red dashed line represents the mean.

(Fig. 5(b)), since the distribution is right skewed and the tail is stretched away from the peak, the mean and the median may seem to be close to each other but they are not actually as close as these values are in CSOLNP and SLSQP. In table 1, we provide the exact values of the mean and the median for all the three optimizers.

Optimizer	mean	median
CSOLNP	67.301	65.977
NPSOL	102.528	68.423
SLSQP	74.281	70.557

Table 1. mean and median values of runtime for the three optimizers when run 250 times on different datasets modeled with a 1-factor threshold model with sample of size 1000 and mvn absolute error tolerance value of 1e-3.

As Table 1 suggests, the mean and the median in NPSOL are not close to each other. The same holds true for Fig. 6 where the same model is run with mvn absolute error tolerance of 1e-7. In Table 2, we present the mean and the median of the runtime distribution for this case.

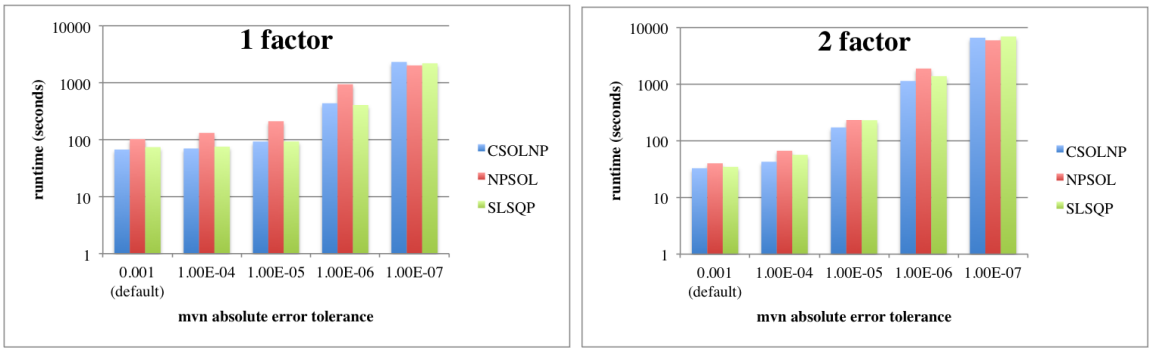
Optimizer	mean	median
CSOLNP	2322.957	2310.191
NPSOL	2019.684	1610.321
SLSQP	2195.522	2152.008

Table 2. mean and median values of runtime for the three optimizers when run 250 times on different datasets modeled with a 1-factor threshold model with sample of size 1000 and mvn absolute error tolerance value of 1e-7.

Table 2 provides similar results to Table 1 in that NPSOL's mean and median are far apart from each other, while for CSOLNP and SLSQP, these values are very close to each other. NPSOL's histograms in Fig. 5(b) and Fig. 6(b) are also very similar, both skewed to the right with long tails. This suggests that in NPSOL, the median is a better measure for central tendency of the distribution, but the stretched tail in Fig. 5(b) and Fig. 6(b) suggests that the variance around the median is very high.

This makes NPSOL an unreliable choice of optimizer. For CSOLNP and SLSQP, the mean and the median are very close to each other and the distribution is less skewed. These two optimizers are more reliable as they provide consistent results with less variance.

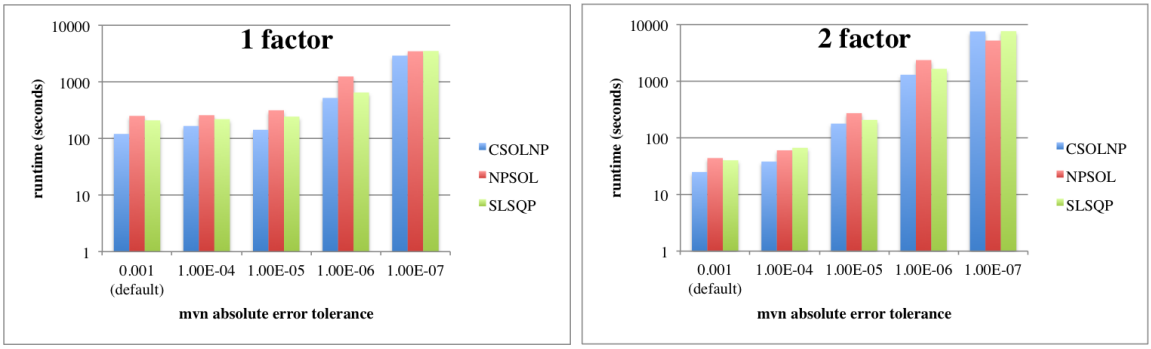
based on the histograms, we illustrate the mean value of the optimizers' runtime for 250 simulations in logarithmic scale on threshold models with 1 and 2 factors on samples of size 1000 in Fig. 7, 10000 in Fig. 8 and size 20000 in Fig. 9. The mean value is presented for different mvn absolute error tolerances ranged from 1e-3 to 1e-7. We also compared the final objective values at which the three optimizers stop. The relative difference between the final objective values are less than 0.5%, and hence not shown.



(a) (b)

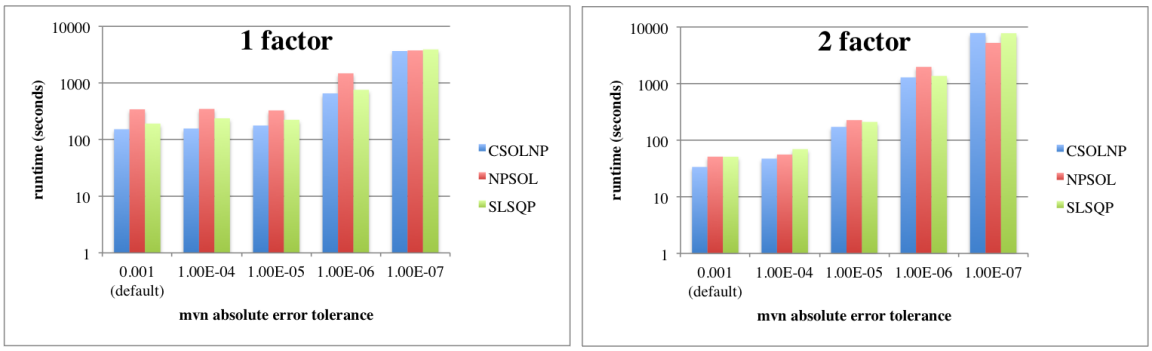
Fig. 7. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 1000 with mvn absolute error tolerance of 1e-3 to 1e-7. Runtimes are averaged over 250 data simulations.

In all the threshold models shown in this section, the optimizers take more time as the requested numerical integration precision for absolute error tolerance is reduced from 1e-3 to 1e-7. This is expected as smaller absolute error tolerance dictates the accuracy of numerical integration. Since there is no closed form solution for the



(a) (b)

Fig. 8. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 10,000 with mvn absolute error tolerance of 1e-3 to 1e-7. Runtimes are averaged over 250 data simulations



(a) (b)

Fig. 9. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 20,000 with mvn absolute error tolerance of 1e-3 to 1e-7. Runtimes are averaged over 250 data simulations.

integration of the multivariate normal distribution, it is carried out numerically to a particular degree of numerical precision. However, the more precise the integral calculation, the longer it takes. In these tests, NPSOL is in general the slowest optimizer, and CSOLNP is the fastest.

5.1.2 Comparison of optimizers' runtime averaged over 250 different starting values

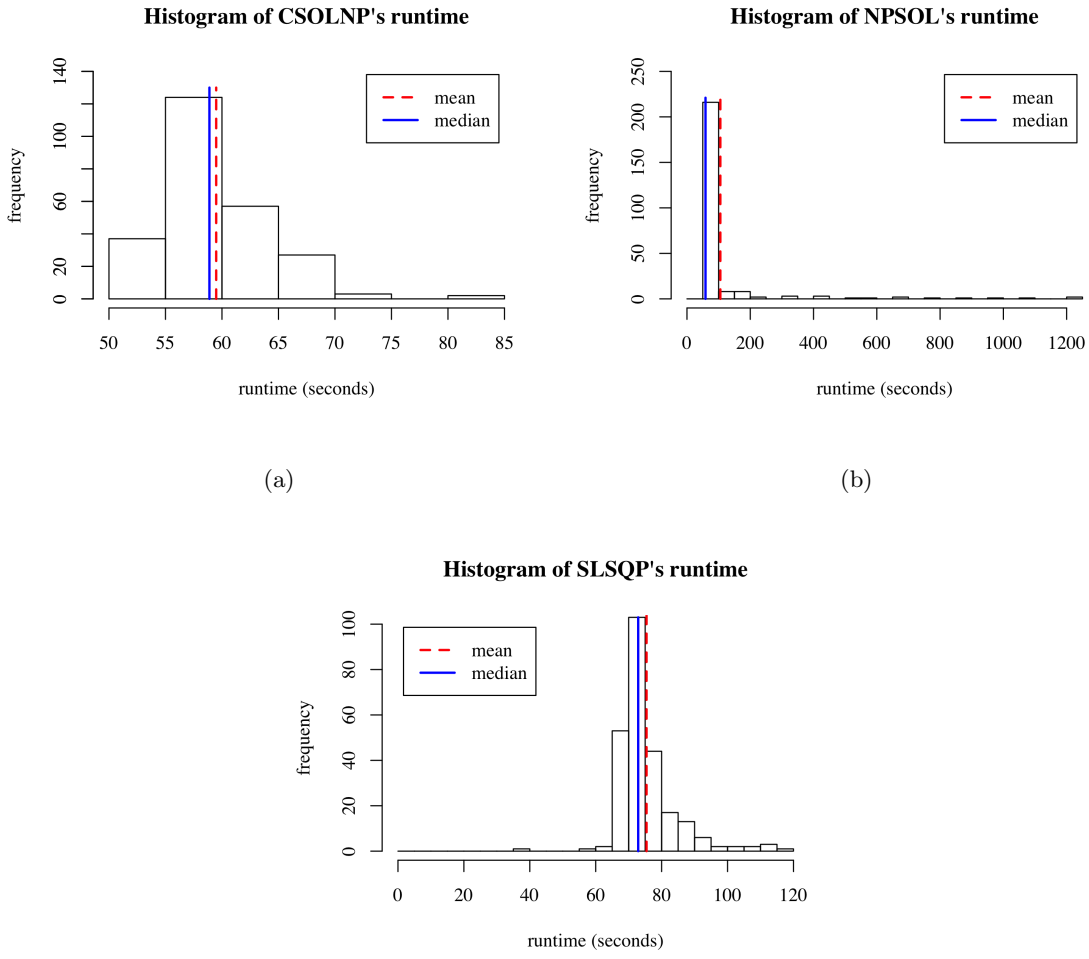
We ran the same 1 factor and 2 factor threshold models with sample sizes 1000, 10000, and 20000 with 250 different starting values and compared their runtime. Starting values can easily affect the optimizers' trajectory to find the solution, and hence can provide important information about optimizers' performance.

We provide two histograms of runtime of the three optimizers in Fig. 10 and 11. The histograms are obtained by running a 1-factor threshold model on a sample of size of 1000 with mvn absolute error tolerance of 1e-3 in Fig. 10, and 1e-7 in Fig. 11. The blue line represents the median of the distribution and the red dashed line is the mean. Similar to the previous section's results, NPSOL's distribution is skewed and has a stretched tail in Fig. 10. Table 3 shows the values of the mean and the median for all the optimizers in this case. In this table, we see that the mean is almost twice the median for NPSOL, while for CSOLNP and SLSQP, these values are almost the same.

Optimizer	**mean**	**median**
CSOLNP	59.48888	58.8894
NPSOL	105.514	58.585
SLSQP	75.42213	72.89215

Table 3. mean and median values of runtime for the three optimizers when run 250 times with different starting values. The model is a 1-factor threshold model run on a sample of size 1000 and mvn absolute error tolerance value of 1e-3.

We choose the mean as a measure for central tendency of the distribution of optimizers' runtime. Fig. 12, 13 and 14 illustrate the results for threshold models with



(a)

(b)

(c)

Fig. 10. Histogram of optimizers' runtime for (a) CSOLNP, (b) NPSOL and (c) SLSQP when run on threshold model with 1 factor and absolute error tolerance of 1e-3 on sample of size 1000. The distribution is formed by running the models 250 times with different starting values. The blue lines show the median of the distributions and the red dashed line represents the mean.

1 and 2 factors on samples of 1000, 10000 and 20000 sizes when mvn absolute error tolerance is reduced from 1e-3 to 1e-7. The results are averaged over 250 replications of the same models with different starting values.

CSOLNP appears to be a better choice of optimizer for threshold model anal-

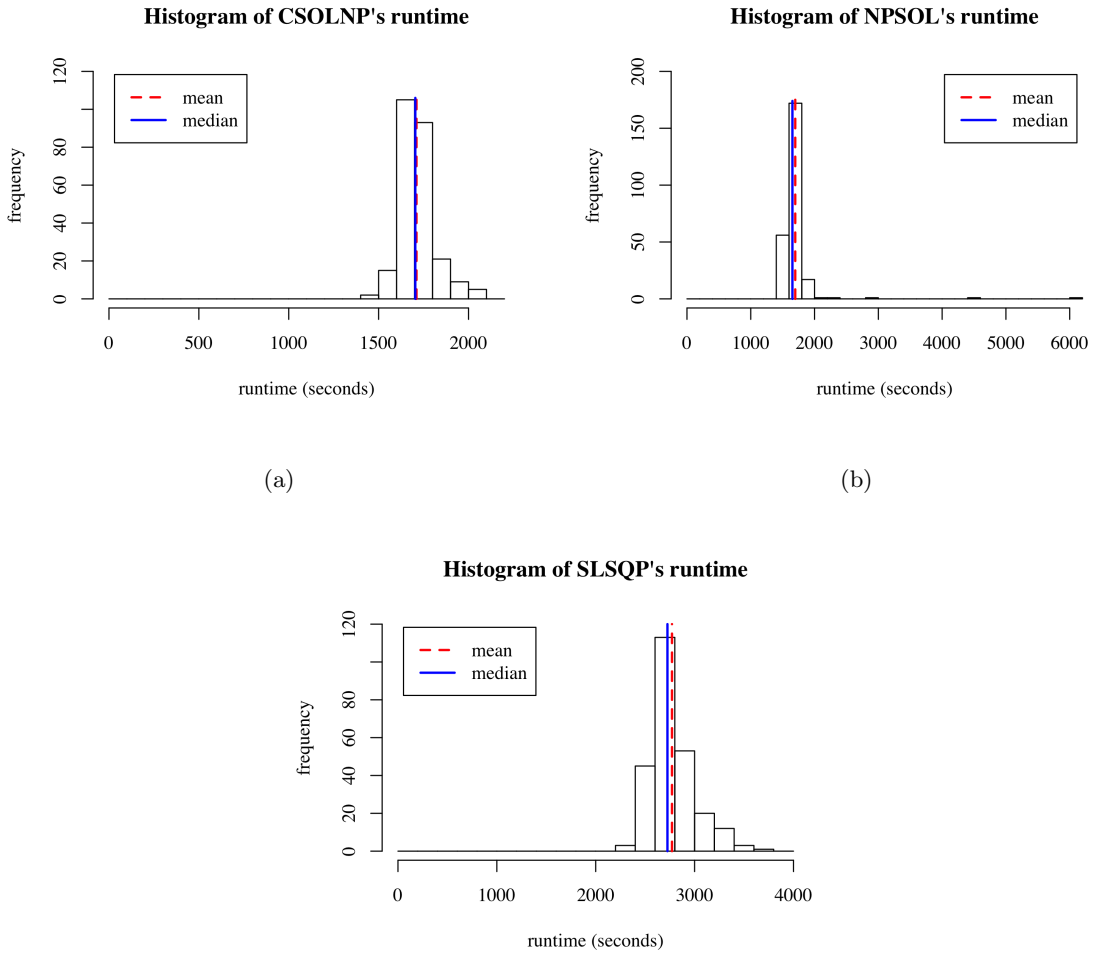
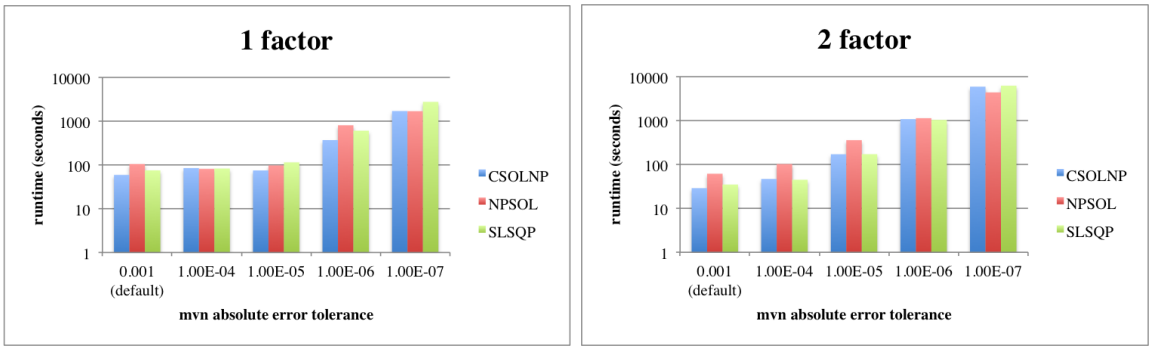


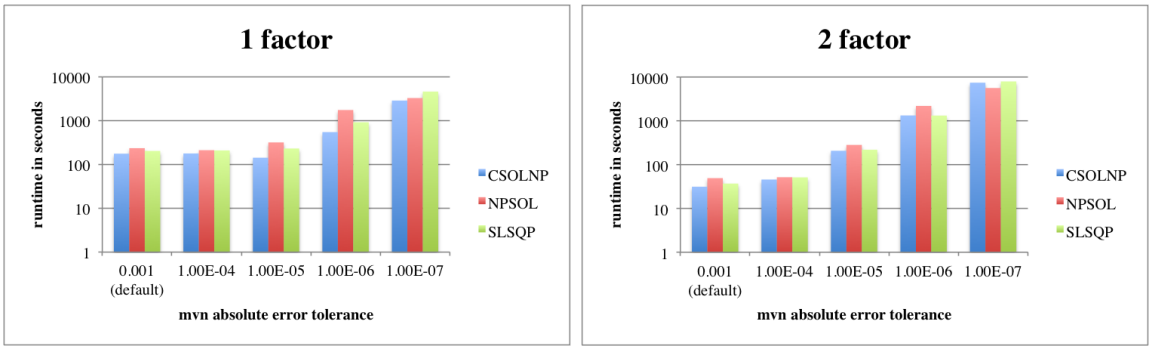
Fig. 11. Histogram of optimizers' runtime for (a) CSOLNP, (b) NPSOL and (c) SLSQP when run on threshold model with 1 factor and absolute error tolerance of 1e-7 on sample of size 1000. The distribution is formed by running the models 250 times with different starting values. The blue lines show the median of the distributions and the red dashed line represents the mean. In all the cases, the mean and the median are very close to each other.

ysis, due to its faster performance, and greater reliability. We see NPSOL as an inconsistent optimizer with largest standard deviations in most of the cases. The histograms of optimizers' runtime proves the inconsistent behaviour of NPSOL as the



(a) (b)

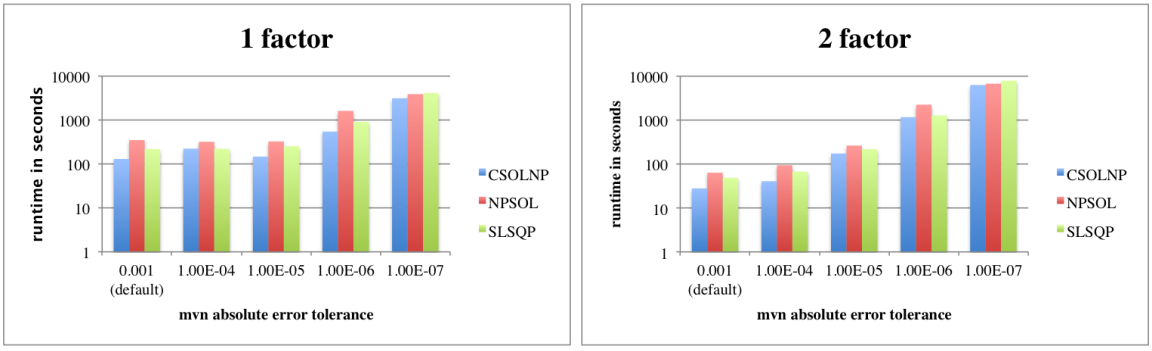
Fig. 12. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 1000 with mvn absolute error tolerance reduced from 1e-3 to 1e-7. Runtimes are averaged over 250 replications of the models with different starting values.



(a) (b)

Fig. 13. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 10000 with mvn absolute error tolerance reduced from 1e-3 to 1e-7. Runtimes are averaged over 250 replications of the models with different starting values.

optimizer's runtime distribution is heavily skewed with long tail. SLSQP takes longer than CSOLNP in all the cases except some rare cases where absolute error tolerance is 1e-7. Since the default value for mvn absolute error tolerance in OpenMx is 1e-3,



(a) (b)

Fig. 14. Runtimes of CSOLNP, NPSOL and SLSQP in logarithmic scale for threshold models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for a sample of size 20000 with mvn absolute error tolerance reduced from 1e-3 to 1e-7. Runtimes are averaged over 250 replications of the models with different starting values.

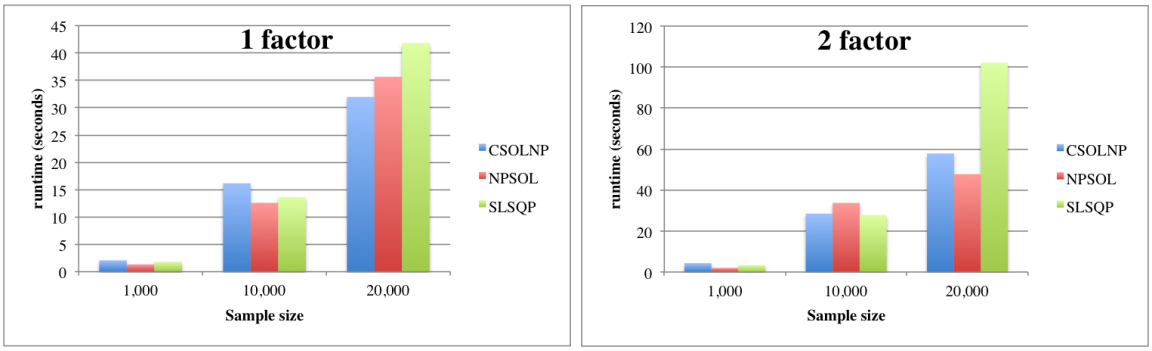
we think it is safe to conclude that CSOLNP is a better choice of optimizer for ordinal data and threshold model analysis.

5.2 Comparison of optimizers on continuous models

Next, we compared the runtime performance of CSOLNP, NPSOL and SLSQP when fitting a factor model to continuous data. The analyses involve data from 5 variables with 1 or 2-factor model. Since multivariate normal integration is not required to calculate the likelihood for continuous data, the mvn parameter value is irrelevant.

5.2.1 Comparison of optimizers' runtime over 250 different datasets

Fig. 15 illustrates runtime of the three optimizers on continuous models with 1 and 2 factors on samples of 1000, 10000 and 20000 sizes respectively. The results are averaged over 250 simulations. The final objective values are almost identical for the



(a) (b)

Fig. 15. Runtimes of CSOLNP, NPSOL and SLSQP for continuous models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for samples of sizes 1000, 10000 and 20000. Runtimes are averaged over 250 different data simulations.

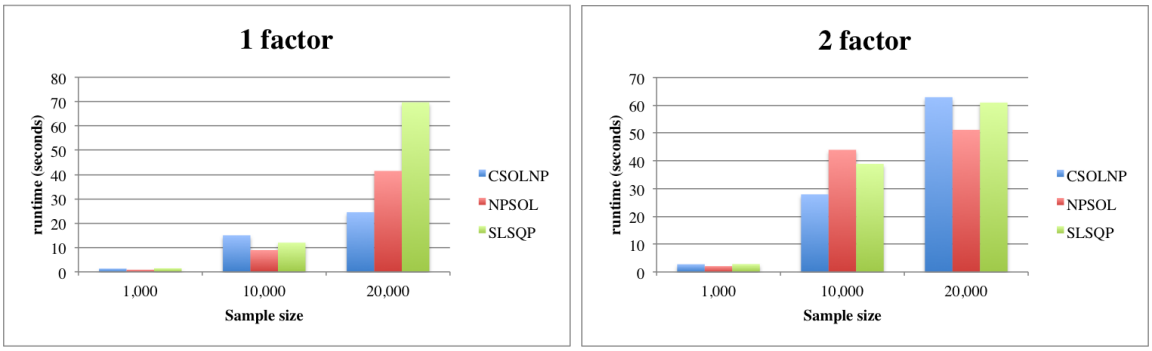
optimizers and hence not shown.

5.2.2 Comparison of optimizers' runtime over 250 different starting values

Fig. 16 illustrates runtime of the three optimizers on continuous models with 1 and 2 factors on samples of 1000, 10000 and 20000 sizes respectively. Results are averaged over 250 replications of models with different starting values. The final objective values are almost identical for the three optimizers and hence not shown.

5.3 Conclusion

We compared the performance of the three optimizers available in the package OpenMx over threshold and continuous models. We ran the optimizers on threshold models with 5 variables and 1 or 2 factors with samples of size 1000, 10000 and 20000. We compared the runtime of the optimizers as the multivariate normal integration precision is increased from 1e-3 to 1e-7. In one scenario, we ran such models over 250



(a) (b)

Fig. 16. Runtimes of CSOLNP, NPSOL and SLSQP for continuous models with (a) 5 variables and 1 factor, and (b) 5 variables and 2 factors for samples of sizes 1000, 10000 and 20000. Runtimes are averaged over 250 different starting values.

different datasets. In another scenario, we ran the models with 250 different starting values. For both scenarios, we compared the mean value of the distributions. We also compared the optimizers' runtime on continuous models with the same number of variables and factors on the same sample sizes.

For threshold models, CSOLNP performs faster than the other two optimizers. We compared the optimizers' runtime distribution and the distributions' mean and median and found CSOLNP and SLSQP to be more reliable and consistent than NPSOL.

For continuous models, we compared the mean and median for runtime distributions of the optimizers for 250 different dataset as well as 250 different starting value scenarios. We did not find any difference between these values for any of the optimizers. Also, the optimizers' runtimes differ only very slightly. No optimizer consistently outperforms the others for continuous models.

CHAPTER 6

APPLICATION OF CSOLNP WITHIN OPENMX ON DRUG USE [1]

Using the newly developed optimization package, we have studied a very large drug abuse dataset collected in Sweden from over one million people to investigate the genetic and environmental contributions in liability to drug use. These cases are ascertained from medical and criminal registries. Table 4 provides a sample of how this dataset looks like.

	Med1	**Crime1**	**Med2**	**Crime2**
1	0	0	0	0
2	1	1	0	0
3	1	0	0	1
4	0	1	1	1
5	1	1	0	1

Table 4. A sample of the Swedish dataset on drug abuse. The dataset consists of four features being medical and criminal records for each pair of the twins, siblings or half-siblings. Features are binary variables being 0 for no medical or criminal record and 1 for presence of medical or criminal records.

We used OpenMx to model the contributions of genetic and environmental risk factors to liability to drug abuse under threshold models [74]. Additive genetic (A), shared (C) and unique (E) environmental factors were included to our model.

[1]Based on the publication [73]: Maes, Hermine H., Michael C. Neale, Henrik Ohlsson, Mahsa Zahery, Paul Lichtenstein, Kristina Sundquist, Jan Sundquist, and Kenneth S. Kendler. "A Bivariate Genetic Analysis of Drug Abuse Ascertained Through Medical and Criminal Registries in Swedish Twins, Siblings and Half-Siblings." Behavior genetics 46, no. 6 (2016): 735-741.

6.1 Statistical analysis

We used a correlated factor model to estimate the contribution of shared and unique genetic and environmental factors between the medical and criminal assessments for drug abuse. Both quantitative and qualitative sex differences in the sources of differences in drug abuse were tested. quantitative analysis across males and females determines whether the estimates of heritability are the same across the sex whereas qualitative analysis focuses on whether the same genetic or environmental factors contribute to both males and females. We tested the following six models:

1. No consideration of quantitative or qualitative differences in ACE components of variances across males and females. i.e. the A, C and E components were constrained to be equal across males and females.

2. Consideration of quantitative sex differences in ACE components of variances. i.e. A, C, and E were freely estimated in both males and females. On the other hand, the correlations between these latent factors were constrained.

3. Consideration of qualitative and quantitative sex differences in ACE components with male-specific genetic factors.

4. Consideration of qualitative and quantitative sex differences in ACE components with female-specific genetic factors.

5. Consideration of qualitative and quantitative sex differences in ACE components with male-specific shared environmental factors.

6. Consideration of qualitative and quantitative sex differences in ACE components with female-specific shared environmental factors.

These models were fitted with a range of acceptable starting values for the A and C parameters. We used CSOLNP as OpenMx's optimization engine for finding the model with the lowest -2 log-likelihood. Table 5 illustrates goodness-of-fit statistics for fitting bivariate ACE models to data from medical and criminal records. Best fitting model is highlighted in bold.

Model	**os**	**ns**	**ep**	**df**	**-2ll**	**AIC**
hom	5,482,904	1,370,726	19	5,482,885	1,097,748.2	-9,868,021.8
qn	5,482,907	1,370,726	26	5,482,881	1,096,939.8	-9,868,822.2
qlAms	5,482,907	1,370,726	29	5,482,878	1,096,890.7	**-9,868,865.3**
qlAfs	5,482,907	1,370,726	29	5,482,878	1,096,918.0	-9,868,838.0
qlCms	5,482,907	1,370,726	29	5,482,878	1,096,909.8	-9,868,846.3
qlCfs	5,482,907	1,370,726	29	5,482,878	1,096,912.7	-9,868,843.4

Table 5. Goodness-of-fit statistics for bivariate ACE models fitted to drug abuse ascertained from medical and criminal records, in designs including twins, full siblings and half siblings. os: number of observed statistics, ns: number of pairs, ep: number of estimated parameters, df: degrees of freedom, -2ll: minus twice the log-likelihood of the data, AIC Akaike information criterion, hom: homogeneity model without qualitative or quantitative sex differences in ACE sources of variance, qn: quantitative sex differences in ACE sources of variance, ql: qualitative and quantitative sex differences in ACE sources of variance, with either sex-specific genetic factors in males (Ams) or females (Afs), or sex-specific shared environmental factors in males (Cms) or females (Cfs)

Table 6 illustrates the parameter estimates. The best fitting model included male-specific genetic factors, in addition to the gender-common genetic factors which were estimated separately for males and females. The variance component estimates for data from medical records (DAM) in males were: additive genetic, 59%; shared

	Amc+Ams=Am	**Cm**	**Em**	**Af**	**Cf**	**Ef**	**Rgmf**
DAM	0.35+0.24=0.59	0.13	0.28	0.52	0.07	0.41	0.82
DAC	0.21+0.45=0.66	0.20	0.15	0.59	0.15	0.26	0.39
DAM-DAC covariance	0.62*/0.76**	1.00	0.40	0.61	1.00	0.38	

Table 6. Estimates of proportions of variance and covariance accounted for by additive genetic, shared and unique environmental sources for drug abuse ascertained from medical (DAM) and criminal (DAC) records, in designs including twins, full siblings and half siblings. Amc gender-common additive genetic contributon in males, Ams, male-specific additive genetic contribution, Am additive genetic contribution in males, Af additive genetic contribution in females, Cm shared environmental contribution in males, Cf shared environmental contribution in females, Em unique environmental contribution in males, Ef unique environmental contribution in females, rgmf genetic correlation across males and females. *Covariance due to genetic factors in common with females (Amc); **covariance due to male-specific genetic factors

environment 13%; and unique environmental influences including measurement error, 28%. For females these estimates were respectively 52, 7 and 41%. For data from criminal records (DAC), these estimates were, respectively, 66, 20 and 15% for males and 59, 15 and 26% for females. We also found out that genetic factors were substantially shared between DAM and DAC in both males (62% for genetic factors in common with females, and 76% for male-specific genetic factors) and females (61%). Shared environmental factors were completely shared in both sexes while specific environmental were moderately shared (40 and 38% respectively for males and females). Estimates of the genetic correlations across males and females were 0.82 for DAM and 0.39 for DAC. This suggests that mostly the same genetic factors contributed to DAM in males and females, while for DAC, different genetic factors contributed in both sexes.

6.2 Comparing optimizers' runtime on the Swedish Data

We have compared runtime of optimizers on the swedish data as well, and the results are consistent with the synthetic data discussed in chapter 4. Fig. 17 represents the runtime of CSOLNP, NPSOL and SLSQP on swedish drug abuse dataset modeled by a threshold model with 1 factor and 4 variables. The runtimes are compared across different multivariate normal absolute error tolerance values, from 1e-3 to 1e-7.

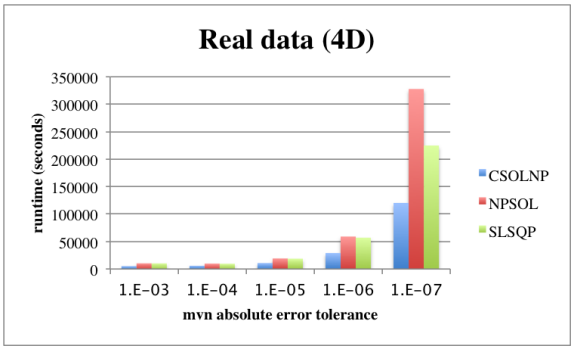


Fig. 17. Runtime of optimizers on real data with a threshold model with 1 factor and 4 variables.

6.3 Conclusion

In this study we applied bivariate genetic structural equation modeling to a very large population-based drug abuse dataset collected in Sweden from over one million people, and analyzed all possible pairs of twins (MZ, DZ, full and half-siblings) to find contributions of genetic and environmental factors in liability to drug abuse. The results showed substantial heritability and moderate contributions of shared environmental factors to drug abuse. Both of these factors were higher in males versus females, and higher for drug abuse ascertained through criminal than medical records.

In the end, we presented a comparison of the optimizers' runtimes on this dataset. CSOLNP converged faster than NPSOL and SLSQP in all five scenarios with mvn

absolute error tolerance decreasing from 1e-3 to 1e-7.

CHAPTER 7

PROFILING OPTIMIZERS AND MEMORY CONSUMPTION COMPARISON

In order to figure out the reasons behind CSOLNP's superior performance, we aimed at profiling all the three optimizers to find the bottlenecks of each algorithm, and see whether we can further improve CSOLNP's performance.

7.1 Valgrind

A profiler is a tool to optimize a program by analyzing the program and reporting on its resource usage such as memory, CPU cycles, and so on. The profiling output provides details about the patterns and peaks of resource consumption. This enables the user to determine if there is a bottleneck in the code, and if so, where the problem is concentrated.

The standard C profiler is gprof, the gnu profiler. Intel VTune is another C profiler. Both can be run on a C/C++ program. R has profiling tools as well, but they cannot be extended into other languages called within R. We found Valgrind to be the only tool suitable for our purposes. It can be called on our R code and can be extended to the C++ backend of OpenMx.

Valgrind is a set of multipurpose programming tools used as a debugger, memory mismanagement detector as well as a profiler and a memory consumption analyzer. It is an excellent free tool suite available on Linux and OS X's platforms for all programming languages, although the main aim of developing this tool is for C and C++ programs due to being more prone to memory bugs.

Valgrind's default tool is Memcheck which is used for detecting memory leaks by checking memory usage such as calls to malloc and free or new and delete in C++. The following are amongst the most popular errors detected by Memcheck:

- use of uninitialized memory
- reading or writing after an allocated block
- reading or writing memory after a block is freed
- forgetting to free a pointer
- releasing a memory block that is already released

Other tools contained in Valgrind wrapper are as the following:

- Cachegrind: a cache profiler collecting statistics about cache misses by simulating a machine with a split L1 cache and a unified L2 cache. Cachegrind is capable of identifying which lines of the source code have caused cache misses. Basically it monitors:
  - L1 instruction cache reads and misses
  - L1 data cache reads and read misses, writes and write misses
  - L2 unified cache reads and read misses, writes and writes misses.
- Callgrind: An extension to Cachegrind which analyzes function calls. This is the tool we used to profile our three optimizers' performances. We explain Callgrind in more details in the following section.
- Massif: Heap memory profiling tool. Massif analyzes memory consumption throughout the program. It gives an overview of the memory used over time. We used Massif to compare memory usage of our three optimizers.

– Helgrind: Thread debugging tool that analyzes the code for presence of different synchronization errors in programs that use POSIX Threads.

In the following section, we provide details about Valgrind's profiling tool Callgrind followed by the results of running Callgrind on our drug abuse problem.

7.2 Callgrind

Callgrind is a profiling tool that counts function calls and the CPU instructions executed within each call and produces a tree of calls which helps analyze the performance of the program in use. To profile with Callgrind, Valgrind is called as: valgrind –tool=callgrind code callgrind_arguments

Since Memcheck is the default tool for Valgrind, in order to use Callgrind, Valgrind has to be called with Callgrind as the preferred tool. With the above command, the program starts to run (of course more slowly than without being triggered by Valgrind). At the end, a summary of the total number of instructions is provided to the user. Fig. 18 provides a screen shot of how this summary looks like:

```
==22417== Events    : Ir
==22417== Collected : 7247606
==22417==
==22417== I   refs:      7,247,606
```

Fig. 18. summary of Callgrind output

The number 22417 is the process id. Ir stands for Instructions Read. Ir counts are the number of instructions executed at the machine level (the number of assembly instructions). Fig. 18 suggests that about 7 million Instructions are read when running this program.

Callgrind measures the cost of each function as the total number of events occurring within that function. Cost of functions can be exclusive or inclusive. By default,

the call counts are exclusive. Exclusive cost of a function includes only the time spent in that function (in terms of Ir) and not in the functions that it calls. Inclusive cost of a function includes the time spent in all functions called within that function, whether directly or indirectly. Exclusive costs help find Highly-traffic functions by looking for functions with the highest counts.

7.2.1 Callgrind results

We ran Callgrind on Swedish drug abuse dataset, which has data on 1.4 million twins, half- and full-siblings. Four binary variables are analyzed, being criminal and medical records for each pair. We ran a one-factor threshold model on this dataset with different multivariate normal absolute error tolerance values from 1e-3 to 1e-7, and the results are provided in Fig. 19

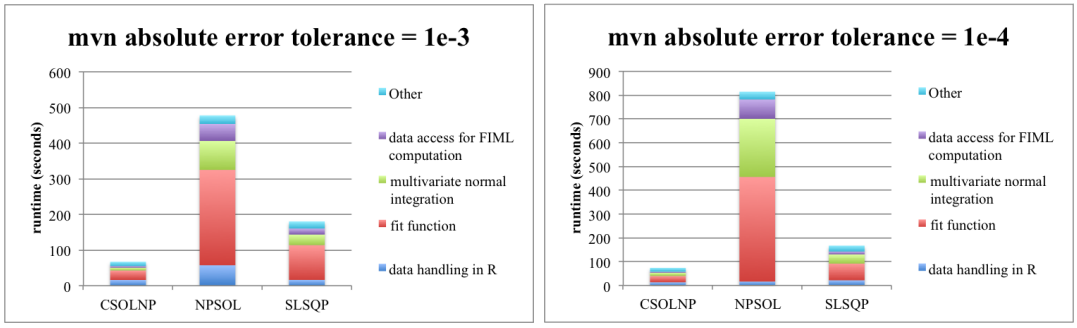
Fig. 19 shows the runtime of each optimizer in terms of the four most costly routines called within each run from the front-end in R to the back-end in C++ and back to the front-end. These routines are:

- Data handling in R: Handling data objects in the front-end at the beginning and the end of the program. This routine consists of serialization of R objects to C++ objects and vice versa.

- Computing the objective function: Full Information Maximum Likelihood fit function is computed

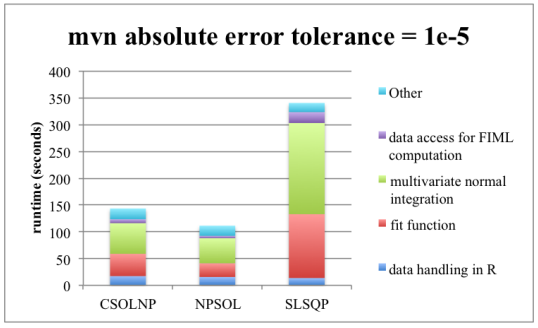
- Data access for objective function calculations: Handling missing data and filtering out the missing values

- Multivariate normal integration: A FORTRAN subroutine for calculating multivariate normal integration

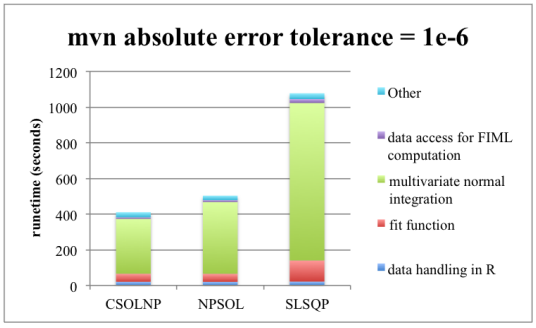


(a)

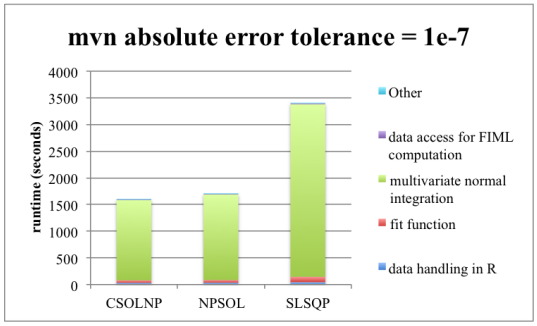
(b)



(c)



(d)



(e)

Fig. 19. profiling results for CSOLNP, NPSOL and SLSQP with multivariate normal absolute error tolerance of (a) 1e-3, (b) 1e-4, (c) 1e-5, (d) 1e-6 and (e) 1e-7.

– other

Multivariate normal integration is a subroutine called within objective function evaluation. We have considered the cost of function evaluation exclusive of this sub-

routine. For mvn absolute error tolerance of 1e-3 and 1e-4, objective function computation is the most costly function. As the mvn absolute error tolerance is increased, it is the multivariate normal integration routine that takes the most time. This is expected as increasing the precision of numerical integration increases the runtime. In general, since multivariate normal integration is done at each objective function evaluation, we can say the function evaluation routine is the most expensive task.

Multivariate normal integration is a subroutine called within objective function evaluation. We have considered the cost of function evaluation exclusive of the cost of this subroutine. For mvn absolute error tolerance of 1e-3 and 1e-4, objective function computation (excluding the numerical integration) is the most costly function. As the mvn absolute error tolerance is reduced (i.e. precision goes up), it is the multivariate normal integration routine that takes the most time. In general, since multivariate normal integration is done at each objective function evaluation, we can say the function evaluation routine is the most expensive task for all values of mvn, with multivariate normal integration taking up a larger proportion time of the function evaluation routine as mvn absolute error tolerance is decreased.

CSOLNP is the fastest optimizer except for the case where mvn absolute error tolerance is 1e-5. For this value of mvn absolute error tolerance, NPSOL is the fastest. Just as with the previous results for simulated data, NPSOL behaves inconsistently. For larger values of mvn absolute error tolerance (1e-3 and 1e-4), it takes the longest, while for the smallest ones (1e-5 to 1e-7) it converges faster.

We believe the main reason behind the faster performance of CSOLNP is due to the numerical gradient computation routine for this optimizer. CSOLNP uses a forward difference approach for finding numerical gradients, while SLSQP uses central difference approach. NPSOL uses forward difference by default, but in case the optimizer cannot find the optimum, it switches to central difference. This is the

exact case for NSPOL with tolerance values of 1e-3 and 1e-4. The optimizer failed to reach to the same minimum as the other two optimizers, even after switching to the central difference approach.

7.3 Massif results

For the purpose of comparing memory consumption of the optimizers, we used Valgrind's heap profiler tool, Massif. Massif registers the memory used by the code at different times. It can be used to identify where and when the program allocates memory on the heap with the aim to either reduce the amount of allocated memory or to release it earlier in the program. To use Massif, Valgrind should be called as: valgrind –tool=massif program

Massif provides a graph of the memory consumption similar to Fig. 20. Massif takes a snapshot of memory usage at certain points during runtime of the program. The bars represent these snapshots. In Fig. 20, Massif has taken 52 snapshots, one per heap allocation/deallocation. Massif reduces the number of snapshots as the program runs longer. The default value for maximum number of snapshots is 100. If the number of snapshots exceeds this number, Massif will throw away the old snapshots in favor of the new ones. Fig. 20 shows memory consumption of CSOLNP when run on the Swedish drug abuse dataset (see 6) with a one-factor threshold model and multivariate normal absolute error tolerance value of 1e-3. The figure shows CSOLNP takes 752.0 MB of memory. The x-axis is the number of bytes allocated/deallocated on the heap. There was no difference between optimizers' memory consumption for different values of mvn absolute error tolerance, and hence the Massif graphs for those cases have not been shown.

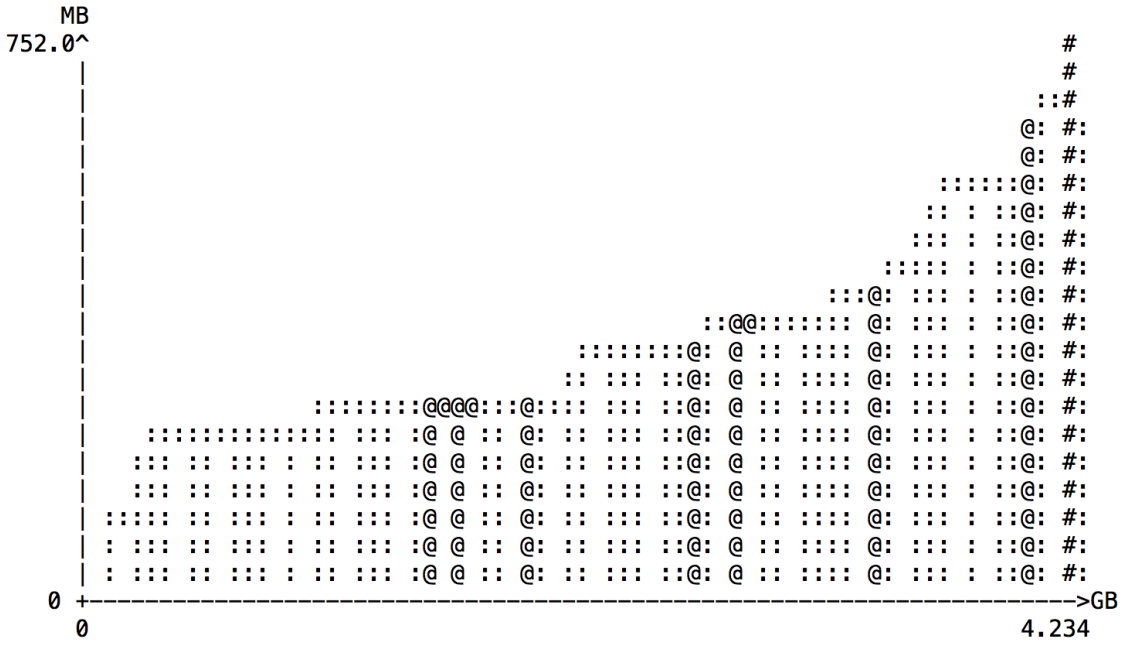


Fig. 20. CSOLNP's memory consumption on the Swedish data with mvn absolute error tolerance of 1e-3

7.4 Conclusion

We profiled CSOLNP, NPSOL and SLSQP with Valgrind's profiling tool Callgrind which counts the number of instructions (at machine level) within each function to find the most expensive functions. We ran Callgrind on the drug abuse dataset and found function evaluations excluding the numerical integration to be the most costly routine for all the three optimizers. As we increased the precision for multivariate normal integration tolerance, computation of multivariate normal integration became the most expensive routine. We also used Valgrind's heap profiling tool to compare memory consumption of the optimizers. All the optimizers used the same amount of memory.

CHAPTER 8

ENHANCING CSOLNP'S PERFORMANCE

We added several features to CSOLNP to improve the performance of the optimizer. These features include: handling analytical gradients; analytical jacobians; the capability of using central difference approximation for computing numerical gradients; and handling infeasible starting points are amongst these features. This chapter provides details about each of these novel features.

8.1 Analytical gradients

We improved the performance of CSOLNP by adding the capability of handling analytical gradients to the optimizer. Previously, CSOLNP would have calculated the gradients at each iteration numerically using forward difference approximation, even in cases where the gradients were made available by the user.

We solved problem 71 in Hock & Schittkowski [75] available by FORTRAN library E04UCF/E04UCA with all the three optimizers in the presence of analytical gradients and without them. This library software is essentially identical to NPSOL. The problem has a nonlinear objective function, bounds on decision variables as well as linear and nonlinear inequality constraints.

The problem may be described as follows:

$$\min f(x) = x_1 x_4 (x_1 + x_2 + x_3) + x_3$$

subject to the bounds :

$$1 \leq x_1 \leq 5$$

$$1 \leq x_2 \leq 5$$

$$1 \leq x_3 \leq 5$$

$$1 \leq x_4 \leq 5$$

to the general linear constraint :

$$x_1 + x_2 + x_3 + x_4 \leq 20$$

and to the nonlinear constraints :

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 \leq 40$$

$$x_1 x_2 x_3 x_4 \geq 25$$

The implementation in OpenMx is as follows:

```
library(OpenMx)
m1 <- mxModel("example",
    mxMatrix(name="pars", nrow=4, ncol=1, free=TRUE, lbound
        =1, ubound=5, values=c(1.01,2,3,4.9)), mxAlgebra(pars
        [1,1]*pars[4,1]*(pars[1,1]+pars[2,1]+pars[3,1])+pars
        [3,1], name="obj"), mxFitFunctionAlgebra("obj"),
        mxConstraint(pars[1,1] + pars[2,1] + pars[3,1] + pars
        [4,1] < 20 ), mxConstraint(pars[1,1]^2 + pars[2,1]^2 +
         pars[3,1]^2 + pars[4,1]^2 < 40), mxConstraint(pars
        [1,1]*pars[2,1]*pars[3,1]*pars[4,1] > 25))
```

```
m1 <- mxRun(m1)
m1$pars$values
m1$obj$values
m1$output$evaluations
m2 <- mxModel("exampleWithAnalyticalGrads",
    mxMatrix(name="pars", nrow=4, ncol=1, free=TRUE, lbound
        =1, ubound=5, values=c(1.01,2,3,4.9), labels=paste("x
        ",1:4,sep="")), mxAlgebra(pars[1,1]*pars[4,1]*(pars
        [1,1]+pars[2,1]+pars[3,1])+pars[3,1], name="obj"),
        mxAlgebra(cbind(2*pars[1,1]*pars[4,1] + pars[4,1]*pars
        [2,1] + pars[4,1]*pars[3,1], pars[1,1]*pars[4,1], pars
        [1,1]*pars[4,1]+1, pars[1,1]^2 + pars[1,1]*pars[2,1] +
         pars[1,1]*pars[3,1]), name="objgrad", dimnames=list(
        NULL, paste("x",1:4,sep=""))),    mxFitFunctionAlgebra(
        algebra="obj", gradient="objgrad"), mxConstraint(pars
        [1,1] + pars[2,1] + pars[3,1] + pars[4,1] < 20 ),
        mxConstraint(pars[1,1]^2 + pars[2,1]^2 + pars[3,1]^2 +
         pars[4,1]^2 < 40), mxConstraint(pars[1,1]*pars[2,1]*
        pars[3,1]*pars[4,1] > 25))
m2 <- mxRun(m2)
m2$pars$values
m2$obj$values
m2$output$evaluations
```

We considered the starting values as $x = (1.01, 2, 3, 4.9)$. In model m1, the gradients are computed numerically using forward difference approximation in case

of CSOLNP and NPSOL, and central difference approximation in case of SLSQP. In model m2, the gradient vector is provided to the optimizers.

The optimal solution is at the point $x = (1.0, 4.74, 3.82, 1.37)$. While CSOLNP and NPSOL reach the solution, SLSQP struggles in finding the minimum. SLSQP find the solution at $x = (1, 1, 1, 1)$ with objective value equal to 4. It appears that SLSQP ignores the constraints totally and finds the solution to the unconstrained problem. Both CSOLNP and NPSOL are capable of finding the solution with all the linear and nonlinear constraints being satisfied at the solution. CSOLNP reaches to the solution in fewer number of function evaluations compared to NPSOL. Both of the optimizers perform faster when analytical gradients are provided. This is expected as the presence of analytical gradients should decrease the number of calls to the objective function and hence reduce the number of function evaluations.

A summary of the three optimizers' performance on this problem, as well as how implementing analytical gradients, has enhanced CSOLNP's performance is provided in Table 7.

Optimizer	x_1	x_2	x_3	x_4	**obj**	**evals(AG)**	**evals(NG)**
CSOLNP	1.00	4.74	3.83	1.37	17.01	1211	1717
NPSOL	1.00	4.74	3.83	1.37	17.01	3176	3290
SLSQP	1	1	1	1	4	NA	NA

Table 7. x_1, x_2, x_3 and x_4 are the decision variables. obj is short for final objective value. evals (AG) stands for evaluations with analytical gradients available to the optimizer and evals (NG) stands for evaluations with numerical gradients.

As Table 7 suggests, CSOLNP needs 1,211 function evaluations to find the optimum in the presence of analytical gradients. When the gradients are not provided to the optimizer, CSOLNP needs 1,717 function evaluations to calculate the gradients

numerically and find the solution. As for NPSOL, the optimizer needs 3,176 function evaluations when analytical gradients are provided, and 3,290 evaluations when they are absent. Both optimizers have used forward difference approximation method for numerical computation of the gradients. Also, CSOLNP reaches the minimum faster than NPSOL as the number of evaluations is fewer with CSOLNP regardless of the presence or absence of the analytical gradients. We have not shown the number of evaluations for SLSQP as the optimizer cannot find the solution from these starting values.

8.2 Analytical Jacobians

Next, we added the capability of handling analytical Jacobians to CSOLNP for the the problems where the optimization involves satisfying some constraints. Jacobian matrix is the matrix of partial derivatives of each constraint with respect to the decision variables. We solved the same problem provided in the previous section with CSOLNP, NPSOL and SLSQP. We considered two scenarios:

- Both analytical gradients and Jacobians are provided
- Only analytical Jacobians are provided

As expected, Table 8 shows that the presence of analytical Jacobians decreased the number of function evaluations even further. However, these improvements require specialized knowledge and explicit programming on behalf of the user, which may not always be available or convenient.

8.3 Central difference approximation

We implemented central difference approximation as an alternative to forward difference approximation which is the default numerical approach for gradient compu-

Optimizer	x_1	x_2	x_3	x_4	obj	evals(AJ)	evals(AJ & AG)
CSOLNP	1.00	4.74	3.83	1.37	17.01	1573	1067
NPSOL	1.00	4.74	3.83	1.37	17.01	1136	268
SLSQP	1	1	1	1	4	NA	NA

Table 8. x_1, x_2, x_3 and x_4 are the decision variables. obj is short for final objective value. evals (AJ) stands for evaluations with analytical Jacobians available to the optimizer and evals (NJ) stands for evaluations with numerical Jacobians.

tation in CSOLNP. Central difference approximation calculates the gradient at each point twice, one at $x+\delta$ and one at $x-\delta$. δ is a small perturbation parameter referred to as gradient step size. Central difference calculates the gradient by the following formula:

$$f'(x) \approx \frac{(f(x+\delta) - f(x-\delta))}{2\delta}$$

The default value for δ in CSOLNP is 1e-7. SLSQP uses Central difference approximation by default. NPSOL uses forward difference approximation by default, but if it cannot find the optimum, it switches to central difference.

We ran the same example with CSOLNP using forward and central difference approximation. The results are provided in Table 10. We have excluded NPSOL and SLSQP from the table. The reason is that NPSOL cannot be commanded by the user to use central difference from the start, and SLSQP is already using central difference and cannot find the solution.

8.4 Infeasible initial point

There are situations where the initial point is infeasible meaning that the inequality constraints are not satisfied at the starting point. This situation cannot be handled by RSOLNP. To start the optimization procedure from a feasible starting

Optimizer	x_1	x_2	x_3	x_4	obj	NG/NJ	AG	AJ	AJ/AG
CSOLNP	1.00	4.74	3.83	1.37	17.01	2725	1355	2437	1067

Table 9. x_1, x_2, x_3 and x_4 are the decision variables. obj is short for final objective value. NG/NJ stands for numerical gradients and Jacobians. AG stands for analytical gradients option on. AJ stands for evaluations with analytical Jacobians available to the optimizer and AJ/AG stands for evaluations with analytical gradients and Jacobians.

Optimizer	x_1	x_2	x_3	x_4	obj	NG/NJ	AG	AJ	AJ/AG
CSOLNP	1.00	4.74	3.83	1.37	17.01	1717	1211	1573	1067
NPSOL	1.00	4.74	3.83	1.37	17.01	3290	3176	1136	268
SLSQP	1.00	1.00	1.00	1.00	4	NA	NA	NA	NA

Table 10. x_1, x_2, x_3 and x_4 are the decision variables. obj is short for final objective value. NG/NJ stands for numerical gradients and Jacobians. AG stands for analytical gradients option on. AJ stands for evaluations with analytical Jacobians available to the optimizer and AJ/AG stands for evaluations with analytical gradients and Jacobians.

point, we overcame this difficulty by replacing the objective function with the sum of violated inequalities, and optimizing the parameters with respect to this new objective function. The solution to this problem will then be used as the starting point for the original problem (original objective function).

We solved the following problem with CSOLNP:

$$\min f(x) = x_5 \tag{8.1}$$

$$\text{subject to :}$$

$$x_1, x_2, x_3, x_4 \geq -1$$

$$x_1 + x_2 + x_3 + x_4 \leq x_5$$

$$x_1 = x_2^2$$

The starting point is $x = (0.1, 0.1, 0.1, 0.1, -1)$. This point is infeasible as x_5 is -1 and hence the constraint $x_1 + x_2 + x_3 + x_4 < x_5$ is violated. To find a feasible direction, CSOLNP changes the original problem to the following:

$$\min f(x) = x_1 + x_2 + x_3 + x_4 - x_5 \tag{8.2}$$

$$\text{subject to :}$$

$$x_1, x_2, x_3, x_4 \geq -1$$

The modified problem 8.2 converged in 2 iterations with an objective value of -0.49 and the solution at point $x = (-0.0582, -0.2414, -0.2197, -0.2197, -0.4961)$. This new feasible point is then treated as the starting point for the original problem 8.1. The original problem converged in 12 iterations with final objective value of -4 and the optimum at $x = (-1, -1, -1, -1)$.

We also solved the same problem we solved in the previous sections with an infeasible point $x = (1, 5, 5, 1)$. The point does not satisfy the constraint $x_1^2 + x_2^2 + x_3^2 + x_4^2 \leq 40$, and hence is infeasible. To find a feasible direction, CSOLNP replaces the objective function $f(x) = x_1x_4(x_1+x_2+x_3)+x_3$ with the violated constraint, and solves the problem with no constraints. This problem converges in two iterations and finds the optimum at $x = (1.50769, 3.98851, 3.98851, 1.50769)$ with the new objective

value of 25.548. We can now solve the original problem with this new point and in the presence of all the constraints. The solution is found in 2,177 evaluations.

8.5 Conclusion

We enhanced CSOLNP's performance by adding the capability of handling analytical gradients and Jacobians, as well as implementing central difference approximation to be used alongside forward difference which is the default numerical gradient computation approach in CSOLNP. We also improved CSOLNP's performance in finding a feasible search direction when the starting point violates one of the inequality constraints.

Given all the three optimizers have the capabilities of handling analytical gradients and Jacobians, we tested these features on a problem from Hock & Schittkowski [75]. We ran all three optimizers on this problem and compared the final objective value and the optimum point. We considered different scenarios for our tests:

- Gradients and Jacobians need to be calculated numerically due to the absence of analytical gradients and Jacobians.
- Only analytical gradients are available to the optimizers.
- Both analytical gradients and Jacobians are available.
- Analytical Jacobians are available but analytical gradients are not available.

Both CSOLNP and NPSOL reached the solution, but SLSQP failed to find a feasible solution. CSOLNP reached the solution faster in the case of numerical computation of gradients and Jacobians. With Analytical gradients and Jacobians available to the optimizers, NPSOL's number of function evaluation dropped dramatically.

For central difference approximation, we only ran CSOLNP, as SLSQP could not find the solution to this problem, and NPSOL uses forward difference by default and cannot be prompted to use central difference approximation from the start. We compared the number of function evaluations with and without analytical gradients and Jacobians. The number of function evaluations decreased as the analytical gradients and Jacobians were fed into the model.

Finally, we tested CSOLNP on the same problem with an infeasible starting point, as well as a new problem with a simple linear objective function and constraints. CSOLNP could find a feasible search direction in both cases.

CHAPTER 9

CONCLUSION

There are inherent difficulties in substance abuse assessment. Behavioral data are mostly of binary or ordinal type, which makes them less accurate than continuous type. A common approach to model ordinal data is to consider a latent, normally distributed continuous variable underlying each ordinal variable. This way, an ordinal variable with 3 categories, for instance, is assumed to follow a normal distribution that is cut to three partitions by two thresholds. The latent continuous variables are only observed when being above or below any of the thresholds. With such an approach to ordinal data, the etiology of substance use behavior can be investigated using Structural Equation Modeling (SEM).

In this dissertation, we have developed an optimization engine for the package OpenMx which is a popular SEM software for estimation of a wide variety of statistical models. Our newly developed optimizer solves general nonlinear optimization problems using Sequential Quadratic Programming algorithm. We have compared the performance of our optimizer with two other implementations of the SQP method available in OpenMx package. While the optimizers usually reach the same minimum, our optimizer is faster and more consistent than the other two when tested on threshold models for ordinal data.

We then applied our newly developed optimizer within OpenMx on a drug abuse dataset collected in Sweden from more than 1 million people to find the contribution of genetic and environmental components in liability to drug abuse as ascertained through medical and criminal records. Modeling twin and sibling data, we found a

substantial contribution of genetic factors and a moderate contribution of environmental factors to likelihood of drug abuse. Males showed higher heritability than did females. Vulnerability to environmental factors was also higher in males than in females. Moreover, both of these factors were higher for drug abuse ascertained through criminal records than medical records.

We compared optimizers' performance on the drug abuse dataset. Similar to the results we got from running threshold models on simulated data, we found CSOLNP to be faster than NPSOL and SLSQP on the drug abuse dataset as well. We then used the Valgrind tool suite to profile the optimizers. We used Callgrind to find the most expensive functions in the three optimizers. The function evaluation routine (excluding its calls to numerical integration routine) was the most time-consuming part. As the multivariate normal integration precision increases, the most expensive function is the multivariate normal integration routine sadmvn [72] which is called many times within each function evaluation. We also compared memory consumption of the optimizers. We used Massif, a heap profiler tool available in Valgrind for this purpose. All three optimizers consumed very similar amount of memory.

Finally we improved CSOLNP's performance in several aspects. We added the capability of handling analytical gradients and Jacobians to the optimizer. We added central difference numerical gradient approximation to CSOLNP. Finally, we improved CSOLNP's performance in dealing with infeasible starting points. We achieved this by replacing the objective function with the sum of inequalities and finding the optimum with respect to this new objective function. After finding a feasible initial point, we solve the original problem.

We tested all these features on a problem from FORTRAN library E04UCF/E04UCA and compared the results across the three optimizers. CSOLNP and NPSOL were capable of finding the solution while SLSQP failed at satisfying the constraints. Fi-

nally, I hope my contributions to the OpenMx R package will prove useful for many years to come.

Appendix A

AKAIKE INFORMATION CRITERION (AIC) AND BAYESIAN INFORMATION CRITERION (BIC)

Akaike information criterion (AIC) [76] is a fit index to compare the quality of a set of statistical models for a given dataset. AIC is based on information theory, and is designed to choose a model that minimizes the Kullback-Leibler distance [77] between the model and the data. It is defined as:

$$AIC = -2\ln\mathcal{L} + 2(p+1) \ ,$$

where $\mathcal{L}$ is the likelihood of the model given the data and p is the number of free parameters in the model. The second term is a penalty term to avoid overfitting.

Bayesian information criterion (BIC) [78] is another fit index for model selection. Similar to AIC, it provides a trade-off between model accuracy and model complexity. BIC is defined as:

$$BIC = -2\ln\mathcal{L} + 2(p+1)\ln(N) \ ,$$

where N is the sample size, and the other terms are the same as described in the definition of AIC.

REFERENCES

[1] United Nations Office on Drugs and Crime. *World drug report 2010*. United Nations Publications, 2010.

[2] Wilson M Compton et al. "Prevalence, correlates, disability, and comorbidity of DSM-IV drug abuse and dependence in the United States: results from the national epidemiologic survey on alcohol and related conditions". In: *Archives of general psychiatry* 64.5 (2007), pp. 566–576.

[3] US Department of Health, Human Services, et al. "The health consequences of smokingfffdfffdfffd50 years of progress: a report of the Surgeon General". In: *Atlanta, GA: US Department of Health and Human Services, Centers for Disease Control and Prevention, National Center for Chronic Disease Prevention and Health Promotion, Office on Smoking and Health* 17 (2014).

[4] Kenneth S Kendler et al. "The structure of genetic and environmental risk factors for common psychiatric and substance use disorders in men and women". In: *Archives of general psychiatry* 60.9 (2003), pp. 929–937.

[5] Jacquelyn L Meyers and Danielle M Dick. "Genetic and environmental risk factors for adolescent-onset substance use disorders". In: *Child and adolescent psychiatric clinics of North America* 19.3 (2010), pp. 465–477.

[6] Eivind Ystrom et al. "Genetic and environmental risk factors for illicit substance use and use disorders: joint analysis of self and co-twin ratings". In: *Behavior genetics* 44.1 (2014), pp. 1–13.

[7] Susan E Young et al. "Genetic and environmental vulnerabilities underlying adolescent substance use and problem use: general or specific?" In: *Behavior genetics* 36.4 (2006), pp. 603–615.

[8] Kenneth S Kendler, John Myers, and Carol A Prescott. "Specificity of genetic and environmental risk factors for symptoms of cannabis, cocaine, alcohol, caffeine, and nicotine dependence". In: *Archives of General Psychiatry* 64.11 (2007), pp. 1313–1320.

[9] Kenneth S Kendler et al. "Specificity of genetic and environmental risk factors for use and abuse/dependence of cannabis, cocaine, hallucinogens, sedatives, stimulants, and opiates in male twins". In: *American Journal of Psychiatry* 160.4 (2003), pp. 687–695.

[10] Regina A Shih, Pamela L Belmonte, and Peter P Zandi. "A review of the evidence from family, twin and adoption studies for a genetic contribution to adult psychiatric disorders". In: *International review of psychiatry* 16.4 (2004), pp. 260–283.

[11] K Silventoinen et al. "The genetic and environmental influences on childhood obesity: a systematic review of twin and adoption studies". In: *International journal of Obesity* 34.1 (2010), pp. 29–40.

[12] Christian J Hopfer, Thomas J Crowley, and John K Hewitt. "Review of twin and adoption studies of adolescent substance use". In: *Journal of the American Academy of Child & Adolescent Psychiatry* 42.6 (2003), pp. 710–719.

[13] Robert Plomin. "The role of inheritance in behavior". In: (1990).

[14] Auke Tellegen et al. "Personality similarity in twins reared apart and together." In: *Journal of personality and social psychology* 54.6 (1988), p. 1031.

[15] Kathleen McCartney, Monica J Harris, and Frank Bernieri. "Growing up and growing apart: a developmental meta-analysis of twin studies." In: *Psychological bulletin* 107.2 (1990), p. 226.

[16] Kenneth S Kendler et al. "A test of the equal-environment assumption in twin studies of psychiatric illness". In: *Behavior genetics* 23.1 (1993), pp. 21–27.

[17] Nancy L Segal. *Entwined lives: Twins and what they tell us about human behavior.* Dutton/Penguin Books, 1999.

[18] Kenneth S Kendler et al. "Illicit psychoactive substance use, heavy use, abuse, and dependence in a US population-based sample of male twins". In: *Archives of general psychiatry* 57.3 (2000), pp. 261–269.

[19] Kathleen R Merikangas et al. "Familial transmission of substance use disorders". In: *Archives of general psychiatry* 55.11 (1998), pp. 973–979.

[20] Roy W Pickens et al. "Heterogeneity in the inheritance of alcoholism: a study of male and female twins". In: *Archives of General Psychiatry* 48.1 (1991), pp. 19–28.

[21] Arpana Agrawal and Michael T Lynskey. "Are there genetic influences on addiction: evidence from family, adoption and twin studies". In: *Addiction* 103.7 (2008), pp. 1069–1081.

[22] Kenneth S Kendler et al. "Genetic and familial environmental influences on the risk for drug abuse: a national Swedish adoption study". In: *Archives of general psychiatry* 69.7 (2012), pp. 690–697.

[23] Ming T Tsuang et al. "Genetic influences on DSM-III-R drug abuse and dependence: A study of 3,372 twin pairs". In: *American journal of medical genetics* 67.5 (1996), pp. 473–477.

[24] Michael C Neale et al. "Methodological issues in the assessment of substance use phenotypes". In: *Addictive Behaviors* 31.6 (2006), pp. 1010–1034.

[25] Erich Leo Lehmann. *Elements of large-sample theory*. Springer Science & Business Media, 1999.

[26] Steven Boker et al. "OpenMx: an open source extended structural equation modeling framework". In: *Psychometrika* 76.2 (2011), pp. 306–317.

[27] Michael C Neale et al. "OpenMx 2.0: Extended structural equation and statistical modeling". In: *Psychometrika* (2015), pp. 1–15.

[28] Valgrind Developers. "Valgrind". In: *Web page at http://valgrind. org (2000–2005)* (2010).

[29] Jodie B Ullman and Peter M Bentler. *Structural equation modeling*. Wiley Online Library, 2003.

[30] Laura Klem. "Structural equation modeling." In: (2000).

[31] Natasha K Bowen and Shenyang Guo. *Structural equation modeling*. Oxford University Press, 2011.

[32] Victoria Savalei and Peter M Bentler. "Structural equation modeling". In: *Corsini encyclopedia of psychology* (2010).

[33] Singgih Santoso. *Structural Equation Modeling*. Elex Media Komputindo, 2011.

[34] Charles Spearman. "" General Intelligence," objectively determined and measured". In: *The American Journal of Psychology* 15.2 (1904), pp. 201–292.

[35] MCCL Neale and Lon R Cardon. *Methodology for genetic studies of twins and families*. Vol. 67. Springer Science & Business Media, 2013.

[36] Steven M Boker et al. "OpenMx User Guide". In: *Release* 1 (2012), pp. 0–1919.

[37] Albert Maydeu-Olivares and John J McArdle. *Contemporary psychometrics*. Psychology Press, 2005.

[38] Steven M Boker, JJ McArdle, and Michael Neale. "An algorithm for the hierarchical organization of path diagrams and calculation of components of expected covariance". In: *Structural Equation Modeling* 9.2 (2002), pp. 174–194.

[39] Scott I Vrieze. "Model selection and psychological theory: a discussion of the differences between the Akaike information criterion (AIC) and the Bayesian information criterion (BIC)." In: *Psychological methods* 17.2 (2012), p. 228.

[40] Hirotugu Akaike. "Factor analysis and AIC". In: *Selected Papers of Hirotugu Akaike*. Springer, 1987, pp. 371–386.

[41] Daire Hooper, Joseph Coughlan, and Michael Mullen. "Structural equation modelling: Guidelines for determining model fit". In: *Articles* (2008), p. 2.

[42] Michael W Browne, Robert Cudeck, et al. "Alternative ways of assessing model fit". In: *Sage focus editions* 154 (1993), pp. 136–136.

[43] JM Linacre and BD Wright. "Chi-square fit statistics". In: *Rasch Measurement Transactions* 8.2 (1994), p. 350.

[44] Douglas S Falconer. "The inheritance of liability to certain diseases, estimated from the incidence among relatives". In: *Annals of human genetics* 29.1 (1965), pp. 51–76.

[45] James H Steiger. "Tests for comparing elements of a correlation matrix." In: *Psychological bulletin* 87.2 (1980), p. 245.

[46] David G Luenberger. *Introduction to linear and nonlinear programming*. Vol. 28. Addison-Wesley Reading, MA, 1973.

[47] Stephen Wright and Jorge Nocedal. "Numerical optimization". In: *Springer Science* 35 (1999), pp. 67–68.

[48] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*. Vol. 97. Springer Science & Business Media, 2005.

[49] Mordecai Avriel. *Nonlinear programming: analysis and methods*. Courier Corporation, 2003.

[50] Trond Steihaug. "The conjugate gradient method and trust regions in large scale optimization". In: *SIAM Journal on Numerical Analysis* 20.3 (1983), pp. 626–637.

[51] Jonathan Richard Shewchuk et al. *An introduction to the conjugate gradient method without the agonizing pain*. 1994.

[52] Joseph-Frédéric Bonnans et al. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[53] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[54] Charles George Broyden. "The convergence of a class of double-rank minimization algorithms 1. General considerations". In: *IMA Journal of Applied Mathematics* 6.1 (1970), pp. 76–90.

[55] Roger Fletcher. "A new approach to variable metric algorithms". In: *The computer journal* 13.3 (1970), pp. 317–322.

[56] David G Luenberger and Yinyu Ye. "Linear and nonlinear programming. International series in operations research & management science". In: *Springer, Berlin. doi* 10 (2008), pp. 978–.

[57] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[58] Harold W Kuhn and Albert W Tucker. "Nonlinear programming". In: *Traces and emergence of nonlinear programming*. Springer, 2014, pp. 247–258.

[59] Alberto Cambini and Laura Martein. *Second order optimality conditions*. Università di Pisa, Dipartimento di statistica e matematica applicata all'economia, 1997.

[60] Roberto Cominetti. "Metric regularity, tangent sets, and second-order optimality conditions". In: *Applied Mathematics and Optimization* 21.1 (1990), pp. 265–287.

[61] Paul T Boggs and Jon W Tolle. "Sequential quadratic programming". In: *Acta numerica* 4 (1995), pp. 1–51.

[62] Frédéric Delbos and Jean Charles Gilbert. "Global linear convergence of an augmented Lagrangian algorithm for solving convex quadratic optimization problems". PhD thesis. INRIA, 2003.

[63] Katta G Murty and Feng-Tien Yu. *Linear complementarity, linear and nonlinear programming*. Vol. 3. Citeseer, 1988.

[64] Sanjay Mehrotra. "On the implementation of a primal-dual interior point method". In: *SIAM Journal on optimization* 2.4 (1992), pp. 575–601.

[65] Irvin J Lustig, Roy E Marsten, and David F Shanno. "Interior point methods for linear programming: Computational state of the art". In: *ORSA Journal on Computing* 6.1 (1994), pp. 1–14.

[66] Philip E Gill et al. *User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming*. Tech. rep. DTIC Document, 1986.

[67] Steven G Johnson. *The NLopt nonlinear-optimization package*. 2014.

[68] Mahsa Zahery, Hermine H Maes, and Michael C Neale. "CSOLNP: Numerical Optimization Engine for Solving Non-linearly Constrained Problems". In: *Twin Research and Human Genetics* 20.4 (2017), pp. 290–297.

[69] Alexios Ghalanos and Stefan Theussl. "Rsolnp: general non-linear optimization using augmented Lagrange multiplier method". In: *R package version* 1 (2012).

[70] Lorenz T Biegler et al. "Large-scale PDE-constrained optimization: an introduction". In: *Large-Scale PDE-Constrained Optimization*. Springer, 2003, pp. 3–13.

[71] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming". In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM. 1984, pp. 302–311.

[72] Alan Genz. "Numerical computation of multivariate normal probabilities". In: *Journal of computational and graphical statistics* 1.2 (1992), pp. 141–149.

[73] Hermine H Maes et al. "A Bivariate Genetic Analysis of Drug Abuse Ascertained Through Medical and Criminal Registries in Swedish Twins, Siblings and Half-Siblings". In: *Behavior genetics* 46.6 (2016), pp. 735–741.

[74] Michael Neale and Lon Cardon. *Methodology for genetic studies of twins and families*. Vol. 67. Springer Science & Business Media, 1992.

[75] Willi Hock and Klaus Schittkowski. "Test examples for nonlinear programming codes". In: *Journal of Optimization Theory and Applications* 30.1 (1980), pp. 127–129.

[76] Hirotogu Akaike. “Information theory and an extension of the maximum likelihood principle”. In: *Selected Papers of Hirotugu Akaike*. Springer, 1998, pp. 199–213.

[77] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[78] Gideon Schwarz et al. “Estimating the dimension of a model”. In: *The annals of statistics* 6.2 (1978), pp. 461–464.

VITA

Mahsa Zahery was born in Tehran, Iran. She graduated from high school in May 2002, and received her Bachelor of Science in Computer Engineering with major in Software at the Azad University of Central Tehran in February 2007. In September 2008, Mahsa began her graduate studies in the field of Bioinformatics & Systems Biology at Chalmers University of Technology in Sweden. She did her master thesis in TU Delft in the Netherlands. Mahsa received her Master of Science degree in July 2010. She then started her Ph.D. in the department of Computer Science at Virginia Commonwealth University in September, the same year. She got her second master's degree, in the field of Computer Science in May 2013. Mahsa is a music lover and enjoys singing in her spare time.