MONASH University

Semidefinite approximation for Gaussian maximum likelihood function

Chi Bach Pham

Supervisor: Dr James Saunderson

Final Report submitted for ECE4095 Project B at

Monash University in 2021

Department of Electrical and Computer System Engineering

Significant contributions

Formulate multiple ways to approximate scalar function $f(x) = \ln(x) + x^{-1}$ that preserve its convexity.

Using the semi-definite approximation of $f(x)$ to minimize the Gaussian likelihood function

Numerical experimentation on estimation parameters to determine the trade-off for each of these parameters

Encase all of the above into an user-friendly `python` package and release for public use.

MONASH University
Engineering

Department of Electrical and
Computer Systems Engineering

ECE4095 Final Year Project 2021 — **Chi Bach Pham**

Maximum likelihood estimation of Gaussian distribution using conventional solver

Supervisor: Dr James Saunderson

Aim

Develop a **semi-definite representation** of the multivariate Gaussian distribution maximum likelihood function **to use in conventional convex optimization solver**.

Background

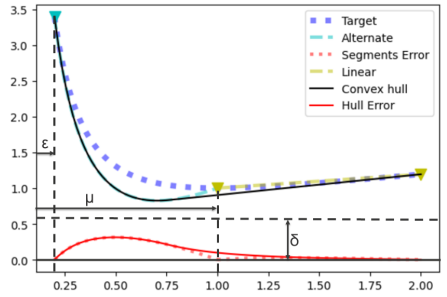
- Numerous previous approach in solving the maximum likelihood function used **very specific algorithm or method** that is **hard to replicate or inaccessible** for people without background in the field
- Conventional convex optimization solvers, which are more accessible is **lacking the ability to solve the function** due to the lack of a semi-definite representation.

Approach

- Using the **separatable** properties of the likelihood function, the way we solve the multivariate version is not much different than that of solving the one-dimensional version.
- So the semi-definite representation can be modify and scale up from the 1-d representation of scalar function **$f(x) = \ln(x) + x^{-1}$**

Solution

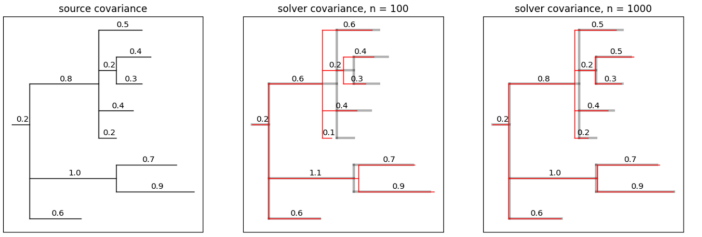
- Two type of semi-definite representable function that was use for the estimation are **$f(x) = x^{-1} + a\,x + b$ (alternate)** and **$f(x) = a\,x + b$ (linear)**, depending on the curvature of the target function.
- Then the whole representation need to be stitched together by a **convex hull to preserve convexity**.
- **Four parameter can be adjust to better fit user's preference**, maximum error(δ), minimum x coordinate(ε), segment type boundary(μ) and error figure function



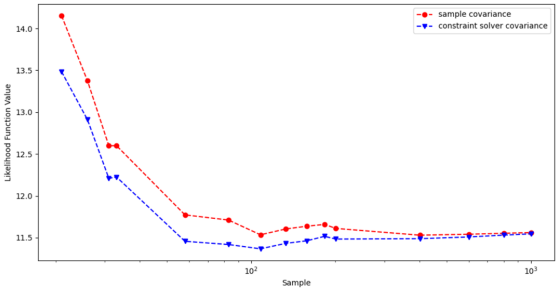
A representation with δ set to 0.6, μ at 1 and the ε is at 0.2. Here, the error figure is calculated as the maximum difference between the approximation and the target. These was set very loosely for demonstration purpose

Experiment

- One of a by-product of using the solvers is that the **user can include extra linear constraints as needed**, which would be more difficult had they had use other method of estimating the covariance.



Above, a dataset is generated from a Brownian motion tree model. Then the solver is ran with the tree constraint, which provided the edge length simultaneously with the covariance.



Then we can see the comparison between the result of the solver with the constraint versus the sample covariance on the right. The solver would yield better result in most cases. Note that as we work with negative log likelihood, we want the likelihood value to be minimal.

Executive Summary

This report outlines the process of solving the Gaussian maximum likelihood estimation problem using an off-the-shelf convex optimization solver. To do so, we must devise a semi-definite programming approximation of the likelihood function. Several ways of approximating the likelihood function were devised during the project and the result was a combination of two of these approaches. Then numerical experimentation was done to balance the estimation parameters. The outcome of this project is a parser that encapsulated all these complex approximation processes into a user-friendly python package for use concurrently with `CVXPY`, a python-based solver.

Abstract

This project aims to develop a semi-definite approximation for the likelihood function of multivariate Gaussian distribution. Previous studies on this topic use specialized methods that are hard to replicate for people without some background in the field. We aim to create a good approximation of the likelihood function so that the Gaussian maximum likelihood estimation problem can be solved by a conventional convex optimization solver. By exploiting properties of the likelihood function we create a parser that can approximate it using many segments of functions that are semi-definite representable, which enables a standard solver like `CVXPY` to solve. The by-product of creating such representation is that extra convex constraints can be added to the covariance of the Gaussian model with ease.

Contents

List of Figures

Abbreviations

PDF	**P**robability **D**ensity **F**unction
MLE	**M**aximum **L**ikelihood **E**stimtion
MLF	**M**aximum **L**ikelihood **F**unction
SDP	**S**emi-**D**efinite **P**rogramming
PSD	**P**ositive **S**emi-**D**efinite

Chapter 1

Introduction

1.1 Background

In statistics, the Gaussian distribution is arguably the most important probability distribution, because it fits many natural phenomena and is widely used to represent random variables with unknown distribution [2]. It is fully characterized by its mean μ and standard deviation σ and is represented by the PDF

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}. \tag{1.1}$$

When generalized into higher dimensions, the multivariate Gaussian distribution has the form

$$p(X) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)} \tag{1.2}$$

where Σ is a positive-definite covariance matrix, X is a vector of random variables and μ is the mean of X In this project, we are more interested in the multivariate version and fitting Σ from the samples of X. As such μ is assumed to be 0, and so the PDF simplifies to

$$p(X) = (2\pi)^{-\frac{d}{2}} \det(\Sigma)^{-\frac{1}{2}} e^{-\frac{1}{2}X^T \Sigma^{-1} X}. \tag{1.3}$$

The PDF gives us the *relative probability* that a value is observed given the covariance. It is relative due to there being infinite possible value for X but it can still infer how much more likely that we observed the data point at this point compare to some other point.

A central problem in statistics is, given a data set $\{X_1, X_2, \ldots, X_n\}$, what would be the most probable value for Σ. As stated earlier, the value of the PDF would be higher if the value of X is more probable. So by making Σ into the variable, we can search for the Σ that maximize p, given X. This problem is called maximum likelihood estimation(MLE). Given samples $\{X_1, X_2, \ldots, X_n\}$ of X, the likelihood function is

$$L(\Sigma) = \prod_{i=1}^{n} p(X_i) = \frac{1}{(2\pi)^{\frac{nd}{2}} \det(\Sigma)^{\frac{nd}{2}}} \prod_{i=1}^{n} e^{-\frac{1}{2} X_i^T \Sigma^{-1} X_i}. \tag{1.4}$$

As we are dealing with exponential and product, as well as it is more desirable to work with a minimization problem, we will use the negative log-likelihood function in this project

$$L(\Sigma) = \sum_{i=1}^{n} -\ln(p(X_i)) = \frac{nd}{2} ln(2\pi) + \frac{n}{2} \ln(\det(\Sigma)) + \frac{1}{2} \sum_{i=1}^{n} X_i^T \Sigma^{-1} X_i. \tag{1.5}$$

Very often, we would like to add convex constraints to the covariance, such as ensuring the diagonal entries are 1. This gives rise to the covariance-constrained maximum likelihood estimation problem, which is the focus of this report. The problem of maximum likelihood estimation is, therefore minimize $L(\Sigma)$ subjected to Σ positive definite. The representation that we need to create in order to be suitable for a SDP solver should have the form

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} c_i x_i \\ \text{subject to} \quad & X = \sum_{i=1}^{m} F_i x_i - F_0 \\ & X \succeq 0. \end{aligned} \tag{1.6}$$

Which basically said that we want to minimize a linear function subject to the constraints that a combination of matrices is positive semi-definite.

1.2 Literature Review

1.2.1 Finding the covariance

Solving the MLE is not the only way we can fit the covariance. The more simple approach on finding the data generating covariance matrix is the simple sample covariance where

$$S_n = \frac{1}{n}\sum_{i=1}^{n} X_i X_i^T. \tag{1.7}$$

Although simple and straightforward, it was noted early on that it can be a biased and inefficient estimator if there is not a substantial amount of data[3]. And the process of adding the constraints might be different for each type of constraint. Over the year, many works has been done to produce a better estimator and many of those focus on solving the MLE which we take after in this project.

1.2.2 Solving the MLE

It is noted in many previous pieces of research that solving the MLE on the inverse covariance would result in a concave function, which leads to a convex optimization problem [4], while solving it on the covariance matrix would not result in a convex problem[2], and would result in a multimodal problem. However, under some condition, namely having a large sample-over-dimension ratio would still result in a convex optimization problem[4].

In many of these research, conventional off-the-shelf convex optimization solvers are not used. But iterative processes like the Newton-Raphson method or gradient descent method are used instead. Or using non-linear algebra in case the multimodal problem is of concern.

Using off-the-shelf solvers should be more accessible for many people. For that reason, `CVXPY` was chosen for this project as it is `python` based, which is a popular and easy to pick up language. While also provide us the ability to add convex constraints more easily.

1.2.3 Adding linear constraints

There are numerous researches where adding linear constraints on the covariance would be desirable[5, 6]. It is used to study repeated time series[7], or to study phylogenetic[8]. Some examples for these constraints are sparse matrices where only a few entries are non-zero, correlation matrices where the diagonal is 1, or the more prominent examples being Brownian motion tree models.

If we try to solve it with the inverse covariance, adding the constraint would add complications as we are adding constraints on the inverse of our objective. Adding the constraints on the covariance would make more sense, but as discussed earlier is usually multimodal and not a convex optimization problem.

However, if we solve it in a specific region where the covariance is between 0 and $2S_n$ where S_n is the sample covariance, it would be a strictly convex optimization problem, with a high probability to contain the true data-generating covariance matrix[4]. This would be the direction this project is headed at as it is more suitable for an SDP solver.

1.2.4 Semi-definite programming

The restriction we put on the covariance ($0 \preceq \Sigma \preceq 2S_n$) means we need solvers that can handle such positive semi-definite (PSD) constraints, which mean we need to use at least an SDP solver. And there are not many available works if we want to go higher on the hierarchy.

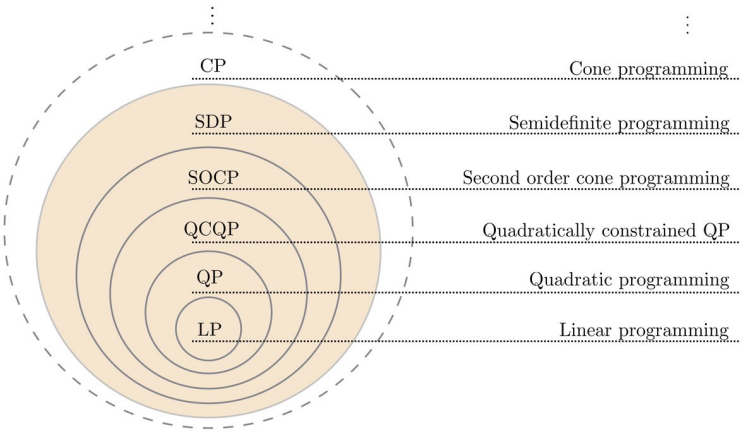


FIGURE 1.1: Hierarchy of convex optimization problems

The idea of creating a process to approximate a convex or concave function into semi-definite programming representation has exist before[9]. As it has the promise of being more accessible as long as we can provide a reasonable approximation error figure.

The paper mention above uses integral representation, coupled with Gaussian quadrature to approximate the matrix logarithm. However, using integral representation did not seem to be a straightforward option for this project, but could be another direction that we could take in the future. This will be discussed in Chapter 6.2.

Chapter 2

Mathematical Background

In this section, we discuss the pre-existing mathematical background that form the foundation for this project to be build upon. In 2.1 we discuss on simplifying the objective function. And then in 2.2 we discuss on how the SDP representation is related to the SDP representation of $f(x) = \ln(x) + x^{-1}$.

2.1 Simplifying the objective function

Equation 1.5 is the equation that we want to minimize but will not be the objective function that we give the solver. As there are properties of it that can be exploited. First we can use $\sum x_i^T \Sigma^{-1} x_i = \text{tr}(\sum x_i^T \Sigma^{-1} x_i)$ as it is scalar and $\text{tr}(x) = x$ if x is scalar. Then we can use the multiplicative property of the determinant $\det(AB) = \det(A)\det(B)$ and the cyclic property of the trace

$\text{tr}(ABC) = \text{tr}(CAB)$ to expand the Equation 1.5

$$\begin{aligned} L(\Sigma) &= \frac{nd}{2}\ln(2\pi) + \frac{n}{2}\ln(\det(\Sigma)) + \frac{1}{2}\sum_{i=1}^{n} x_i^T \Sigma^{-1} x_i \\ &= \frac{nd}{2}\ln(2\pi) + \frac{n}{2}\ln(\det(\Sigma S_n^{-1})\det(S_n)) + \frac{1}{2}\text{tr}(\Sigma^{-1}\sum_{i=1}^{n} x_i^T x_i) \\ &= \frac{nd}{2}\ln(2\pi) + \frac{n}{2}\ln(\det(\Sigma S_n^{-1})) + \\ &\quad \frac{n}{2}\ln(\det(S_n)) + \frac{n}{2}\text{tr}(\Sigma^{-1} S_n). \end{aligned} \tag{2.1}$$

In Equation 2.1, outside from the two constant terms, the variable in the oher two are inverse of each other and hence we can use substitution, $X = \Sigma S_n^{-1}$. And the function that we would try to minimize at this point is

$$\begin{aligned} &\text{Min} && \frac{nd}{2}\ln(2\pi) + \frac{n}{2}\ln(\det(\Sigma S_n^{-1})) + \frac{n}{2}\ln(\det(S_n)) + \frac{n}{2}\text{tr}(\Sigma^{-1} S_n) \\ \Leftrightarrow &\text{Min} && \ln(\det(\Sigma S_n^{-1})) + tr(\Sigma^{-1} S_n) \\ \Leftrightarrow &\text{Min} && \ln(\det(X)) + \text{tr}(X^{-1}). \end{aligned} \tag{2.2}$$

2.2 The connection to $f(x) = \ln(x) + x^{-1}$

The value of the Equation 2.2 is actually only dependant on the eigenvalue of X. Since X is a symmetric matrix and we can write it as $X = U\Lambda U^T$ where Λ is diagonal matirx with $\Lambda_{i,i} = \lambda_i$, and U is a orthagonal and so satisfy $U^T U = UU^T = I$. Using the cyclic property of trace and the multiplicative of the determinant we see that

$$\Longrightarrow \begin{cases} \text{tr}(X^{-1}) = \text{tr}(U^{-T}\Lambda^{-1}U^{-1}) = \text{tr}(\Lambda^{-1}U^{-T}U^{-1}) = \text{tr}(\Lambda^{-1}) = \sum_i \lambda_i^{-1} \\ \det(X) = \det(U\Lambda U^T) = \det(U)\det(\Lambda)\det(U^T) = \det(UU^T)\det(\Lambda) = \prod_i \lambda_i \end{cases}$$

$$\Longrightarrow \ln(\det(X) + tr(X^{-1}) = \sum_i (\ln(\lambda_i) + \lambda_i^{-1}).$$

We can see that the value of $F(X) = ln(det(X) + tr(X^{-1})$ is only dependant on its eigenvalues. Let $f(x) = ln(x) + x^{-1}$. Define

$$f(X) = f(U\Lambda U^T) = U \begin{bmatrix} f(\lambda_1) & 0 & \dots & 0 \\ 0 & f(\lambda_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & f(\lambda_d) \end{bmatrix} U^T$$

and $F(X) := tr(f(X))$

Now, we can reorganize the optimisation problem

$$\begin{aligned} &\text{Min } F(X) && \text{s.t } 0 \preceq X \preceq 2I \\ \Leftrightarrow &\text{Min } tr(f(X)) && \text{s.t } 0 \preceq X \preceq 2I \\ \Leftrightarrow &\text{Min } y && \text{s.t} \begin{cases} tr(f(X)) \leq y \\ 0 \preceq X \preceq 2I. \end{cases} \end{aligned} \tag{2.3}$$

$f(x)$ still needs a semi-definite representation before it is ready for the SDP solver. Fortunately, we can lift the SDR of a linear function using Theorem 3.1 from[10]. Note that the Kronecker product is replaced with multiplication in our case because F terms are scalar,

$$\begin{aligned} f(x) \leq y \Leftrightarrow \exists \mathbf{y} \in \mathbb{R}^n : F_0 + xF_X + yF_Y + \sum_{i=1}^{n} t_i F_i \succeq 0 \\ \Leftrightarrow tr(f(X)) \leq y \Leftrightarrow \exists Y, T_1 \dots, T_n \in \mathbb{R}^{n \times n} : \\ F_0 I + XF_X + YF_Y + \sum_{i=1}^{n} F_i T_i \succeq 0 \\ tr(Y) \leq y \end{aligned} \tag{2.4}$$

where I is the identity matrix of side $d \times d$, and $F_0, \dots, F_n, F_X, F_T$ are what we need to determine by approximating $f(x) = ln(x) + x^{-1}$, this will be discuss in Chapter 3. To summarize, the final SDP problem for the MLE of Gassian

distribution is:

$$\text{Min}\, t \;\; s.t \begin{cases} F_0 + XF_X + YF_y + \sum_{i=1}^{n} T_i F_i \succeq 0 \\ tr(Y) \leq y \\ 0 \preceq X \preceq 2I \end{cases} \tag{2.5}$$

Chapter 3

Detailed Discussion

A reason for why we need an SDP approximation of $f(x)$ is that it contains the logarithm which belongs to a class of function called transcendental functions, while SDP usually work within the boundary of algebraic functions. Hence the need for approximation. Within this boundary of algebraic functions, there are a lot of ways that we can approximate the logarithm or $f(x)$, we will start with the simplest approach which use a lot of linear segments.

Another thing that the approximation differ from a faithful representation is the boundary, as $f(x)$ goes to inf at $x = 0$, we introduce ε, which defined the left boundary for x, so instead of $0 \leq x \leq 2$, we have $\varepsilon \leq x \leq 2$.

3.1 Linear segments approximation

The form for the epigraph of a linear segment is

$$(x, y) : \begin{cases} ax + b - y \geq 0 \\ 0 \leq x \leq 2. \end{cases} \tag{3.1}$$

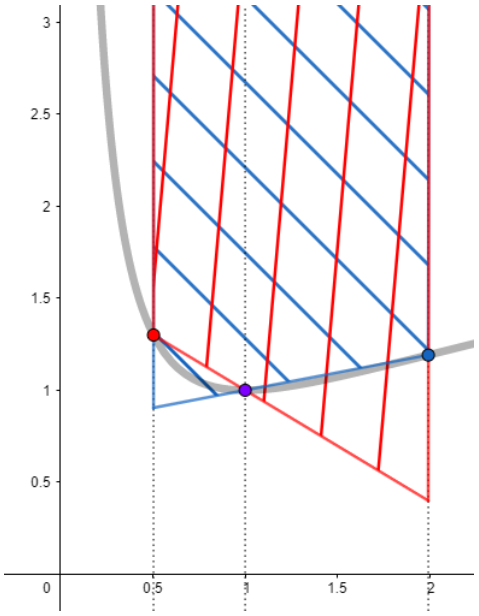


FIGURE 3.1: The epigraph combination of 2 linear segments is the intersection of their epigraph

A line segment naturally do not approximate a curve well, for that reason, we use many short segments to imitate the curvature of $f(x)$. And a nice thing about linear segment is that the combination of the their epigraph is the same as the intersection of their epigraph. Which mean they can be put together right away without any other constraints. And hence the epigraph of $f(x)$ in the region

$$(x, y) : \begin{cases} a_1 x + b_1 - y \geq 0 \\ a_2 x + b_2 - y \geq 0 \\ \vdots \\ a_n x + b_n - y \geq 0 \\ 0 \leq X \leq 2. \end{cases} \tag{3.2}$$

Equation 3.1 can be lift off in to a PSD constraints and in turn, Equation 3.2 can also be lift off the same way, providing a complete epigraph for Equation 2.5's

first and last constraint

$$(X,Y): \begin{cases} a_1X + b_1I - Y \succeq 0 \\ a_2X + b_2I - Y \succeq 0 \\ \vdots \\ a_nX + b_nI - Y \succeq 0 \\ 0 \preceq X \preceq 2I. \end{cases} \tag{3.3}$$

The problem now is to find appropriate line segments to produce the most accurate representation while using the least amount of segment. The segments would connect two points that sit on $f(x)$, i.e connecting $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Lets call these $x_1, x_2, ...x_n$ way-points. The question now would be how to space these way-points out for the best result. Spacing them out evenly would not produce good results as the function is very steep at the left side, which means it needs high resolution on these part while the right side is quite flat and do not require high resolution. For this reason, an adaptive scheme was implemented and will be discussed in the next subsection.

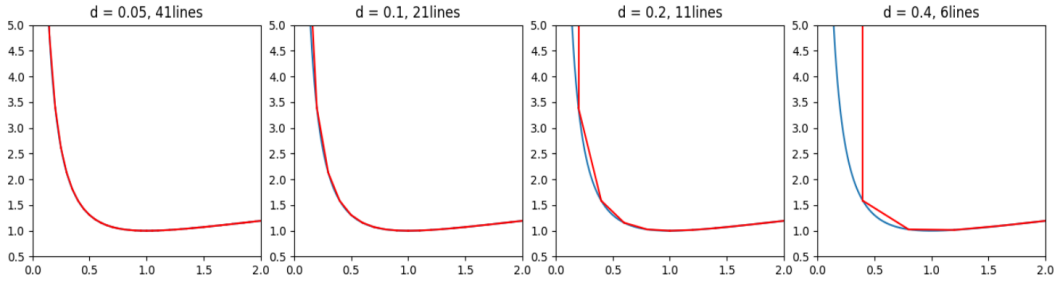


FIGURE 3.2: Linear segments with evenly spaced way-points

3.1.1 Adaptive scheme for choosing way-points

As we want to adjust the resolution in according to the curvature of $f(x)$, we introduced a new estimation parameter δ, define as the maximum absolute difference between the line through $(x_i, f(x_i))$ and $(x_j, f(x_j))$ in between x_i and x_j for consecutive i, j.

The adaptive scheme work by finding the largest distance between two consecutive way-points so that the error is less than a certain threshold. This can be done using binary search, the process can be summarized like so:

1: Start at $x_1 = 2$

2: Conduct binary search in the region between ε and $x_{current}$, to find the smallest value of x_{next} so that the error of the segment connecting them is less than δ.

3: Advance $x_{current}$ to x_{next} and repeat step 2 until $x_{current}$ is less than ε

4: Return the way-points x_1 to x_n

This process brings two benefits. One is that we can make effective uses of the segment. Two is that it is dependant on a reasonable criterion which is δ. So any adjustment made by the user would be much more intuitive.

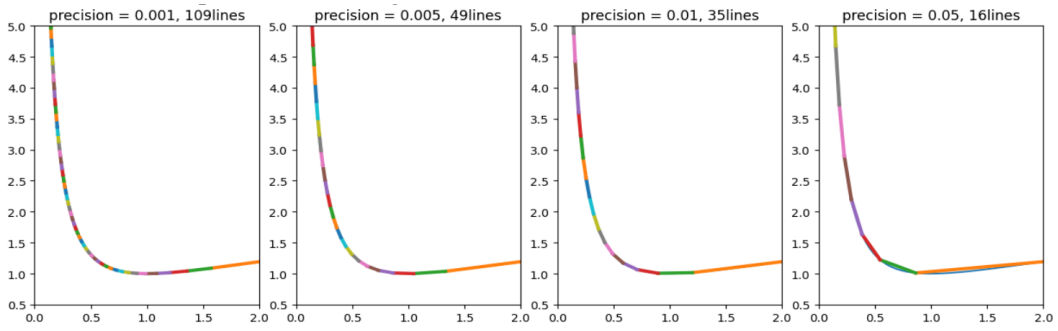


FIGURE 3.3: Linear segments with adaptive way-points

The linear segments method provide a good approximation of $f(x)$, but as $f(x)$ is very steep near 0, the amount of segment increase greatly once we increase the precision(i.e decrease δ) or expand the boundary(i.e decrease ε). The next logical step would be using segments that possess an inherent curvature. The next section will discuss the next step up from linear segments, which use cubic instead of a line.

3.2 Cubic segments approximation

A problem with cubic is that it is not strictly convex, and required a convex hull. The description of the convex hull is provided in [1], which stated that for a cubic $p(x)$ with $x \in [a, b]$ the convex hull of the cubic is

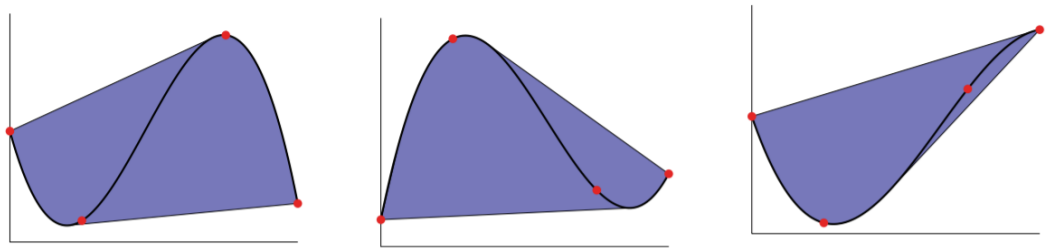


FIGURE 3.4: Convex hulls of the graphs of cubic polynomials on an interval.[1]

$$
\begin{aligned}
x_1 = a, \quad x_2 = a + \frac{1}{4}(b-a), \quad x_3 = a + \frac{3}{4}(b-a), \quad x_4 = b. \\
\begin{bmatrix} 3\alpha_2 & 0 \\ 0 & 12\alpha_4 \end{bmatrix} + \alpha_3 \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \succeq 0, \\
\begin{bmatrix} 3\alpha_3 & 0 \\ 0 & 12\alpha_1 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \succeq 0, \\
\sum_{i=1}^{4} \alpha_i = 1, \quad \sum_{i=1}^{4} \alpha_i x_i = x, \quad \sum_{i=1}^{4} \alpha_i f(x_i) = y.
\end{aligned} \tag{3.4}
$$

Where x_i are the interpolation points of the polynomial. And $x, y, \alpha_1, \alpha_2, \alpha, \alpha_4$ are scalar value.

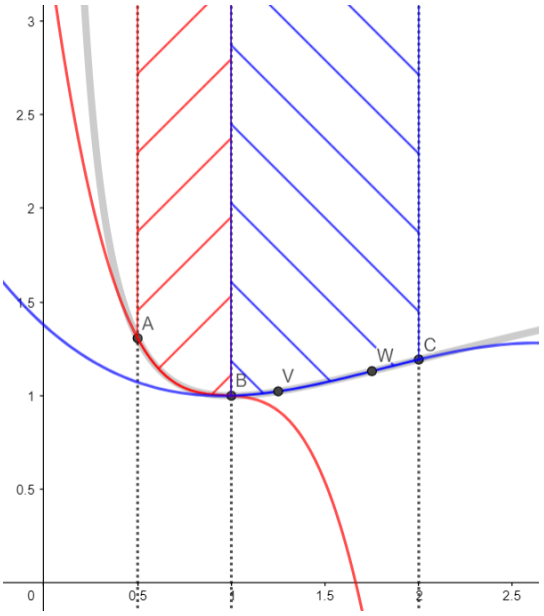


FIGURE 3.5: The epigraph combination of 2 cubic segments is the **union** of their epigraph

Two problem arises for the above representation. Firstly, we want all the points above the polynomial as well where the above expression does not contain. This can be resolve easily by making the expression for y become an inequality, $\sum \alpha_i f(x_i) \leq y$. Secondly, the combination of these epigraphs is a union instead of an intersection like in the linear segment case. And the union of these convex sets does not guarantee convexity. For this reason, we need to use the convex hull of the union, the formula for such set is [11]

$$conv(S) = \left\{ \sum_{i=1}^{n} \lambda_i s_i \middle| n \in \mathbb{N} \wedge \sum_{i=1}^{n} \lambda_i = 1 \wedge \forall i \in \{1, \ldots, n\} : \lambda_i \geq 0 \wedge s_i \in S \right\} \tag{3.5}$$

where s_i is the set containing all the points of the epigraphs, and λ is the scaling factor, serving a similar function to α_i in Equation 3.4. So to complete the representation for the cubic we first need to lift Equation 3.4 into a PSD constraint, then adjust the scaling factor λ_i and α_i so that they add up to 1. The

final representation is

$$\begin{bmatrix} 3\alpha_{2,i} & 0 \\ 0 & 12\alpha_{4,i} \end{bmatrix} + \alpha_{3,i} \begin{bmatrix} I & 2I \\ 2I & 4I \end{bmatrix} \succeq 0,$$
$$\begin{bmatrix} 3\alpha_{3,i} & 0 \\ 0 & 12\alpha_1 \end{bmatrix} + \alpha_{2,i} \begin{bmatrix} 1I & 2I \\ 2I & 4I \end{bmatrix} \succeq 0,$$
$$\sum_{j=1}^{4} \alpha_{j,i} = \lambda_i, \quad \sum_{j=1}^{4} \alpha_{j,i}\bar{x}_{j,i} = X_i, \quad \sum_{j=1}^{4} \alpha_{j,i} f(\bar{x}_{j,i}) = Y_i.$$
$$\sum_{i=1}^{m} \lambda_i = I, \quad \sum_{i=1}^{m} X_i = X, \quad \sum_{i=1}^{m} Y_i = Y.$$

where we have m cubics and $\bar{x}_{j,i}$ is the j^{th} interpolation point of the i^{th} cubic. The adaptive scheme described in previous Section 3.1.1 can also be used here.

Even though the curvature of the cubic helps reduce the amount of segment if we increase precision. Disappointingly, the cost of the convex hull is much larger than the linear segments, while not addressing the other problem. Which was $f(x)$ is very steep near $x = 0$, which means when reducing ε, it does not improve much compare to the linear segments while being much more costly.

3.3 Alternate linear segment approximation

Near $x = 0$, $f(x)$ was dominated by x^{-1} and if we want to stay with polynomial, we would need one with high degree which could be even more costly than the cubic. The reason for us to go with the approximation is only due to the logarithm as discuss in Chapter 3, and we can represent x^{-1} just fine. Which mean we could have the segment as $x^{-1} + ax + b$

$$x^{-1} + ax + b \leq y$$

$$\iff \begin{cases} x^{-1} \leq y_1 & \text{(3.6a)} \\ ax + b \leq y_2 & \text{(3.6b)} \\ y_1 + y_2 \leq y. \end{cases}$$

With this segment, a and b can be determine by using the adaptive approach discuss in Section 3.1.1 but instead of calculating error between the line and $f(x)$, the error is between the line and $ln(x)$. Then Equation 3.6b can be lift off into a PSD constraint just like regular segments discuss in Section 3.1. To lift Equation 3.6a, we can utilize Sylvester's criterion, which stated a matrix M is positive semi-definite if its top-left 1-by-1, 2-by-2,..., n-by-n matrix have positive determinant

$$\begin{cases} 1 - xy_1 \geq 0 \\ x \geq 0 \end{cases} \iff \begin{bmatrix} x & 1 \\ 1 & y_1 \end{bmatrix} \succeq 0. \tag{3.7}$$

So the representation of an alternate segment is

$$\begin{bmatrix} X & I \\ I & Y_1 \end{bmatrix} \succeq 0$$
$$aX + bI - Y_2 \preceq 0$$
$$Y_1 + Y_2 \preceq Y. \tag{3.8}$$

Once again we are dealing with an union of sets instead of an intersection(Figure 3.6). Which mean we need to make some adjustment to Equation 3.8

$$\begin{bmatrix} \lambda_i X_i & \lambda_i \\ \lambda_i & \lambda_i Y_{1i} \end{bmatrix} \succeq 0$$
$$a\lambda_i X_i + b\lambda_i - \lambda_i Y_{2i} \preceq 0$$
$$\lambda_i Y_{1i} + \lambda_i Y_{2i} - \lambda_i Y_i \preceq 0$$
$$\sum_{i=1}^{n} \lambda_i = I, \quad \sum_{i=1}^{n} \lambda_i X_i = X, \quad \sum_{i=1}^{n} \lambda_i Y_i = Y \tag{3.9}$$

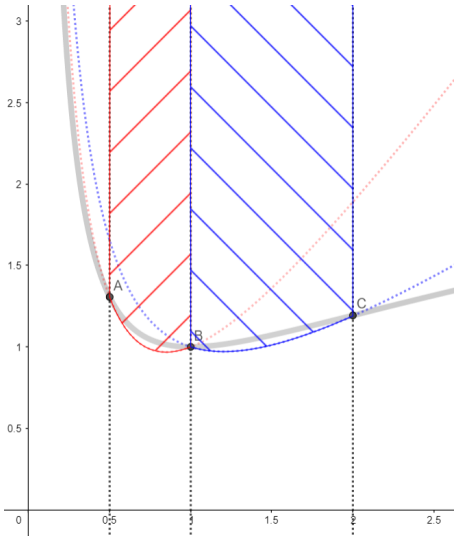


FIGURE 3.6: The epigraph combination of 2 alternate linear segments is the **union** of their epigraph

where we have n segments. The reason for multiplying all terms in the first three equations with λ_i of Equation 3.9 is that had we not, λ_i and X_i would be two separate variables bound by a multiplication constraint, and that is not allowed in SDP. So by multiplying all single X_i terms with λ_i, we can treat $\lambda_i X_i$ as a single variable and cancel the multiplication.

3.4 Final approximation

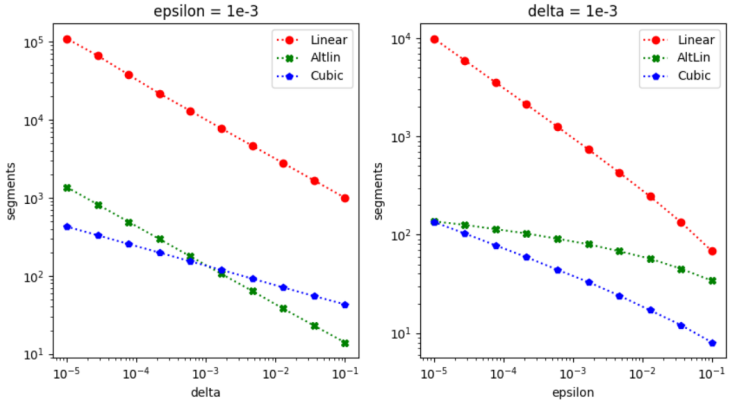


FIGURE 3.7: Comparison between segments type with different δ an ε

So far, the most costly segment is cubic segments due to a large amount of matrix inequality but it can have a low amount of segments. It is more and more worthwhile to use when δ is lower and lower. The alternate linear segments are a bit less costly, and the cost gets justify when ε is low. The linear segment is extremely cheap compare to the supposed improved options. But if the user requires low δ and ϵ, there could be 100 times more linear segment compare to the other two.

So in general, we can stay with linear segments as long as δ and ε are forgiving. But when ε is low, the alternate linear segment is an appealing choice. When δ is low, the cubic is marginally better than the alternate linear segment but we usually would not go too low on δ.

A thing about δ is, decreasing δ does not guarantee better results as there are overhead errors due to the other factor like the data. And improving the accuracy on our side would do nothing to those sort of errors. And so even though the improved segments are better with low δ, it is unnecessary for many cases.

Most of the segments are situated near 0 and a thing we could do is combining the property of the linear segment and the alternate linear segment. Here, let's introduced a new estimation parameter μ, defined as the boundary where we switch from using the linear segment to the alternate linear segment. So basically, we create a representation of $f(x)$ in the segment between $[\varepsilon, \mu]$ using alternate linear segment and a linear representation of $f(x)$ in the segment between $[\mu, 2]$. Then stitch them together using a convex hull of the union. This involved adding scaling parameters like λ to the linear segment epigraph. The final optimization

problem is

$$\min\ t\ \ s.t\ \begin{cases} \begin{bmatrix} \lambda_i X_i & \lambda_i \\ \lambda_i & \lambda_i Y_{1i} \end{bmatrix} \succeq 0 \\ a_{alt,i}\lambda_i X_i + b_{alt,i}\lambda_i - \lambda_i Y_{2i} \preceq 0 & \forall i \in [1,n] \\ \lambda_i Y_{1i} + \lambda_i Y_{2i} - \lambda_i Y_i \preceq 0 \\ \sum_{i=1}^{n} \lambda_i = \lambda_{alt}, \quad \sum_{i=1}^{n} \lambda_i X_i = X_{alt}, \quad \sum_{i=1}^{n} \lambda_i Y_i = Y_{alt} \\ a_{lin,j} X_{lin}\lambda_{lin} + b_{lin,j}\lambda_{lin} - Y_{lin}\lambda_{lin} \preceq 0 & \forall j \in [1,m] \\ \lambda_{lin} + \lambda_{alt} = I, \quad X_{lin}\lambda_{lin} + X_{alt} = X, \quad Y_{lin}\lambda_{lin} + Y_{alt} = Y \\ 0 \preceq X, X_{alt}, X_{lin}, X_i \forall i \preceq 2I \\ S_n^{-\frac{1}{2}} X S_n^{-\frac{1}{2}} = cov \\ \mathrm{tr}(Y) \leq t \end{cases}$$

where n is the number of alternate linear segments and m is the number of linear segments. $a_{alt,i}$, $b_{alt,i}$, $a_{lin,i}$ and $b_{lin,i}$ are generated from the adaptive scheme depending on the estimation parameters. S_n is the sample covariance calculated from the data. cov is the final result, the estimated covariance of the data. All other are matrices of size $d \times d$ except t, which is a scalar value of $F(X)$.

3.5 Relative Error

Throughout this chapter, we have been sticking to the same error function, which was the maximum absolute difference between the target and the approximation. Since $f(x)$ is very steep at 0, a larger error at this end would not have a drastic change to the function. In this region the error even comes from the fact that we need to stitch the epigraphs with convex hull and the error figure provided by the user does not tell the true story. Another thing that was observed is that

the result of the solver rarely goes into this region which justifies setting a more lenient error figure at this side.

So we introduced an option to choose to switch to relative error instead. Which is $\frac{err(x)}{f(x)}$. So an error of 1 at $f(x) = 1000$ would result in the error figure of 10^{-3}. This improves run time a lot because it reduces the amount of the segment even more

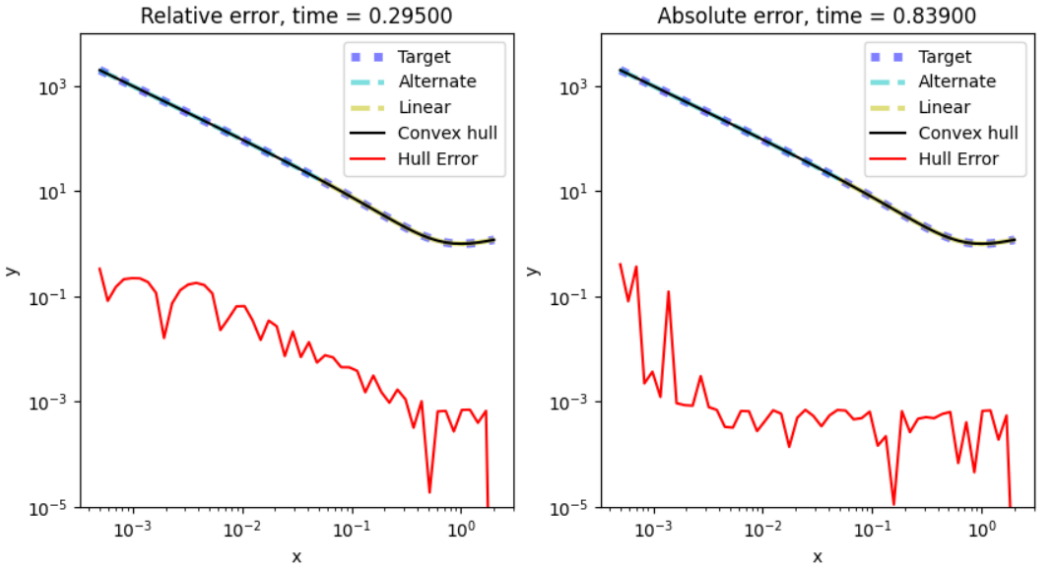


FIGURE 3.8: Relative error versus Absolute error when $\delta = 7 \times 10^{-4}$, $\varepsilon = 5 \times 10^{-4}$ and $\mu = 2 \times 10^{-2}$

Chapter 4

Overview

This chapter discusses how everything we have discusses in the previous chapter is put together into a user-friendly package that disguises all the complicated math behind it. We will go through the block diagram in this chapter the same way the data travel, from input through to the result.

The package is available online at:

https://github.com/pcbach/LinGaussCov

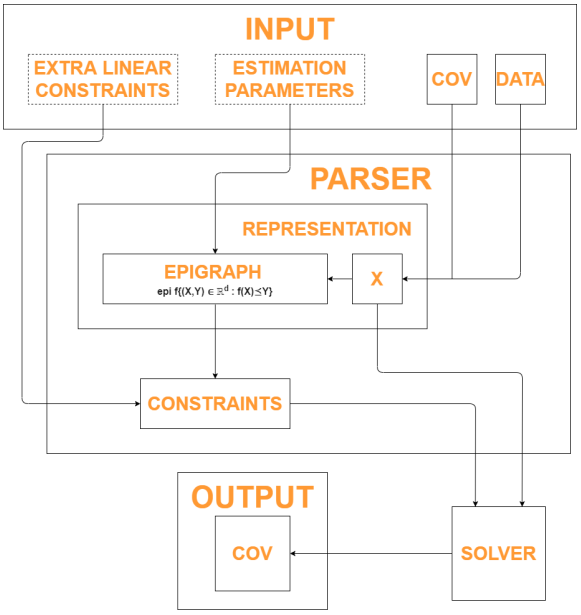


FIGURE 4.1: Block diagram of the parser

4.1 Block diagram

4.1.1 Input

This section describes the input, where the user interacts with the package to describe what they want to solve. There are four inputs that the user can provide for the parser, two of which is optional:

Cov: This is the `CVXPY` matrix variable of size $d \times d$, the value of this variable will change at the end and is also the output of the solver.

Data: This is the data that is used to calculate the sample covariance (S_n) and is an `numpy` array of size $n \times d$

Estimation parameter: (Optional) This is the four parameters discuss in the previous chapter: δ, ϵ, μ and error figure `max` or `snr`. The user can choose not to specify this and a default value will be used.

Extra linear constraint: (Optional) This is any linear constraint that the user wishes to put upon cov.

4.1.2 Parser

As previously stated in Equation 2.5, the epigraph discuss in Chapter 3 will be constraints for the solver while the objective function is t where $t = tr(Y)$. First, the adaptive scheme in Section 3.1.1 is used to find all the coefficients for the lines. Then the Equation 3.10 can just be implemented straightforwardly and pack into a list of constraints.

Then any other linear constraints in the input can then also be added to the constraint list.

4.1.3 Solver

The solver used here is `MOSEK` as it is probably the best option provided with `CVXPY`. Even though it is shown as a separate block from the parser, the process of interacting with the solver is also packaged inside. The user essentially does not need to interact with the underlying mathematics content.

4.1.4 Output

The return value of the solver is the cov that is given by the user at first.

4.2 Usage example

4.2.1 Simplest example

Here we just solve the base function without any data, the result should be an identity matrix. Code:

```
import numpy as np
import cvxpy
import ps
d = 6
cov = cvxpy.Variable((d, d),PSD = True)
Problem = ps.LGC(cov, d, data = [])
ans = Problem.solve(solver = cvxpy.MOSEK)
print(ans)
print(Problem.time)
```

The `Problem.time` is actually the solver time and not real time. Output:

```
[[ 1. -0.  0. -0. -0.  0.]
 [-0.  1. -0.  0.  0.  0.]
 [ 0. -0.  1. -0. -0.  0.]
 [-0.  0. -0.  1.  0.  0.]
 [-0.  0. -0.  0.  1. -0.]
 [ 0.  0.  0.  0. -0.  1.]]
0.765
```

4.2.2 Source covariance example

Here we generate data from a source covariance, and use that data to find the covariance, the result should be similar to this source covariance. A constraint for ones in the diagonal is added. Code:

```
import numpy as np
import cvxpy
import ps
d = 6
covariance = [
 [ 1.   ,0.08  ,-0.48 ,-0.04 ,-0.18 ,-0.32],
 [ 0.08 , 1.   , 0.02 ,-0.08 , 0.38 , 0.1 ],
 [-0.48 , 0.02 , 1.   ,-0.03 , 0.44 ,-0.1 ],
 [-0.04 ,-0.08 ,-0.03 , 1.   ,-0.27 ,-0.14],
 [-0.18 ,0.38  , 0.44 ,-0.27 , 1.   ,-0.2 ],
 [-0.32 ,0.1   ,-0.1  ,-0.14 ,-0.2  , 1.  ]]
mean = np.zeros(d)
data = np.random.multivariate_normal(mean, covariance, 5000)
cov = cvxpy.Variable((d, d),PSD = True)
Problem = ps.LGC(cov, d, data = data)
Problem.add_constraint([cov[i,i] == 1 for i in range(d) ])
ans = Problem.solve(solver = cvxpy.MOSEK)
print(ans)
print(Problem.time)
```

Output:

```
[[ 1.     0.077 -0.48  -0.035 -0.186 -0.315]
 [ 0.077  1.     0.016 -0.08   0.376  0.102]
 [-0.48   0.016  1.    -0.032  0.443 -0.105]
 [-0.035 -0.08  -0.032  1.    -0.273 -0.144]
 [-0.186  0.376  0.443 -0.273  1.    -0.2  ]
 [-0.315  0.102 -0.105 -0.144 -0.2    1.   ]]
1.312
```

Chapter 5

Experiments

This chapter discusses the experiment conduct with the parser. From trying out the best performing estimation parameters to fitting synthetic data as well as real-world data.

5.1 Estimation Parameter Experiment

This section carries experiments on the estimation parameters to determine a good default setting for the user.

5.1.1 Delta

The solving time is usually higher with tighter constraints, which is obvious, the outlier is due to other processes also running on the computer at the same time and could affect performance. But we can see that in this particular run, the error kind of improved up until a point around $5 \times 10^{-}4$ and flat out as δ is decrease further. This holds for other runs as well, which is expected as there are overhead errors as discussed before and we can only improve things on our end to a certain degree.

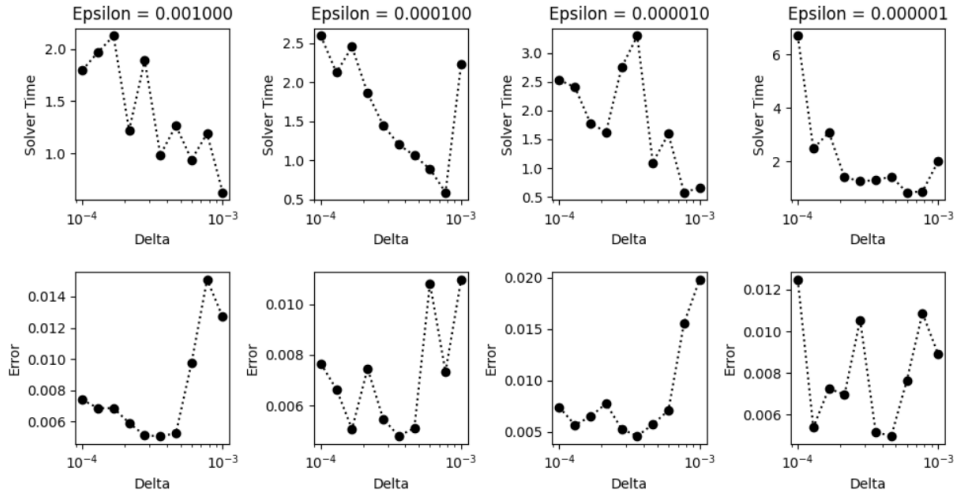


FIGURE 5.1: Solver time and error figure with different δ and ε values on a 6 by 6 matrix

From these observations, the default δ value is $5 \times 10^{-}4$.

5.1.2 Epsilon

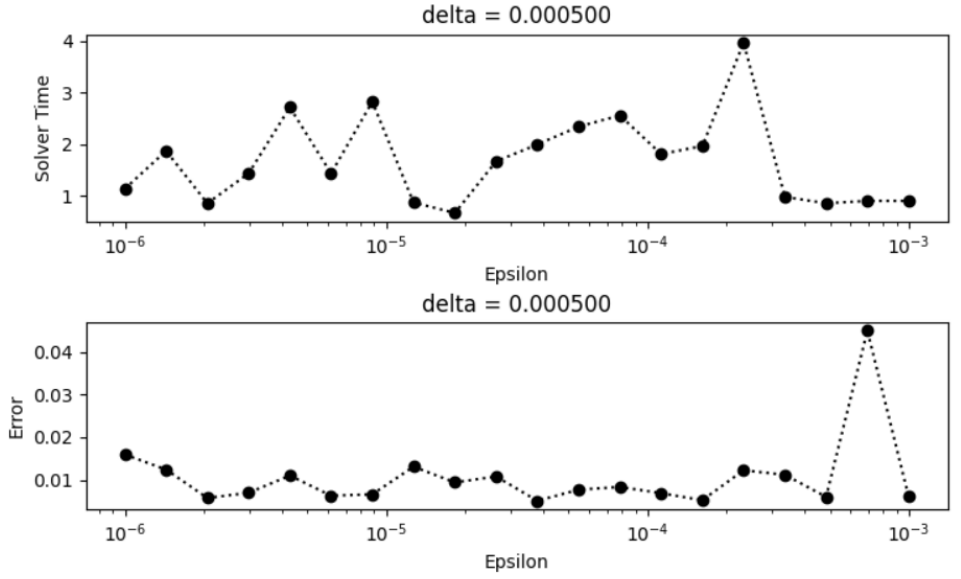


FIGURE 5.2: Solver time and error figure with different ε values on a 6 by 6 matrix

Changing ε actually would not have a large effect on the solver unless X is close to 0. But having a tight ε also does not increase the solver time much, so we set it to a reasonably small value for the default setting, which is at $1 \times 10^{-}5$

5.1.3 Mu

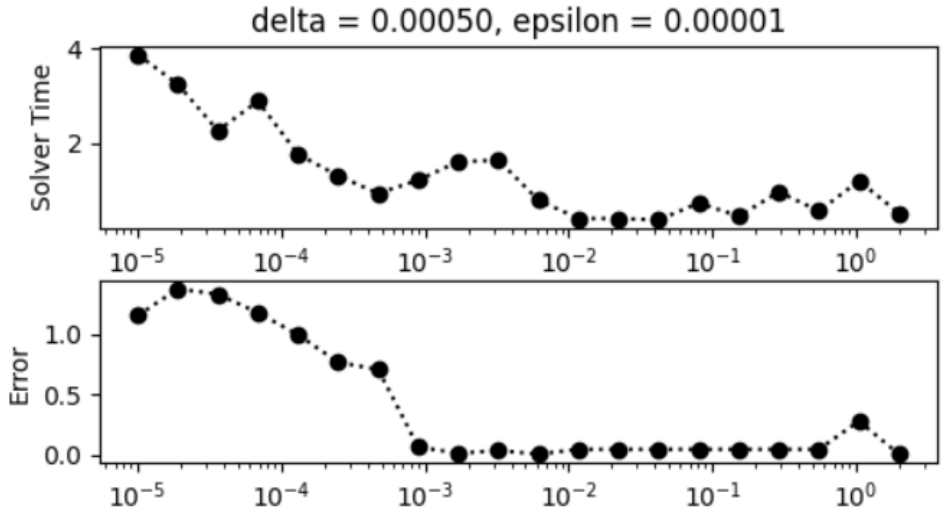


FIGURE 5.3: Solver time and error figure with different μ values on a 6 by 6 matrix

A rather interesting occurrence happens when μ is low, this is because at this boundary, the linear segments just way too inadequate to approximate $f(x)$ with relative error. While setting a low μ value essentially forces the parser to use the linear segments. If we set it to absolute error, there would be much more segments (about 20 times at $\mu = 10^{-4}$ and around 70 times at $\mu = 10^{-5}$). So the linear segments are unusable at this region and μ work the best around 10^{-2} and should not be any lower than 10^{-3}.

5.1.4 Experiment Conclusion

The default value that was decided after these experiment is $\delta = 5 \times 10^{-}4, \varepsilon = \varepsilon$ and $\mu = 10^{-2}$.

Another discovery made during this experiment is that the interface between `CVXPY` and `MOSEK` seems to be inefficient and problem take a lot more time to set up compare to the time it take to solve. Which is why having 20 to 70 more times segments in the experiment with μ when using absolute error is so indesirable.

5.2 Correlation matrices

5.2.1 Experiment setup

As mentioned before, solving the MLE with the covariance matrix while using an off-the-shelf solver bring the benefit of adding linear constraints on the covariance matrix. A simple example of this is the correlation matrix where the diagonal is 1.

To conduct this experiment, a correlation matrix is randomly generated. Then data is then generated from this source covariance. A subset of this data is then used to calculate the sample covariance and use as data for the solver. The solver is solved with default estimation parameters and a constraint where the diagonal entries are all 1. Then the likelihood function value of these two answers is calculated and compare.

The accuracy here is the value of the likelihood function when scored with the data-generating covariance matrix.

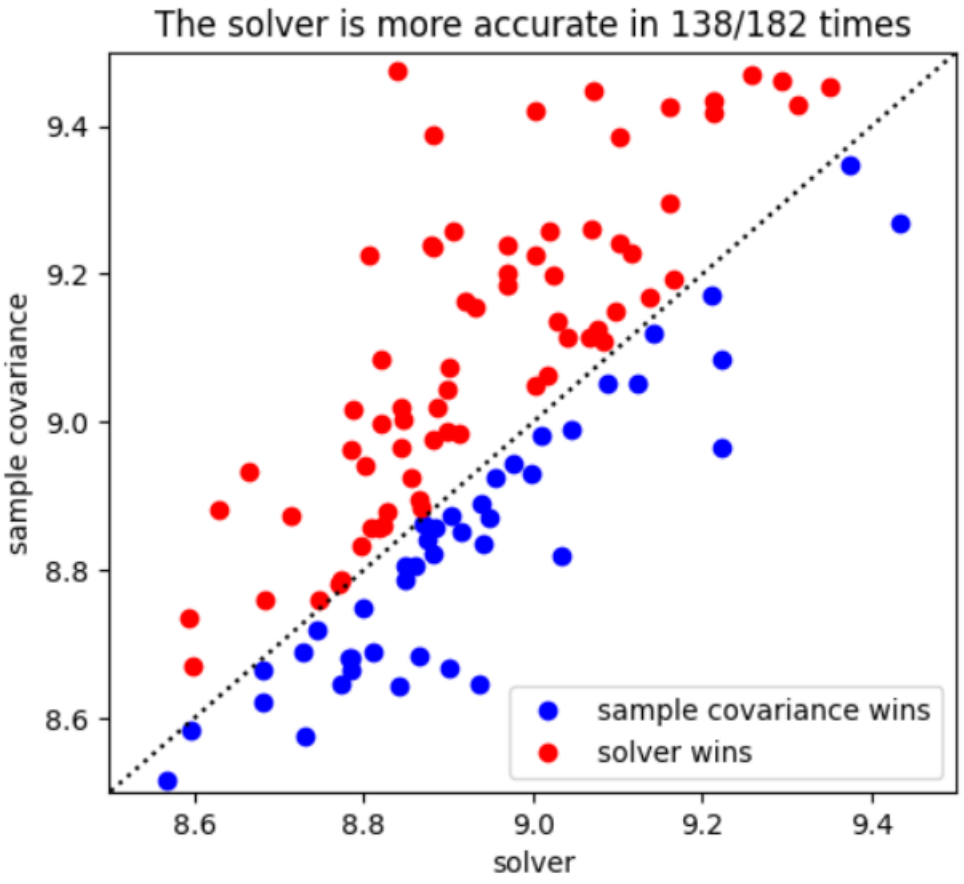


FIGURE 5.4: Solver performance versus sample covariance performance

5.2.2 Result

The result is non-decisive on which is more accurate, which is expected because the constraints is not too special and the sample covariance is usually a good estimator.

5.3 Brownian Tree Motion Model

5.3.1 Experiment setup

Another type of constraint that regularly appears in engineering problems is the Brownian motion tree model constraint. Given a rooted tree T on a set of node $V = \{1, \ldots, r\}$ and with p leaves, where $p \leq r$, the corresponding Brownian motion tree model consists of all covariance matrices of the form

$$\Sigma_v = \sum_{i \in V} v_i e_{de(i)} e_{de(i)}^T \tag{5.1}$$

Where $e_{de(i)} \in \mathbb{R}$ is a 0/1-vector with entry 1 at position j if leaf j is a descendant of node i and 0 otherwise. Here the parameter v_i describe branch length and the covariance between any two leaves i, j is the amount of shared ancestry between these leaves[4].

traceA structure with such structure that comes to mind is the internet. If you send data packets to 2 different websites, a variance on a shared path between the two websites would affect both sides and the covariance between them should be proportional to the length of this shared path.

To conduct this experiment, 8 universities around the world was chosen. We purposely choose these websites as they seem to host their website on-site and not through a third party which could behave differently. We pinged 8 websites at the same time to collect the round trip time data for a long period to form a dataset (1000 times each website throughout the day). Then we ran the solver without the structure first and determine a reasonable tree structure from studying the covariance matrix and determine the tree structure by hand. Then, we ran the solver again, this time with only 15 data points but with a structure, and compare that to the sample covariance.

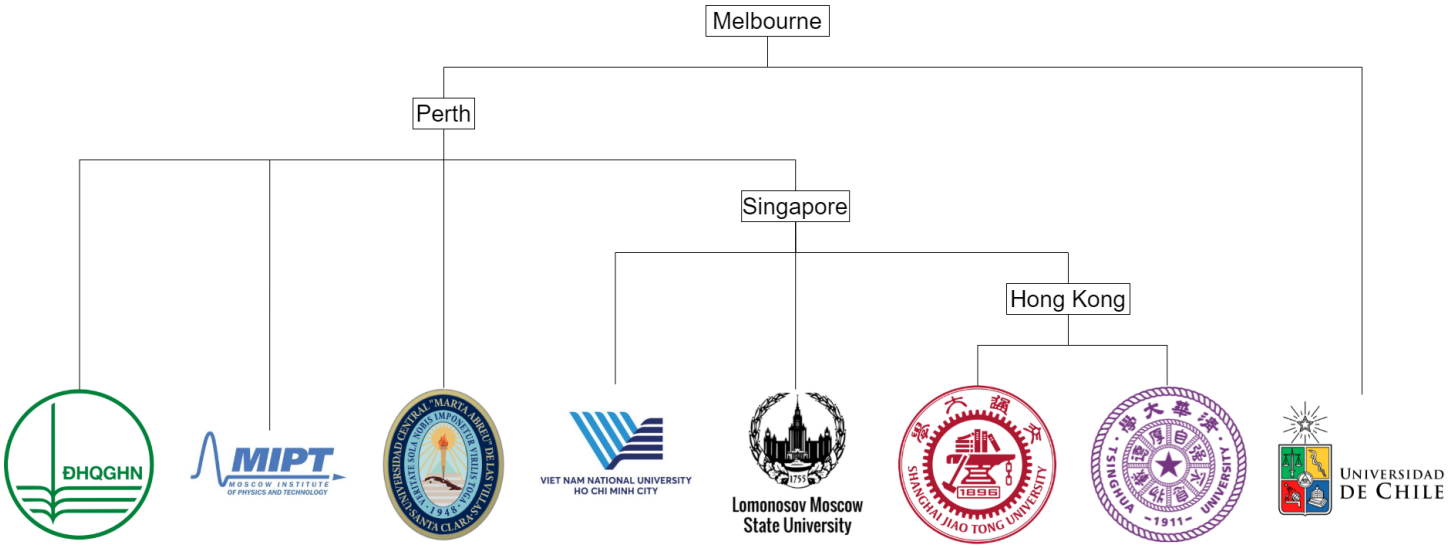


FIGURE 5.5: Proposed network tree structure

5.3.2 Result

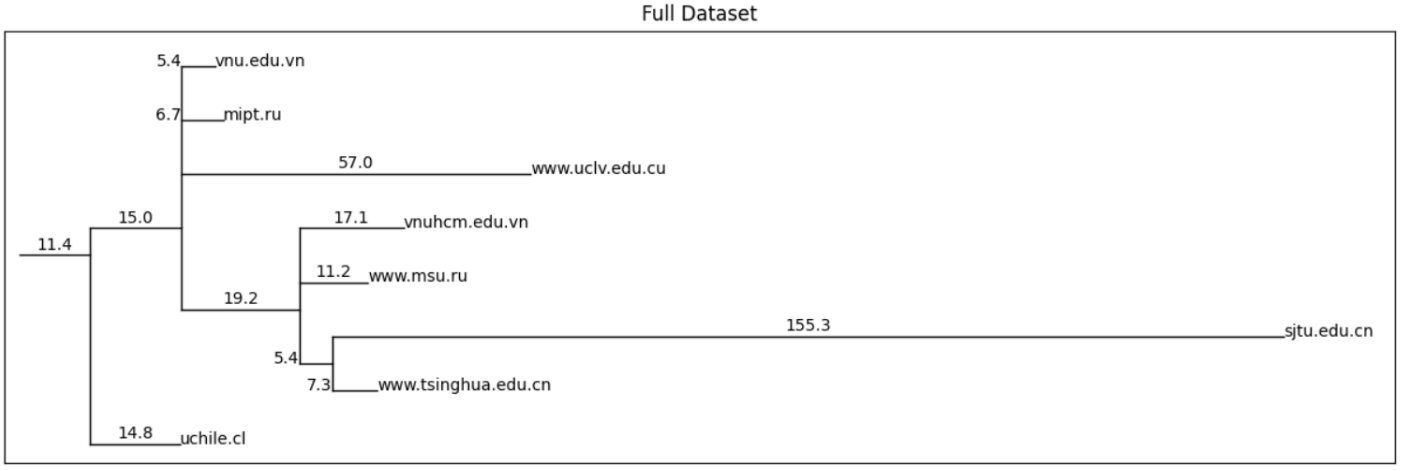


FIGURE 5.6: The solver ran with a full dataset

As discussed in previous experiment, as the correlation matrix constraint is not that complicated, the sample covariance perform comparable to the solver. However in this case, with a more complicated structure, we should expect a more significant difference in performance. Which is very clearly observed on the right.

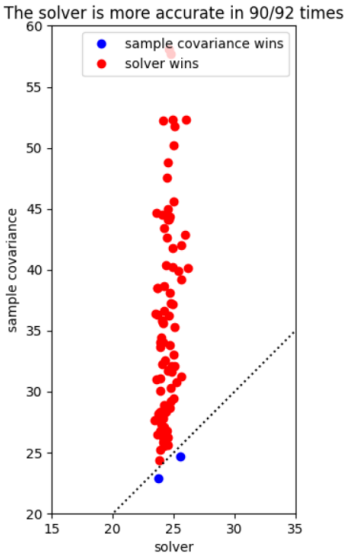


FIGURE 5.7: Solver performance versus sample covariance performance

A nice thing that came as a byproduct is that if we overlaid the output the subset's tree structure over the full data's tree structure we can kind of see which path is behave abnormally for this particular subset of data.

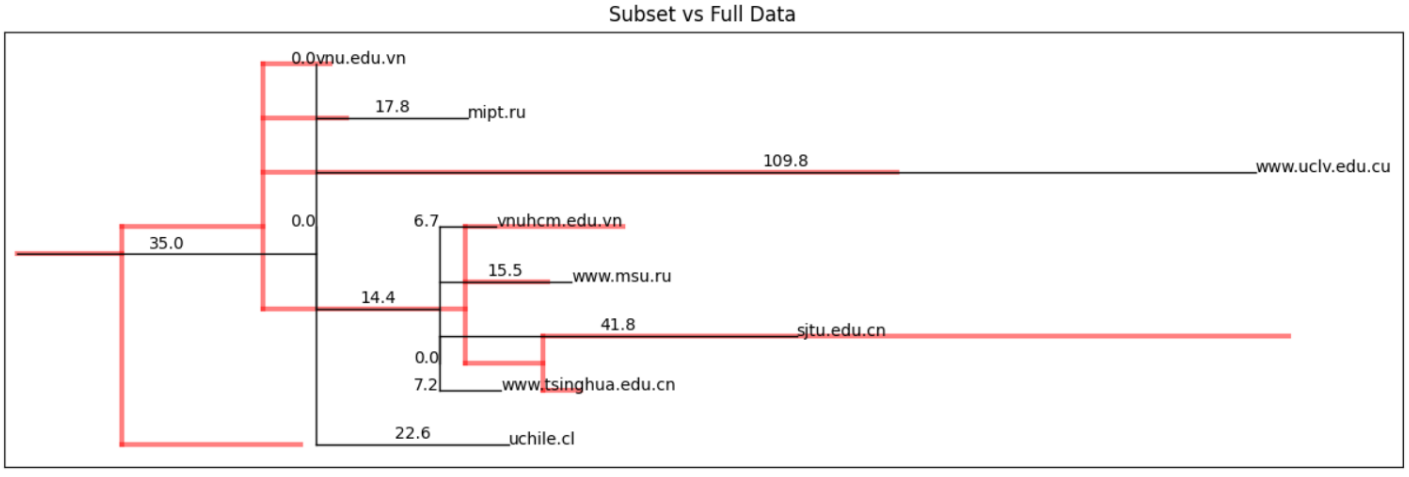


FIGURE 5.8: The subset tree structure(black) is laid over the full dataset tree structure(red)

Chapter 6

Conclusion

Solving the MLE is a very popular problem but is not always an easy task for many people. While there are many situations where the sample covariance does not provide adequate results. This project succeeded in creating a parser that encapsulated a semi-definite approximation of the Gaussian likelihood function. Which could help people solve the MLE with constraints on the covariance matrix using an off-the-shelf solver with very short syntax with acceptable accuracy.

The parser is also proven to work with complicated constraints like a Brownian Tree Model. Making adding constraints on the MLE problem easier than ever. This could help solve many engineering problems where a structured covariance matrix is involved.

There are still limitations and improvements that can be done on this project and will be discussed in the following section.

6.1 Limitation

As we are dependant on the solver, we carry with us any drawback of the solver.

The parser can be quite slow at times. This usually happens when setting up problems with lots of constraints. This could be due to the parser inefficiency or the `CVXPY-MOSEK` interface inefficiency.

The problem created by the parser can be unsolvable at times for the solver. Once again the cause of this is unknown, it could be the parser fault, or `MOSEK` fault, nothing can be sure until we conduct further experiments.

6.2 Future Work

More experiments can be conducted to determine the problem stated in the previous section. This can range from reviewing our code to delve deep into `CVXPY` source code

As mentioned in an earlier chapter, the approach we take is not the only one, there has been similar work on semi-definite approximation of other functions like the logarithm[9]. Where they used integral approximation instead. This is quite different from our approach. Since from another viewpoint, they are approximating the function by stacking functions on top of each other, while we work from left to right. A problem with our case is that when using an integral representation on $f(x)$ it turns into an integral of a rational function with degree 2, which seems more complicated than the logarithm case.

We have also try another approach using the Taylor series where we break $f(x)$ up into many polynomials. This poses some promising property like how it preserves the convexity but overall seems to be much more complicated than the method we are using and requires more time to devise a reasonable solution.

Another thing regarding `Python`, eventhough it seems appealing to casual user and beginners. In a more academic setting, a package in R might be more appealing to professionals. And releasing a R package will be the next thing in our list.

References

[1] P. P. G. Blekherman and R. Thomas, *Semidefinite optimization and convex algebraic geometry.* Philadelphia: SIAM, 2012, ch. 3, p. 69.

[2] J. J. Warnes and B. D. Ripley, "Problems with likelihood estimation of covariance functions of spatial gaussian processes," *Biometrika*, vol. 74, no. 3, pp. 640–642, 9 1987.

[3] C. Stein, "Inadmissibility of the usual estimator for the mean of a multivariate normal distribution," in *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability.* Berkeley, CA: University of California Pres, 1956, p. 197–206.

[4] C. P.Zwiernik and D.Richards, "Maximum likelihood estimation for linear gaussian covariance models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 79, no. 4, pp. 1269–1292, 4 2016.

[5] M. Pourahmadi, "Covariance estimation: The glm and regularization perspectivess," *Statistical Science*, vol. 26, no. 3, pp. 369–387, 2011.

[6] P. Z. B. Sturmfels, S. Timme, "Estimating linear covariance models with numerical nonlinear algebra," *Algebraic Statistic*, vol. 11, no. 1, pp. 31–52, 2020.

[7] T. W. Anderson, "Estimation of covariance matrices which are linear combinations or whose inverses are linear combinations of given matrice," in *Essays in Probability and Statistic.* Chapel Hill, N.C: University of North Carolina Pres, 1970, pp. 1–24.

[8] J. Felsenstein, "Maximum-likelihood estimation of evolutionary trees from continuous characters," *American Journal of Human Genetics*, vol. 25, p. 471–492, 2020.

[9] J. S. H. Fawzi and P. Parrilo, "Semidefinite approximations of the matrix logarithm," *Foundations of Computational Mathematics*, vol. 19, no. 2, pp. 259–296, 2018.

[10] G. Sagnol, "On the semidefinite representation of real functions applied to symmetric matrices," *Linear Algebra and its Applications*, vol. 439, no. 10, pp. 2829–2843, 2013.

[11] M. H. B. Gartner and E. Welzl, "Convex hull," Available at: https://www.ti.inf.ethz.ch/ew/courses/CG13/lecture/Chapter%203.pdf (2013-01-10), accessed: 2020-31-10.