# Schrödinger Bridges in Optimal Transport

by

Luna Dams

to obtain the degree of Bachelor of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Thursday July 10, 2025 at 11:30 AM.

Student number: 5542995
Project duration: April 1, 2025 – July 10, 2025
Thesis committee: Dr. ir. G. N. J. C. Bierkens, TU Delft, supervisor
Dr. R. C. Kraaij, TU Delft

**Delft institute of Applied Mathematics**
**Faculty of Electrical Engineering, Mathematics and Computer Science**

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

# Preface

In the 18th century, Gapsard Monge [8] first introduced the Optimal Transport problem with a rather intuitive, informal example: suppose a construction worker has to shovel a large pile of sand to a new location on the construction site. This worker takes multiple trips carrying heavy loads, and has to rebuild the pile in a possibly different shape at the new location. One can imagine the burden of this task and might want to analyse the most efficient way to move the sand.

Despite the intuitive character of this problem, it has been proven challenging to solve. In the early 20th century, Erwin Schrödinger introduced what is now known as the Schrödinger Bridge problem - providing a new perspective of this problem on the base of stochastics. The focus on this thesis lies on deriving the solution to the Schrödinger bridge problem, by using both an exact analytical and numerical methods.

# Summary

This thesis examines the connection between the Optimal Transport (OT) and the Schrödinger Bridge (SB) problem. Both analytical and numerical approaches to finding the optimal solution are discussed.

Originating from an engineering perspective, Optimal Transport seeks to find the minimal cost of transporting mass from one distribution to another. In the reformulation to the SB problem, these distributions are considered probability densities, and the solution can best be described as the most likely transition from the initial density to the other, expressed as a probability law. The optimal probability law is found by minimizing its Kullback-Leibler (KL) divergence when compared to a prior.

The theoretical mathematical foundations of the OT and SB problems are first explained. Here, the OT problem is approximated with an entropy regularisation term, which links it directly to the SB problem, by means of the KL-divergence.

To derive the optimal solution of the SB problem, its similarity with the discrete entropic OT problem is discussed. The existence of the unique solution to the discrete problem is found by its strict convexity. With the Lagrangian, the form of the solution it is established. In the continuous case, an iterative scheme to finding the solution is developed using similar computations. Scaling functions that define the unique optimal probability are obtained.

For one-dimensional Gaussian marginals, analytical update schemes are derived. First this is done for marginals with zero mean. For the Gaussians with non-zero mean, a more complicated derivation is explained, that considers the shifts in the mean.

Python implementations, found in appendices C, D, E and on GitHub [1], are made for all three update schemes, using numerical integration. The performance of the algorithms is compared, and their results show to be extremely similar. The general algorithm converges faster, but also takes more numerical approximations.

From the newly made computational general algorithm, an extension is derived that visualises intermediate marginals over a time grid, creating the "bridge". This is done for both Gaussian and non-Gaussian marginals. It can here be seen that generally, the algorithm performs better when marginals are "standard" Gaussian.

All in all, the Schrödinger Bridge provides a useful algorithm that can be used to solve potentially complicated OT problems. Both numerical implementations as analytical approaches give useful tools for finding solutions effectively. Additionally, bridge visualisations can be made neatly for a broad range of marginals.

Throughout the process of writing this thesis, the AI programme ChatGPT was consulted for suggestions on academic sources with supporting arguments. Furthermore, it has been implemented to check for small errors in Python code as well as spelling, grammar and idiom mistakes, whilst maintaining the human-written text.

[1] Available at `https://github.com/LunaDams/Schrodinger-Bridge-Python-Codes`

# Contents

# 1 Introduction

At its essence, OT describes the transport from an initial distribution to a target one in the most cost-effective way. The problem originating in the engineering field can be viewed as a mathematical optimization problem, where the total cost is minimised, while considering the individual cost of each element - in Monge's [8] example, this means minimising a worker's total labour of moving a sand pile, while taking into consideration the effort it takes to move each individual grain.

Although the basic idea behind OT may be intuitive, finding the solution of such problems often is not: when distributions become non-standard or exist in high-dimensions, computing the analytical solution can become near impossible. It took a renewed interest in numerical analysis and the invention of computers in the early to mid 20th century [3] to create computational algorithms capable of solving the complex OT problem. Recently, the theory has regained popularity due to its use in machine learning [10] and the possibility to solve large-scale problems.

One important insight in the mid 20th century was that of Erwin Schrödinger: instead of viewing the distributions from an engineering or physical standpoint, Schrödinger analysed these distributions as probabilities [4]: turning the OT problem into a stochastic one. His goal was to find the so-called Schrödinger Bridge (SB): the probability law that would transport one distribution into the other, which is optimal when it would be the "most likely", compared to a prior, or expected, probability law describing the transport flow. This thesis focuses on finding solutions to the Schrödinger Bridge problem, by using both analytical and numerical computations.

First, the preliminaries on Optimal Transport and its link to the Schrödinger Bridge are given in chapter 2, where the mathematical denotation of the problem is discussed.

In chapter 3, the underlying theory for finding the algorithmic solution to the Schrödinger Bridge is explained, partially by looking at its discrete counterpart. where existence and uniqueness of the discrete solution will also be established. For the continuous SB problem, deriving this is omitted due to its complexity. Section 3.2 thus focusses completely on computing the update scheme for finding the solution, using similar techniques as the ones used for the discrete problem.

From the underlying theory, analytical solutions for one-dimensional Gaussian distributions are found in chapter 4, where both the distributions with a zero mean and a non-zero mean, respectively, are analysed. These solutions provide new algorithms, specific to their examples.

For the general theoretical update scheme as well as for the Gaussian cases, computational algorithms are coded using Python, and their results will be compared in chapter 5.

Finally, chapter 6 shows visualisations of the transport through intermediate distributions found with the SB algorithm. Both for Gaussian and non-Gaussian marginals, such computations are made. The effect of different parameters on the performance of the algorithm is discussed here.

# 2

# Preliminaries on Optimal Transport and the Schrödinger Bridge

## 2.1. Preliminaries on Optimal Transport

Suppose that for the example of the construction worker, a random variable $X$ describes the original locations of the grains of sand, and a random variable $Y = T(X)$ their target locations, where $T$ is the transport map over which the sand grains move. The cost of moving a grain of sand is usually determined by its moving distance. If a grain is located at location $x_i$, and moves to location $y_j$, then for all $x_i \in X$, $y_j \in Y$, the transport costs are stored in a so-called cost matrix $C_{i,j} = c(x_i, y_j)$.

By the use of random variables, the problem above is already approached from a probabilistic viewpoint, as Schrödinger suggested - this is a more relaxed and simplified version of Monge's original problem. Since this paper will solely focus on the stochastic approach, the original physical problem will not be explained in detail here. For the interested reader, the book by Peyré and Cuturi [9] gives an elaborate break-down of this original, more general, problem.

Instead of the map $T$, by the stochastic nature of the problem, the transition matrix $P_{i,j}$ can now be used to describe the flow from $x_i$ to $y_j$, for any $x_i \in X$, $y_j \in Y$. In fact, $P_{i,j}$ is a coupling matrix of the distributions $a$ and $b$ of $X$ and $Y$ respectively, by the use of definition 1.

**Definition 1.** *(Coupling matrices)*
*For discrete distributions $(a, b)$ of random variables $(X, Y)$, the set of coupling matrices $U$ is defined by:*

$$U(a, b) := \left\{ P \in \mathbb{R}_+^{n \times n} : P\mathbf{1}_n = a,\, P^T\mathbf{1}_n = b \right\},$$

*where $\mathbf{1}_n$ is the vector of ones of length $n$.*
*If $P \in U(a, b)$, then $P$ is called a coupling matrix of $(a, b)$.*

With the matrices $C_{i,j}$ and $P_{i,j}$, the OT problem can now be defined formally:

**Definition 2.** *(Optimal transport - discrete case)*
*For discrete measures $a$ and $b$, and cost matrix $C_{i,j}$, the optimal transport (OT) problem is given by:*

$$L_C(a, b) := \min_{P_{i,j} \in U(a,b)} \langle P, C \rangle = \min_{P_{i,j} \in U(a,b)} \sum_{i,j} C_{i,j} P_{i,j}.$$

Naturally, one can imagine that when large piles of sand need to be moved, taking each individual grain of sand into consideration - which can add up to millions, if not billions of grains - may not be very efficient and will create extremely large scale problems. Alternatively, the piles of sand will be modelled as densities by the use of continuous random variables $\mathcal{X}$ and $\mathcal{Y}$, with corresponding probability density functions $\alpha$ and $\beta$. Now the cost matrix shall be replaced by a cost function $c(x, y)$, and instead of the

transition matrix, the coupling $\pi(x, y)$ is used, which is a joint distribution with marginals $\alpha$ and $\beta$.

**Definition 3.** *(Coupling set)*
*For probability density functions $(\alpha, \beta)$ of continuous random variables $(\mathcal{X}, \mathcal{Y})$, the set $\mathcal{U}$ of joint distributions with marginals $\alpha$ and $\beta$ is defined by:*

$$\mathcal{U}(\alpha, \beta) := \left\{ \pi(x, y) : \mathbb{R}^n \to \mathbb{R}^n : \int_{\mathbb{R}^n} \pi(x, y) \mathrm{d}y = \alpha, \int_{\mathbb{R}^n} \pi(x, y) \mathrm{d}x = \beta \right\}.$$

*If $\pi \in \mathcal{U}(\alpha, \beta)$, then $\pi$ is called a coupling of $\alpha$ and $\beta$.*

With this cost function and coupling, an analogous definition can be given for the continuous OT problem:

**Definition 4.** *(Optimal transport - continuous case)*
*For continuous measures $\alpha$ and $\beta$, and cost function $c(x, y)$, the optimal transport (OT) problem is given by:*

$$\mathcal{L}_c(\alpha, \beta) := \min_{\pi \in \mathcal{U}(\alpha, \beta)} \int_{\mathbb{R}^n \times \mathbb{R}^n} c(x, y) \mathrm{d}\pi(x, y).$$

Note that the definition of the continuous case may also hold for discrete measures.

In an article by Chen et al. [4], it is stated that especially in higher dimensions, finding the solution of the OT problem is challenging due to its size. It is suggested to find an approximate solution by adding noise in the form of an entropy term:

**Proposition 1.** *(OT with entropy regularisation)*
*The OT problem of definition 4 can be approximated with an entropy regulariser, that is:*

$$\mathcal{L}_c(\alpha, \beta) \approx \min_{\pi \in \mathcal{U}(\alpha, \beta)} \int_{\mathbb{R}^n \times \mathbb{R}^n} c(x, y) \mathrm{d}\pi(x, y) + \varepsilon \int_{\mathbb{R}^n \times \mathbb{R}^n} \pi(x, y) \log(\pi(x, y)) \mathrm{d}x \mathrm{d}y. \tag{2.1}$$

Here, the value of $\varepsilon$ determines the amount of added noise: if the limit $\varepsilon \downarrow 0$ is taken, then the exact solution, although possibly unstable, can be found. As can be seen in figure 2.1, taking a larger value of $\varepsilon$ gives a less accurate solution, that is "flattened out": the peaks are dampened to avoid instability.

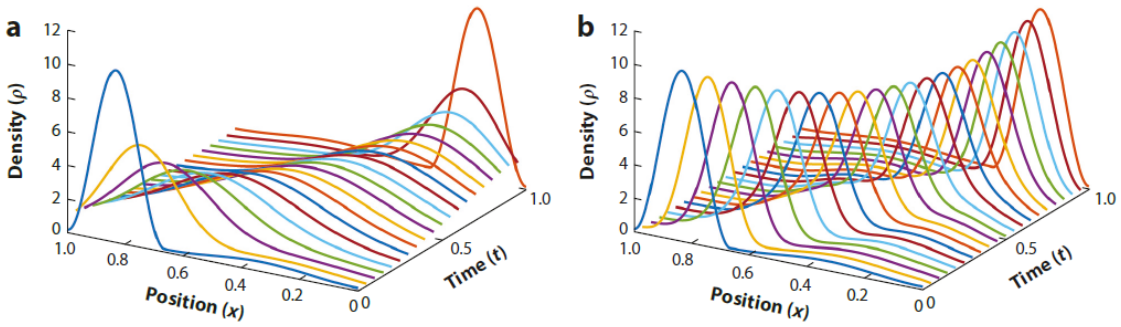



**Figure 2.1:** Optimal Transport with entropy regularisation [4], where (a) $\varepsilon = \sqrt{0.5}$, and (b) $\varepsilon = \sqrt{0.15}$.

## 2.2. Preliminaries on the Schrödinger Bridge

With the entropy regularisation term of proposition 1, the equivalence of the OT and Schrödinger Bridge (SB) problems can be explained. Clearly, when the value of $\varepsilon$ increases, the second term in the entropic OT problem dominates. In fact, the SB problem focuses on minimising this second term. Schrödinger discovered that optimising this entropy term coincides with finding a probability law $\mathcal{P}$ with marginals $\alpha$ and $\beta$, that minimises the KL-divergence.

**Definition 5.** *(KL-divergence)*
*Let $C[0,1]$ the set of continuous paths on $\mathbb{R}^n$, over time interval [0,1]. For probability laws $\mathcal{P}(\cdot)$ and $\mathcal{Q}(\cdot)$ on $C[0,1]$ , the Kullback-Leibler (KL) divergence, also called the relative entropy, is defined by:*

$$D(\mathcal{P}||\mathcal{Q}) := \int_{C[0,1]} \log\Big(\frac{\mathrm{d}\mathcal{P}}{\mathrm{d}\mathcal{Q}}\Big)\mathrm{d}\mathcal{P}.$$

For the SB problem, $\mathcal{Q}$ is the prior probability law, describing the expected way of how $\mathcal{P}$ "should" walk. Schrödinger thus reduced the entropic regulariser to a measure that finds the "most likely" path from the original to the target distribution.

Throughout this paper, the assumption is made that $\mathcal{Q}$ is continuous path on $\mathbb{R}\times\mathbb{R}$, from the ends $s = 0$ to $t = 1$ and is induced by the Brownian motion kernel:

$$q_{\sigma^2}(s,x,t,y) = \frac{1}{\sqrt{2\pi\sigma^2|t-s|}}\exp\Big(-\frac{(x-y)^2}{2\sigma^2|t-s|}\Big), \tag{2.2}$$

such that, for any measurable set $A \subset \mathbb{R}\times\mathbb{R}$:

$$\mathbb{P}[\mathcal{Q}\in A] = \int_A q_{\sigma^2}(0,x,1,y)\mathrm{d}x\mathrm{d}y = \int_A \frac{1}{\sqrt{2\pi\sigma^2}}\exp\Big(-\frac{(x-y)^2}{2\sigma^2}\Big)\mathrm{d}x\mathrm{d}y, \tag{2.3}$$

where the Brownian motion kernel $q_{\sigma^2}$ is used as the probability density function of $\mathcal{Q}$. Furthermore, $\mathcal{P}$ is also assumed to be a path at the ends $t = (0,1)$ that is absolutely continuous on $\mathbb{R}\times\mathbb{R}$. By the fact that it has marginals $\alpha$ and $\beta$, it is derived that $\pi(x,y)$ is its probability density function, and thus :

$$\mathbb{P}[\mathcal{P}\in A] = \int_A \pi(x,y)\mathrm{d}x\mathrm{d}y, \text{ with } \pi\in\mathcal{U}(\alpha,\beta). \tag{2.4}$$

With the above assumptions, the following result is obtained:

**Proposition 2.** *(Schrödinger bridge problem)*
*Let $\Pi(\alpha,\beta) := \big\{\mathcal{P}\in C[0,1] : \mathcal{P}$ is a path at the ends $t=0, t=1$, with marginals $\alpha$ and $\beta\big\}$ the set of probability laws that have the desired conditions as described above. By the assumptions of equations 2.3 and 2.4, the SB problem is then reduced to:*

$$\inf_{\mathcal{P}\in\Pi(\alpha,\beta)} D(\mathcal{P}||\mathcal{Q}) = \min_{\pi\in\mathcal{U}(\alpha,\beta)} \int_{\mathbb{R}\times\mathbb{R}} \pi(x,y)\log\left(\frac{\pi(x,y)}{q_{\sigma^2}(0,x,1,y)}\right)\mathrm{d}x\mathrm{d}y.$$

It has been proven by Fortet [7] that the SB problem has a unique optimal solution that can be found by a converging update scheme. In the next chapter, a derivation of this update scheme is given.

# 3

# Derivation of the iterative process

## 3.1. Discrete case

In the discrete case, the OT problem is also generally approximated with entropy regularisation, which is described as:

$$L_C^\epsilon(a,b) := \min_{P \in U(a,b)} \langle P, C \rangle - \epsilon H(P), \tag{3.1}$$

where

$$H(P) := -\sum_{i,j} P_{i,j}(\log(P_{i,j}) - 1) \tag{3.2}$$

is the discrete entropy of the coupling matrix $P$, as described in the book by Peyré [9].

The existence of the solution is guaranteed under the *feasibility condition*:

$$\sum_i a_i = \sum_j b_j, \tag{3.3}$$

stating that the solution to the minimisation problem exists whenever the total mass of the initial and target distribution are equal.

Cuturi [5] showed that this condition ensures the non-emptiness of $\mathcal{U}(a,b)$. Moreover, since the constraints defining $\mathcal{U}(a,b)$ are equalities, and the fact that the matrix $\mathcal{P}_{i,j}$ is strictly positive $\forall i, j$, Slater's condition [2] is satisfied. Although Slater's condition shall not be discussed in detail here, interested readers are referred to chapters 4 and 5 of the book by Boyd and Vandenberghe [2]. Together, these two conditions, with the fact that the problem is convex (in fact, it is even strictly convex, see appendix A), ensure the existence of a solution to the minimisation problem.

The uniqueness of the optimal solution comes from the strict convexity of the objective function for all positive values of $\varepsilon$ (see appendix A). In fact, the following proposition holds:

**Proposition 3.** *(Unique solution to a strictly convex problem)*
*If an optimiser $x^*$, of the minimisation problem*

$$\begin{aligned} &\min_x \ f(x) \\ &s.t. \ x \in \mathbb{R}. \end{aligned}$$

*exists, then it is unique if $f(x)$ is a strictly convex function on $\mathbb{R}$.*

*Proof.* Suppose the optimal solution is not unique. Then there exists minimisers $x^*$ and $y^* \in \mathbb{R}$ such that $f(x^*) = f(y^*)$. Since $\mathbb{R}$ is a convex set, $tx^* + (1-t)y^* \in \mathbb{R}$ for some $t \in (0,1)$. But then by the strict convexity of $f$:

$$f(tx^* + (1-t)y^*) < tf(x^*) + (1-t)f(y^*) = tf(x^*) + (1-t)f(x^*) = f(x^*), \tag{3.4}$$

which leads to a contradiction, since $x^*$ is the minimiser. So it must hold that $x^*$ is unique. □

It thus follows that $L_C^\epsilon(a,b)$ has a unique minimum on $\mathbb{R}$. A closed form of the minimiser $P$ is found by computing the Lagrangian, from which an iterative update scheme - called the Sinkhorn algorithm - is found.

**Proposition 4.** *(Discrete OT solution)*
*The optimal solution to the discrete OT problem of equation 3.1 is of the form $P_{i,j} = u_i K_{i,j} v_j$, $\forall i, j = 1, ....., n$. Here, $K_{i,j} = \exp(-\frac{C_{i,j}}{\varepsilon})$, and $u_i$ and $v_j$ are scaling vectors in $\mathbb{R}_+^n$.*

*Proof.* Recall that by definition 1 the following constraints hold:

$$P\mathbf{1}_n = a; \tag{3.5}$$
$$P^T\mathbf{1}_n = b. \tag{3.6}$$

Let $v_1, v_2 \in \mathbb{R}^n$ be dual variables for each of the constraints. Then the Lagrangian of $L_c^\varepsilon(a,b)$ is given by:

$$\mathcal{E}(P, v_1, v_2) = \langle v_1, a - P\mathbf{1}_n\rangle + \left\langle v_2, b - P^T\mathbf{1}_n\right\rangle + \langle P, C\rangle - \varepsilon H(P) \tag{3.7}$$

To find the optimal solution of $P$, the partial derivative with respect to $P_{i,j}$ must be taken. The optimal solution is then found by computing the stationary point, that is, equating the partial derivative to zero for all $i, j = 1, ..., n$. It follows that:

$$\begin{aligned} \tfrac{\partial \mathcal{E}(P,v_1,v_2)}{\partial P_{i,j}} &= -(v_1)_i - (v_2)_j + C_{i,j} - \varepsilon\left(\log(P_{i,j}) + \tfrac{P_{i,j}}{P_{i,j}} - 1\right) \\ &= -(v_1)_i - (v_2)_j + C_{i,j} - \varepsilon\log(P_{i,j}), \qquad \forall i, j = 1, ..., n. \end{aligned} \tag{3.8}$$

Equating the above to zero, it is immediately found that:

$$\begin{aligned} -(v_1)_i - (v_2)_j + C_{i,j} - \varepsilon\log(P_{i,j}) &= 0 \\ \log(P_{i,j}) &= \tfrac{-(v_1)_i + C_{i,j} - (v_2)_j}{\varepsilon} \\ P_{i,j} &= \exp\left(\tfrac{-(v_1)_i}{\varepsilon}\right)\exp\left(\tfrac{C_{i,j}}{\varepsilon}\right)\exp\left(\tfrac{-(v_2)_j}{\varepsilon}\right) \end{aligned} \tag{3.9}$$

Thus, it is found that $P_{i,j} = u_i K_{i,j} v_j$, with $K_{i,j} = \exp\left(\frac{C_{i,j}}{\varepsilon}\right)$, and scaling vectors such that $u_i = \exp\left(\frac{-(v_1)_i}{\varepsilon}\right)$ & $v_j = \left(\frac{-(v_2)_j}{\varepsilon}\right)$, as desired. □

Finding the scaling vectors $u_i$ and $v_j$ can be done by applying the Sinkhorn algorithm, which is defined by taking an initial, arbitrary guess for $v^0$, and then updating $u$ and $v$ according to the steps:

$$u^{(k+1)} := \tfrac{a}{Kv^k}, \qquad v^{(k+1)} := \tfrac{b}{Ku^{(k+1)}}. \tag{3.10}$$

These update steps shall not be further explained here, as the main focus is to find the algorithm for the continuous case. Many explanations on this algorithm have already been given, such as the one by Peyré [9]. Even computationally, there exist coded algorithms, for example in the Python OT package.

## 3.2. Continuous case

In the contiuous case, showing existence and uniqueness of the solution has been proven to be a lot more complicated than for its discrete counterpart. Robert Fortet has has proven the existence and uniqueness (up to scaling) in 1940 [7] using similar arguments as for the discrete case, but even its redux given in the article by Essid and Pavon [6] is quite extensive.

Because of the complexity of the proof in the continuous case, this section will focus on explaining the iterative steps used in Chen et al. [4] to find the solution of the above described SB problem, rather than

showing that the unique (up to scaling) solution exists. By taking a similar approach as for the discrete case, namely calculating the Lagrangian, such an explanation is given.

Recall that the OT problem is approximated with the Schrödinger bridge as in proposition 2, that is, the solution to:

$$\inf_{\mathcal{P}\in\Pi(\alpha,\beta)} D(\mathcal{P}||\mathcal{Q}) = \min_{\pi\in\mathcal{U}(\alpha,\beta)} \int_{\mathbb{R}\times\mathbb{R}} \pi(x,y)\log\left(\frac{\pi(x,y)}{q_{\sigma^2}(0,x,1,y)}\right)\mathrm{d}x\mathrm{d}y. \tag{3.11}$$

**Theorem 1.** *(SB solving algorithm)*
*Given the Markov kernel $q_{\sigma^2}(0,x,1,y)$, as well as the probability measures $\alpha(\mathrm{d}x) = \rho_0(x)\mathrm{d}x$ and $\beta(\mathrm{d}y) = \rho_1(y)\mathrm{d}y$ on $\mathbb{R}$, the unique optimal solution to the SB problem $\mathcal{P}$ is defined by:*

$$\mathcal{P}(A) = \int_A \hat{\phi}_0(x)q(0,x,1,y)\phi_1(y)\mathrm{d}x\mathrm{d}y,$$

*Where functions $\hat{\phi}_0(x)$ and $\phi_1(y)$ are found using the system:*

$$\rho_0(x) = \phi_0(x)\hat{\phi}_0(x); \tag{3.12}$$
$$\rho_1(y) = \phi_1(y)\hat{\phi}_1(y), \tag{3.13}$$

*With:*

$$\hat{\phi}_1(y) = \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y)\hat{\phi}_0(x)\mathrm{d}x; \tag{3.14}$$
$$\phi_0(x) = \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y)\phi_1(y)\mathrm{d}y. \tag{3.15}$$

*Proof.* Since $\mathcal{P}$ is a joint probability law with marginals $\alpha$ and $\beta$, at the boundaries it holds that:

$$\int_{\mathbb{R}} \pi(x,y)\mathrm{d}y = \rho_0(x) \qquad \text{at } t=0; \tag{3.16}$$
$$\int_{\mathbb{R}} \pi(x,y)\mathrm{d}x = \rho_1(y) \qquad \text{at } t=1. \tag{3.17}$$

Choosing functions $\lambda_1(x), \lambda_2(y) \in \mathbb{R}^n$ for both the constraints, the Lagrangian is then given by:

$$\begin{aligned}\mathcal{E}(\mathcal{P},\lambda_1,\lambda_2) \quad &= \int_{\mathbb{R}\times\mathbb{R}} \lambda_1(x)\left[\rho_0(x) - \int \pi(x,y)\mathrm{d}y\right]\mathrm{d}x \\ &+ \int_{\mathbb{R}\times\mathbb{R}} \lambda_2(y)\left[\rho_1(y) - \int \pi(x,y)\mathrm{d}x\right]\mathrm{d}y \\ &+ \int_{\mathbb{R}\times\mathbb{R}} \pi(x,y)\log\left(\tfrac{\pi(x,y)}{q_{\sigma^2}(0,x,1,y)}\right)\mathrm{d}x\mathrm{d}y.\end{aligned} \tag{3.18}$$

Now, the optimal solution is found by equating the derivative of the Lagrangian with respect to $\pi$ to zero, noting that the equation must hold for any function $\pi(x,y) \in \mathcal{U}(\alpha,\beta)$.

Taking the derivative with respect to $\pi(x,y)$:

$$\frac{\mathrm{d}\mathcal{E}(\mathcal{P},\lambda_1,\lambda_2)}{\mathrm{d}\pi(x,y)} = -\int_{\mathbb{R}\times\mathbb{R}} \lambda_1(x)\mathrm{d}x\mathrm{d}y - \int_{\mathbb{R}\times\mathbb{R}} \lambda_2(y)\mathrm{d}x\mathrm{d}y + \int_{\mathbb{R}\times\mathbb{R}} \log\big(\frac{\pi(x,y)}{q_{\sigma^2}(0,x,1,y)}\big)\mathrm{d}x\mathrm{d}y + \int_{\mathbb{R}\times\mathbb{R}} \frac{\pi(x,y)}{\pi(x,y)}\mathrm{d}x\mathrm{d}y. \tag{3.19}$$

Rewriting and equating to zero gives:

$$\int_{\mathbb{R}\times\mathbb{R}} \left[-\lambda_1(x) - \lambda_2(y) + \log\left(\frac{\pi^*(x,y)}{q_{\sigma^2}(0,x,1,y)}\right) + 1\right]\mathrm{d}x\mathrm{d}y = 0, \tag{3.20}$$

where $\pi^*$ is the minimiser that gives the optimal probability law $\mathcal{P}^*$. By non-negativity of the integrand ($\lambda_1(x)$ and $\lambda_2(y)$ can be any two functions in $\mathbb{R}^n$, therefore assuming the equation also holds whenever $\lambda_1(x), \lambda_2(y) \in \mathbb{R}^n_{<0}$), equation 3.20 can only hold if the integrand itself is equal to zero. Therefore:

$$\begin{aligned} -\lambda_1(x) + -\lambda_2(y) + \log\left(\tfrac{\pi^*(x,y)}{q_{\sigma^2}(0,x,1,y)}\right)+1 &= 0 \\ \log\left(\tfrac{\pi^*(x,y)}{q_{\sigma^2}(0,x,1,y)}\right) &= -1 + \lambda_1(x) + \lambda_2(y) \\ \pi^*(x,y) &= q_{\sigma^2}(0,x,1,y) * \exp(-1 + \lambda_1(x) + \lambda_2(y)) \\ &= \exp(\lambda_1(x) - \tfrac{1}{2}) q_{\sigma^2}(0,x,1,y) \exp(\lambda_2(y) - \tfrac{1}{2}) \\ &= \hat{\phi}(x) q_{\sigma^2}(0,x,1,y) \phi(y). \end{aligned} \tag{3.21}$$

So indeed, by computing the Lagrangian of $\mathcal{P}$ it is found that $\mathcal{P}^* = \int_{\mathbb{R}^n} \hat{\phi}(x) q_{\sigma^2}(0,x,1,y) \phi(y) \mathrm{d}x\mathrm{d}y$, with functions $\hat{\phi}(x) = c \exp(\lambda_1(x))$ and $\phi(y) = \frac{1}{c} \exp(\lambda_2(y))$, for some constant $c$, and some functions $\lambda_1(x)$, $\lambda_2(y)$.

Rewriting $\hat{\phi}(0,x) = \hat{\phi}_0(x)$ and $\phi(1,y) = \phi_1(y)$, by the marginal constraints as described in equations 3.16 and 3.17 it now follows that:

$$\int_{\mathbb{R}^n} \hat{\phi}_0(x) q_{\sigma^2}(0,x,1,y) \phi_1(y) \mathrm{d}y = \rho_0(x) \tag{3.22}$$

$$\int_{\mathbb{R}^n} \hat{\phi}_0(x) q_{\sigma^2}(0,x,1,y) \phi_1(y) \mathrm{d}x = \rho_1(y). \tag{3.23}$$

Since $\hat{\phi}_0(x)$ is constant w.r.t. $y$ and $\phi_1(y)$ is constant w.r.t. $x$, this gives the system:

$$\rho_0(x) = \phi_0(x) \hat{\phi}_0(x); \tag{3.24}$$

$$\rho_1(y) = \phi_1(y) \hat{\phi}_1(y), \tag{3.25}$$

With:

$$\hat{\phi}_1(y) = \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) \hat{\phi}_0(x) \mathrm{d}x; \tag{3.26}$$

$$\phi_0(x) = \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) \phi_1(y) \mathrm{d}y. \tag{3.27}$$

Equations 3.24 - 3.27 exactly describe the update steps used in the iterative algorithm for finding the solution to the SB problem, as desired. □

# 4

# Analytical solution for 1D Gaussian Marginals

## 4.1. Gaussians with zero mean

An application fto the Schrödinger bridge problem (SBP) is given in the one-dimensional Gaussian case with zero mean, called the *Bernstein case*. Here, both the marginals are normal densities and are defined on $\mathbb{R}$ as:

$$\rho_0(x) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-x^2}{2\sigma_0^2}) \tag{4.1}$$

$$\rho_1(y) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp(\frac{-y^2}{2\sigma_1^2}) \tag{4.2}$$

And transition kernel $q_{\sigma^2}(0, x, 1, y)$ is used, repeated here for readability:

$$q_{\sigma^2}(0, x, 1, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(y-x)^2}{2\sigma^2}). \tag{4.3}$$

Here, $\sigma^2$ serves a similar purpose as $\varepsilon$ in the original entropic OT problem of proposition 1, and figure 2.1. In an article by Essid et al.[6], it was presented that this example has a unique solution whenever $\sigma^2 + \sigma_0^2 - \sigma_1^2 > 0$, which holds if $\sigma_0^2 > \sigma_1^2$. If this is not the case, one can interchange the variances of $\rho_0$ and $\rho_1$ to ensure that the solution exists.

However, in Essid's article [6], there is no closed-form solution given for this example. The goal of this section is to find an analytical solution of the Schrödinger bridge problem with marginals $\rho_0$ & $\rho_1$ and transition kernel $q_{\sigma^2}$ as given by equations 4.1, 4.2 and 4.3 respectively.

For cases in higher dimensions or with more complex marginals, computations will be infeasible to solve by hand, and thus the computer algorithm based of off the update steps found in chapter 3 can be used to find a solution.

In order to find a solution for the SB problem in the one-dimensional Bernstein case, recall from theorem 1 that the optimal measure $\mathcal{P}$ is given by:

$$\mathcal{P}(\mathbb{R}^2) = \int_{\mathbb{R}^2} \hat{\phi}_0(x) q_{\sigma^2}(0, x, 1, y) \phi_1(y) \mathrm{d}x \mathrm{d}y. \tag{4.4}$$

Using the marginals and kernel as above, the following result is obtained:

**Proposition 5.** *(Analytical Gaussian case with zero mean)*
*For marginals $\rho_0$ & $\rho_1$, and transition kernel $q_{\sigma^2}$ as in equations 4.1, 4.2 and 4.3 respectively, the optimal solution to the Schrödinger Bridge problem is found by a converging algorithm that uses the update steps:*

$$a_i = \frac{\sqrt{2\pi\sigma_0^2}}{c_i\sqrt{1+\frac{\sigma^2}{d_i}}}; \qquad \frac{1}{b_i} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{d_i}}; \tag{4.5}$$

$$c_{i+1} = \frac{\sqrt{2\pi\sigma_1^2}}{a_i\sqrt{1+\frac{\sigma^2}{b_i}}}; \qquad \frac{1}{d_{i+1}} = \frac{1}{\sigma_1^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{b_i}}, \qquad \forall i \in \mathbb{N}. \tag{4.6}$$

*With the initial conditions:*

$$a_0 = 1; \qquad \frac{1}{b_0} = 0; \tag{4.7}$$

$$c_1 = \sqrt{2\pi\sigma_1^2}; \qquad \frac{1}{d_1} = \frac{1}{\sigma_1^2}, \tag{4.8}$$

*such that:*

$$\hat{\phi}_0^i(x) = \frac{1}{a_i}\exp(-\frac{x^2}{2b_i}); \qquad \phi_1^{i+1}(y) = \frac{1}{c_{i+1}}\exp(-\frac{y^2}{2d_{i+1}}), \qquad \forall i \in \mathbb{N}. \tag{4.9}$$

*Proof.* Only a sketch of the proof is given here. The fact that the algorithm converges comes from the existence of the unique solution as mentioned earlier in chapter 3, and the update steps are found by induction. This proof sketch will show the first two steps of the induction process.

**Initial expressions of $\hat{\phi}_0$ and $\phi_1$:**
To solve the SB problem, the initial guess $\hat{\phi}_0^0(x) = 1$ is taken first. It then follows from equations 3.26 and 4.3 that the first guess for $\hat{\phi}_1(y)$ is given by:

$$\begin{aligned}
\hat{\phi}_1^1(y) &= \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) * 1 \mathrm{d}x \\
&= \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) \mathrm{d}x \\
&= \int_{\mathbb{R}} \tfrac{1}{\sqrt{2\pi\sigma^2}}\exp(\tfrac{-(y-x)^2}{2\sigma^2}) \mathrm{d}x \\
&= \tfrac{1}{\sqrt{2\pi\sigma^2}} \int_{\mathbb{R}} \exp(\tfrac{-(y-x)^2}{2\sigma^2}) \mathrm{d}x \\
&= \tfrac{1}{\sqrt{2\pi\sigma^2}} * \sqrt{2\pi\sigma^2} \\
&= 1.
\end{aligned} \tag{4.10}$$

Where the integral is evaluated using change of variables multiple times. A complete calculation of this integral can be found in appendix B.1. From equations 3.25 and 4.2, it is given immediately that:

$$\phi_1^1(y) = \frac{\rho_1(y)}{\hat{\phi}_1(y)} = \frac{\rho_1(y)}{1} = \rho_1(y) = \frac{1}{\sqrt{2\pi\sigma_1^2}}\exp(\frac{-y^2}{2\sigma_1^2}). \tag{4.11}$$

Now, computing the new guess $\hat{\phi}_0^1(x)$ is done by using the update steps as in theorem 1. First $\phi_0^1(x)$ needs to be updated, which is calculated as follows:

$$\begin{aligned}
\phi_0^1(x) &= \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) * \phi_1^1(y) \mathrm{d}y \\
&= \int_{\mathbb{R}} \tfrac{1}{\sqrt{2\pi\sigma^2}}\exp(\tfrac{-(y-x)^2}{2\sigma^2}) * \tfrac{1}{\sqrt{2\pi\sigma_1^2}}\exp(\tfrac{-y^2}{2\sigma_1^2}) \mathrm{d}y \\
&= \tfrac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_1^2}} \int_{\mathbb{R}} \exp(-\tfrac{(y-x)^2}{2\sigma^2} - \tfrac{-y^2}{2\sigma_1^2}) \mathrm{d}y.
\end{aligned} \tag{4.12}$$

To solve this equation, $-\frac{(y-x)^2}{2\sigma^2} - \frac{y^2}{2\sigma_1^2}$ is rewritten as a second order polynomial $-\frac{1}{2}\frac{1}{A_0}(y - A_1)^2 + A_2$,

where $x$ is considered a constant in this equation. The substitution is found whenever:

$$A_0 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{\sigma_1^2}} \tag{4.13}$$

$$A_1 = \frac{x}{1 + \frac{\sigma^2}{\sigma_1^2}} \tag{4.14}$$

$$A_2 = \frac{1}{2}x^2\left(\frac{1}{\sigma^2 + \frac{\sigma^4}{\sigma_1^2}} - \frac{1}{\sigma^2}\right). \tag{4.15}$$

Evaluating the integral of $\exp(-\frac{1}{2}\frac{1}{A_0}(y - A_1)^2 + A_2)$ over $\mathbb{R}$ with respect to the variable $y$, as in appendix B.2, it follows that:

$$\int_{\mathbb{R}} \exp(-\frac{1}{2}\frac{1}{A_0}(y - A_1)^2 + A_2)\mathrm{d}y = \sqrt{2\pi A_0}\exp(A_2). \tag{4.16}$$

Substituting $A_0$, $A_1$ and $A_2$ as well as equation 4.16 back into equation 4.12 it is found that:

$$\phi_0^1(x) = \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_1^2}}\sqrt{\frac{2\pi}{\frac{1}{\sigma^2} + \frac{1}{\sigma_1^2}}}\exp\Big(\frac{1}{2}x^2\Big(\frac{1}{\sigma^2 + \frac{\sigma^4}{\sigma_1^2}} - \frac{1}{\sigma^2}\Big)\Big). \tag{4.17}$$

And therefore the next guess for $\hat{\phi}_0(x)$ becomes:

$$\hat{\phi}_0^1(x) = \frac{\rho_0(x)}{\phi_0^1(x)} = \frac{1}{a_1}\exp\Big(\frac{-x^2}{2b_1}\Big), \tag{4.18}$$

where:

$$a_1 = \frac{\sqrt{2\pi\sigma_0^2}\sqrt{2\pi}}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_1^2}\sqrt{\frac{1}{\sigma^2} + \frac{1}{\sigma_1^2}}} = \frac{\sqrt{2\pi\sigma_0^2}}{\sqrt{2\pi\sigma_1^2\left(1 + \frac{\sigma^2}{\sigma_1^2}\right)}}. \tag{4.19}$$

$$\frac{1}{b_1} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{\sigma_1^2}}. \tag{4.20}$$

Thus, the equations 4.5 & 4.6 as well as equation 4.9, with their corresponding initial conditions, hold for the first iteration of the algorithm.

**Second iteration of $\hat{\phi}_0$ and $\phi_1$:**
In order to find the general expressions $\hat{\phi}_0^i$ and $\phi_1^i$, a reevaluation of the iterative process is made. First a new guess for $\hat{\phi}_1$ is calculated with $\hat{\phi}_0^1$:

$$\begin{aligned}
\hat{\phi}_1^2(y) &= \int_{\mathbb{R}} q_{\sigma^2}(0, x, 1, y) * \hat{\phi}_0^1(x)\mathrm{d}x \\
&= \int_{\mathbb{R}} \tfrac{1}{\sqrt{2\pi\sigma^2}}\exp(-\tfrac{x-y}{2\sigma^2}) * \tfrac{1}{a_1}\exp(-\tfrac{x^2}{2b_1})\mathrm{d}x \\
&= \tfrac{1}{a_1\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}}\exp(-\tfrac{x-y}{2\sigma^2} - \tfrac{x^2}{2b_1})\mathrm{d}x \\
&= \tfrac{1}{a_1\sqrt{2\pi\sigma^2}}\sqrt{2\pi B_0}\exp(B_2),
\end{aligned} \tag{4.21}$$

where

$$B_0 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{b_1}} \tag{4.22}$$

$$B_2 = \frac{1}{2}y^2\Big(\frac{1}{\sigma^2 + \frac{\sigma^4}{b_1}} - \frac{1}{\sigma^2}\Big). \tag{4.23}$$

Here, $B_0$ and $B_2$ are calculated in the same manner as $A_0$ and $A_2$ are computed for equation 4.12, that is, by making the substitution $-\frac{x-y}{2\sigma^2} - \frac{x^2}{2b_1} = -\frac{1}{2}\frac{1}{B_0}(x - B_1) + B_2$.

The new guess for $\phi_1$ is then given by:

$$\phi_1^2(y) = \frac{\rho_1(y)}{\hat{\phi}_1^2(y)} = \frac{\frac{1}{\sqrt{2\pi\sigma_1^2}}\exp(\frac{-y^2}{2\sigma_1^2})}{\frac{1}{a_1\sqrt{2\pi\sigma^2}}\sqrt{2\pi B_0}\exp(B_2)} = \frac{1}{c_2}\exp(-\frac{y^2}{2d_2}), \tag{4.24}$$

with

$$c_2 = \frac{\sqrt{2\pi\sigma_1^2}}{a_1\sqrt{1+\frac{\sigma^2}{b_1}}} \tag{4.25}$$

$$\frac{1}{d_2} = \frac{1}{\sigma_1^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{b_1}}. \tag{4.26}$$

The process is repeated once more to find the next guess $\hat{\phi}_0^2$ by first computing $\phi_0^2$. The following result is obtained:

$$\begin{aligned} \phi_0^2(x) &= \int_{\mathbb{R}} q_{\sigma^2}(0, x, 1, y) * \phi_1^2(y)\mathrm{d}y \\ &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi\sigma^2}}\exp(\frac{-(y-x)^2}{2\sigma^2}) * \frac{1}{c_2}\exp(\frac{-y^2}{2d_2})\mathrm{d}y \\ &= \frac{1}{c_2\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}}\exp(-\frac{(y-x)^2}{2\sigma^2} - \frac{-y^2}{2d_2})\mathrm{d}y \\ &= \frac{1}{c_2\sqrt{2\pi\sigma^2}}\sqrt{2\pi C_0}\exp(C_2), \end{aligned} \tag{4.27}$$

with

$$C_0 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{d_2}} \tag{4.28}$$

$$C_2 = \frac{1}{2}x^2\big(\frac{1}{\sigma^2 + \frac{\sigma^4}{d_2}} - \frac{1}{\sigma^2}\big). \tag{4.29}$$

Again, the similar substitution $-\frac{(y-x)^2}{2\sigma^2} - \frac{-y^2}{2d_2} = -\frac{1}{2}\frac{1}{C_0}(y - C_1) + C_2$ is made. Then, $\hat{\phi}_0^2$ is computed as:

$$\hat{\phi}_0^2(x) = \frac{\rho_0(x)}{\phi_0^2(x)} = \frac{1}{a_2}\exp\big(\frac{-x^2}{2b_2}\big), \tag{4.30}$$

where:

$$a_2 = \frac{\sqrt{2\pi\sigma_0^2}\sqrt{2\pi}}{c_2\sqrt{2\pi\sigma^2}\sqrt{\frac{1}{\sigma^2}+\frac{1}{d_2}}} = \frac{\sqrt{2\pi\sigma_0^2}}{c_2\sqrt{1+\frac{\sigma^2}{d_2}}}; \tag{4.31}$$

$$\frac{1}{b_2} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{d_2}}. \tag{4.32}$$

Indeed, $\hat{\phi}_0^2$ and $\phi_1^2$ also satisfy equations 4.5, 4.6 and 4.9, as desired. The proof follows by induction. □

## 4.2. Gaussians with non-zero mean

In a similar application as for the one-dimensional Bernstein case of section 4.1, this section covers the solution to the SB problem by again taking one-dimensional Gaussian marginals, where instead one of the marginals now has a distribution with a non-zero mean $\mu$. This gives the application with marginals $\rho_0(x)$ and $\rho_1(y)$ defined on $\mathbb{R}$ and transition kernel $q_{\sigma^2}(0, x, 1, y)$ defined on $\mathbb{R}^2$ as:

$$\rho_0(x) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp(\frac{-(x-\mu)^2}{2\sigma_0^2}),\ \mu \neq 0; \tag{4.33}$$

$$\rho_1(y) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp(\frac{-y^2}{2\sigma_1^2}); \tag{4.34}$$

$$q_{\sigma^2}(0, x, 1, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(y-x)^2}{2\sigma^2}). \tag{4.35}$$

Note that the choice of $\rho_0$ being distributed with a non-zero mean is trivial: If $\rho_1$ were to be distributed with a mean unequal to zero instead, the roles of $\rho_0$ and $\rho_1$ can be reversed in the computations.

To find an analytical solution for the above case, a similar method is represented as for the Bernstein case of chapter 4.1. The same system of equations as presented in theorem 1 is used to find the initial guesses for $\hat{\phi}_0$ and $\phi_1$, and later general expressions $\hat{\phi}_0^i$ and $\phi_1^i$ are presented for all $i \in \mathbb{N}$. The following result is obtained:

**Proposition 6.** *(Analytical Gaussian case with non-zero mean)*
*For marginals $\rho_0$ & $\rho_1$, and transition kernel $q_{\sigma^2}$ as in equations 4.33, 4.34 and 4.35 respectively, the optimal solution to the Schrödinger Bridge problem is found by a converging algorithm that uses the update steps:*

$$a_i = \frac{\sqrt{1+\frac{\sigma^2}{e_i}}}{d_i\sqrt{2\pi\sigma_0^2}} \exp(\frac{c_i^2}{2b_i} - \frac{f_i^2}{2(e_i+\frac{e_i^2}{\sigma^2})} + \frac{f_i^2}{2e_i} - \frac{\mu^2}{2\sigma_0^2}); \tag{4.36}$$

$$\frac{1}{b_i} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2+\frac{\sigma^4}{e_i}}; \tag{4.37}$$

$$c_i = b_i(\frac{\mu}{\sigma_0^2} - \frac{f_i}{\sigma^2+e_i}); \qquad \forall i \in \mathbb{N}, \tag{4.38}$$

$$d_{i+1} = \frac{\sqrt{1+\frac{\sigma^2}{b_i}}}{a_i\sqrt{2\pi\sigma_1^2}} \exp(\frac{f_{i+1}^2}{2e_{i+1}} - \frac{c_i^2}{2(b_i+\frac{b_i^2}{\sigma^2})} + \frac{c_i^2}{2b_i}); \tag{4.39}$$

$$\frac{1}{e_{i+1}} = \frac{1}{\sigma_1^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2+\frac{\sigma^4}{b_i}}; \tag{4.40}$$

$$f_{i+1} = -e_{i+1}\frac{c_i}{\sigma^2+b_i}, \qquad \forall i \in \mathbb{N}. \tag{4.41}$$

$$\tag{4.42}$$

*With the initial conditions:*

$$a_0 = 1; \qquad d_1 = \frac{1}{\sqrt{2\pi\sigma_1^2}};$$
$$\frac{1}{b_0} = 0; \qquad \frac{1}{e_1} = \frac{1}{\sigma_1^2};$$
$$c_0 = 0; \qquad f_1 = 0.$$

*Such that:*

$$\hat{\phi}_0^i(x) = a_i \exp\big(-\frac{(x-c_i)^2}{2b_i}\big); \qquad \phi_1^{i+1}(y) = d_{i+1}\exp\big(-\frac{(y-f_{i+1})^2}{2e_{i+1}}\big), \forall i \in \mathbb{N}. \tag{4.43}$$

*Proof.* Again, for the same reasons as in section 4.1, only a sketch of the proof is given. In a similar manner as for proposition 5, the first two steps of the iterative process shall be given, from which the proof follows by induction.

**Initial expressions of $\hat{\phi}_0$ and $\phi_1$:**
As for the proof sketch of proposition 5, the initial guess $\hat{\phi}_0^0 = 1$ is used. Since $\rho_1(y)$ and $q_{\sigma^2}(0,x,1,y)$ remained unchanged with respect to their equations in section 4.1 (that is, equations 4.2 & 4.34 are the same as well as equations 4.3 & 4.35), it follows immediately that $\hat{\phi}_1^1(y) = 1$, $\phi_1^1(y) = \frac{1}{\sqrt{2\pi\sigma_1^2}}\exp(\frac{-y^2}{2\sigma_1^2})$. and consequently $\phi_0^1(x) = \frac{1}{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_1^2}}\sqrt{\frac{2\pi}{\frac{1}{\sigma^2}+\frac{1}{\sigma_1^2}}}\exp\big(\frac{1}{2}x^2(\frac{1}{\sigma^2+\frac{\sigma^4}{\sigma_1^2}} - \frac{1}{\sigma^2})\big)$.

However, since equation 4.33 is not equal to equation 4.1, the computation of $\hat{\phi}_0^1(x)$ is slightly different than in the proof of proposition 5, which will result in expressions that alter significantly to those of equations 4.5 - 4.9.

$\hat{\phi}_0^1(x)$ is computed as follows:

$$\hat{\phi}_0^1(x) = \frac{\rho_0(x)}{\phi_0^1(x)} = \frac{\sqrt{2\pi\sigma^2}\sqrt{2\pi\sigma_1^2}}{\sqrt{2\pi\sigma_0^2}\sqrt{\frac{2\pi}{\frac{1}{\sigma^2}+\frac{1}{\sigma_1^2}}}}\frac{\exp\big(-\frac{(x-\mu)^2}{2\sigma_0^2}\big)}{\exp\big(\frac{1}{2}x^2(\frac{1}{\sigma^2+\frac{\sigma^4}{\sigma_1^2}} - \frac{1}{\sigma^2})\big)} = a_1\exp\big(-\frac{(x-c_1)^2}{2b_1}\big), \tag{4.44}$$

where:

$$a_1 = \frac{\sqrt{\sigma^2+\sigma_1^2}}{\sigma_0}\exp\big(\frac{c_1^2}{2b_1} - \frac{\mu^2}{2\sigma_0^2}\big); \tag{4.45}$$
$$\frac{1}{b_1} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2+\frac{\sigma^4}{\sigma_1^2}}; \tag{4.46}$$
$$c_1 = b_1\frac{\mu}{\sigma_0^2}. \tag{4.47}$$

Thus, $\hat{\phi}_0^1$ and $\phi_1^1$ both satisfy equations 4.36 - 4.43 with their corresponding initial conditions, and so the proposition holds for the first iteration.

**Second iteration of $\hat{\phi}_0$ and $\phi_1$:**
The second iteration for finding the solution is done analogously. First $\hat{\phi}_1^2(y)$ is computed:

$$\begin{aligned}\hat{\phi}_1^2(y) &= \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y) * \hat{\phi}_0^1(x)\mathrm{d}x \\ &= \frac{a_1}{\sqrt{2\pi\sigma^2}}\int_{\mathbb{R}}\exp\big(-\frac{(x-y)^2}{2\sigma^2} - \frac{(x-c_1)^2}{2b_1}\big)\mathrm{d}x \\ &= \frac{a_1}{\sqrt{2\pi\sigma^2}}\sqrt{2\pi R_0}\exp(R_2),\end{aligned} \tag{4.48}$$

with:

$$R_0 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{b_1}}; \tag{4.49}$$

$$R_1 = R_0(\frac{y}{\sigma^2} + \frac{c_1}{b_1}); \tag{4.50}$$

$$R_2 = \frac{R_1^2}{2R_0} - \frac{y^2}{2\sigma^2} - \frac{c_1^2}{2b_1} \tag{4.51}$$

which are found by making the substitution $-\frac{(x-y)^2}{2\sigma^2} - \frac{(x-c_1)^2}{2b_1} = -\frac{(x-R_1)}{2R_0} + R_2$. With $\hat{\phi}_1^2(y)$, $\phi_1^2(y)$ is computed as follows:

$$\phi_1^2(y) = \frac{\rho_1(y)}{\hat{\phi}_1^2(y)} = \frac{\sqrt{2\pi\sigma^2}}{a_1\sqrt{2\pi\sigma_1^2}\sqrt{2\pi R_0}} \exp\big(-\frac{y^2}{2\sigma_1^2} - R_2\big) = d_2 \exp\big(-\frac{(y-f_2)^2}{2e_2}\big), \tag{4.52}$$

where:

$$d_2 = \frac{\sqrt{1+\frac{\sigma^2}{b_1}}}{a_1\sqrt{2\pi\sigma_1^2}} \exp(\frac{f_2^2}{2e_2} - \frac{c_1^2}{2(b_1 + \frac{b_1^2}{\sigma^2})} + \frac{c_1^2}{2b_1}); \tag{4.53}$$

$$\frac{1}{e_2} = \frac{1}{\sigma_1^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2 + \frac{\sigma^4}{b_1}}; \tag{4.54}$$

$$f_2 = -e_2\frac{c_1}{\sigma^2 + b_1}. \tag{4.55}$$

Now $\hat{\phi}_0^2(x)$ is computed by first evaluating $\phi_0^2(x)$ , in a similar manner as for $\hat{\phi}_1^2(y)$:

$$\begin{aligned} \phi_0^2(x) &= \int_{\mathbb{R}} q_{\sigma^2}(0,x,1,y)\phi_1^2(y)\mathrm{d}y \\ &= \frac{d_2}{\sqrt{2\pi\sigma^2}} \int_{\mathbb{R}} \exp\big(-\frac{(x-y)^2}{2\sigma^2} - \frac{(y-f_2)^2}{2e_2}\big)\mathrm{d}y \\ &= \frac{d_2}{\sqrt{2\pi\sigma^2}}\sqrt{2\pi S_0}\exp(S_2). \end{aligned} \tag{4.56}$$

By making the substitution $-\frac{(x-y)^2}{2\sigma^2} - \frac{(y-f_2)^2}{2e_2} = -\frac{1}{2S_0}(y-S_1)^2 + S_2$,
$S_0$, $S_1$ and $S_2$ are given by:

$$S_0 = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{e_2}}; \tag{4.57}$$

$$S_1 = S_0(\frac{x}{\sigma^2} + \frac{f_2}{e_2}); \tag{4.58}$$

$$S_2 = \frac{S_1^2}{2S_0} - \frac{x^2}{2\sigma^2} - \frac{f_2^2}{2e_2}. \tag{4.59}$$

Then $\hat{\phi}_0^2(x)$ is evaluated by:

$$\hat{\phi}_0^2(x) = \frac{\rho_0(x)}{\phi_0^2(x)} = \frac{\sqrt{2\pi\sigma^2}}{d_2\sqrt{2\pi\sigma_0^2}\sqrt{2\pi S_0}} \exp\big(-\frac{(x-\mu)^2}{2\sigma_0^2} - S_2\big) = a_2 \exp\big(-\frac{(x-c_2)^2}{2b_2}\big), \tag{4.60}$$

with:

$$a_2 = \frac{\sqrt{1+\frac{\sigma^2}{e_2}}}{d_2\sqrt{2\pi\sigma_0^2}}\exp\left(\frac{c_2^2}{2b_2} - \frac{f_2^2}{2(e_2+\frac{e_2^2}{\sigma^2})} + \frac{f_2^2}{2e_2} - \frac{\mu^2}{2\sigma_0^2}\right); \tag{4.61}$$

$$\frac{1}{b_2} = \frac{1}{\sigma_0^2} - \frac{1}{\sigma^2} + \frac{1}{\sigma^2+\frac{\sigma^4}{e_2}}; \tag{4.62}$$

$$c_2 = b_2\left(\frac{\mu}{\sigma_0^2} - \frac{f_2}{\sigma^2+e_2}\right). \tag{4.63}$$

Therefore, it follows that the proposition also holds for the second iteration, as the equations 4.36 - 4.43 are again satisfied. The proof of proposition 6 follows by induction. □

# 5

# Comparison of computational algorithms

## 5.1. Methodology of computational algorithms

For the algorithms of theorem 1 and propositions 5 & 6, computational algorithms in the form of Python programmes have been made. The full codes can be found in appendices C.1, D and E respectively. The goal of this chapter is to compare the performance of the general algorithm to the algorithms for both the Gaussian cases.

Before doing so, one remark must be made: for the update steps in theorem 1 as well as the computation of the KL-divergence improper integrals are used, which cannot be directly calculated computationally. Therefore, they must be approximated using a numerical method. Since the mean and variances of the marginals can be chosen, and the grid can be altered accordingly, the improper integrals can be evaluated as finite ones. This can be done by adjusting the grid so that function values will remain zero outside the grid. Doing this is possible, as it is known by the existence of the solution that the integrals do not diverge.

Still, the finite integrals must be approximated, as the programme cannot evaluate these exactly. An appropriate numerical method to do so is by using the trapezoidal rule, as explained by Vuik et al. [11]. This rule, described in proposition 7,is applied in the computational algorithms of appendices C, D and E.

**Proposition 7.** *(Trapezoidal rule)*
*The integral of a function $f(x) \in C[a,b]$ can be approximated with a linear interpolation polynomial $L_1(x)$, defined by:*

$$L_1(x) := \frac{b-x}{b-a} f(a) + \frac{x-a}{b-a} f(b).$$

*The integral is then approximated by:*

$$\int_a^b f(x)\mathrm{d}x \approx \int_a^b L_1(x)\mathrm{d}x = \frac{b-a}{2}(f(a)+f(b)).$$

*If $f(x) \in C[a,b]$ is evaluated over a grid of $n+1$ points $x_0, \ldots\ldots, x_n$, with $x_0 = a, x_n = b$, then the integral of $f$ can be approximated by the sum of approximate integrals for each interval $[x_{i-1}, x_i]$, that is:*

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} L_1(x) = \sum_{i=1}^{n} \frac{x_i - x_{i-1}}{2}(f(x_{i-1}) + f(x_i)).$$

## 5.2. Result comparison

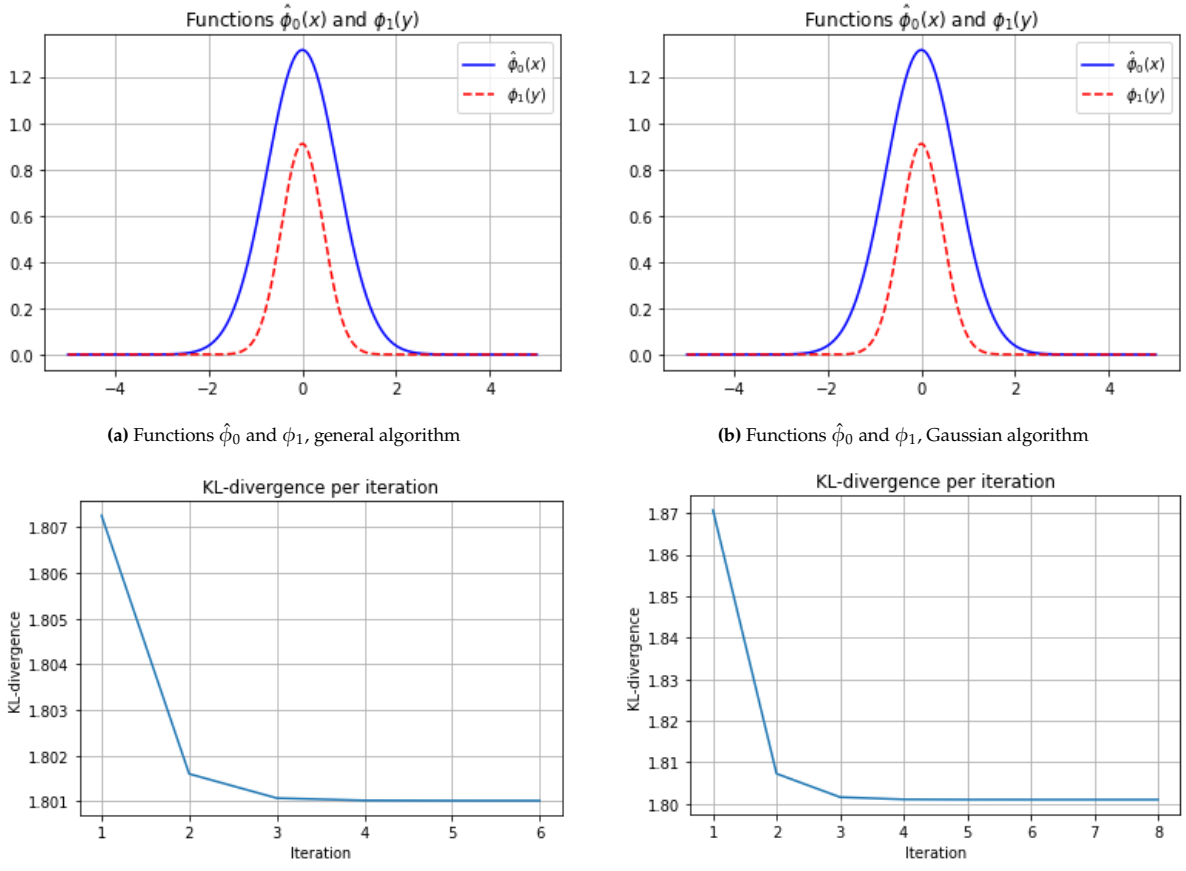
In this section, under the assumptions that the marginals are distributed normally, that is $\rho_0 \sim \mathcal{N}(\mu, \sigma_0^2)$, $\rho_1 \sim \mathcal{N}(0, \sigma_1^2)$, and transition kernel $q_{\sigma^2}$ as in equation 4.35, the results of the computational algorithms shall be compared.

In total, three computations are discussed in this chapter: in all cases, $\sigma_0^2 = 0.5^2$ and $\sigma_1^2 = 0.4^2$ are used. The values of $\mu$ and $\sigma^2$ shall vary over the computations.

**Computation 1:** $\mu = 0$ **,** $\sigma^2 = 0.25$

Taking zero means and a noise variance $\sigma^2 = 0.25$, stable results are obtained, where functions $\hat{\phi}_0$ and $\phi_1$ are neatly scaled Gaussians. It can be seen that both algorithms converge fast: the general algorithm of theorem 1 converges in 6 iterations, and the algorithm for zero-mean Gaussians of proposition 5 does so in 8 iterations. Here convergence is established whenever the relative error between the last two iterations is smaller than a tolerance $1E-06$, for all of the update steps.

It can be seen that the algorithms produce very similar results for the chosen parameters: the plots in subfigures 5.1a and 5.1b are near identical, and the KL-divergence converges to a minimum in both cases, with extremely similar values, as can be seen in subfigures 5.1c and 5.1d, as well as in table 5.1.




**(a)** Functions $\hat{\phi}_0$ and $\phi_1$, general algorithm

**(b)** Functions $\hat{\phi}_0$ and $\phi_1$, Gaussian algorithm

**(c)** KL-divergence per iteration, general algorithm

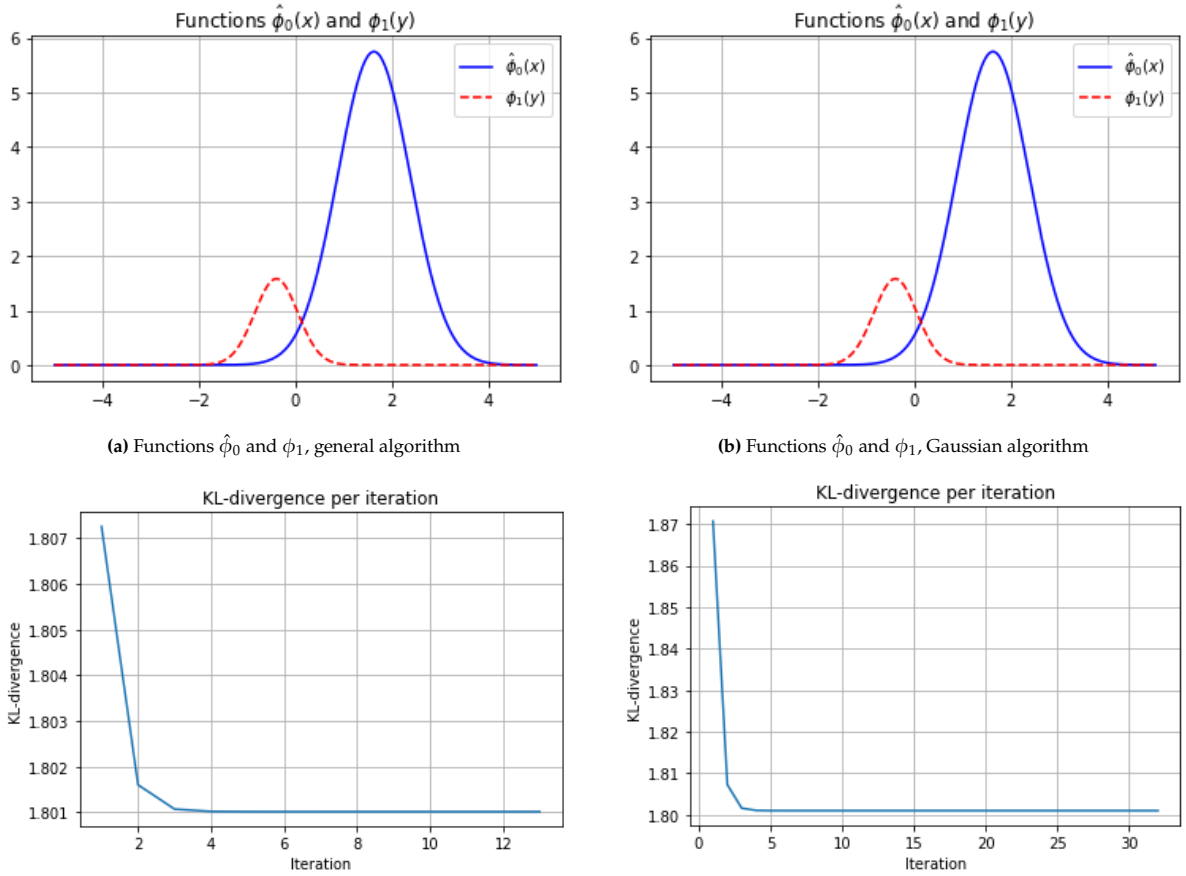
**(d)** KL-divergence per iteration, Gaussian algorithm

**Figure 5.1:** Results of both algorithms for $\mu = 0$, $\sigma_0^2 = 0.25$, $\sigma_1^2 = 0.16$, and $\sigma^2 = 0.25$

**Computation 2:** $\mu = 0.5$ , $\sigma^2 = 0.25$
If the value of $\sigma^2$ is again 0.25 , but the value of the mean is shifted to $\mu = 0.5$, the algorithm for the Gaussian marginals with non-zero means as described in proposition 6 can be used and compared to the general algorithm.

For these values of $\mu$ and $\sigma^2$, convergence is established after 13 and 32 iterations, for the general and Gaussian algorithm, respectively. Again, both algorithms produce extremely similar results for the plots of $\hat{\phi}_0$ & $\phi_1$, and for the value of the KL-divergence, which can be seen in figure 5.2 and table 5.1.

For this computation, results seem to be stable, as $\hat{\phi}_0$ & $\phi_1$ are nicely scaled Gaussian functions and the KL-divergence is minimised over the iterations. One interesting result is that the value of the KL-divergence is nearly equal to that of computation 1. From this result one could conclude that the value of the mean has a minimal impact on the value of the KL-divergence, even though functions $\hat{\phi}_0$ and $\phi_1$ alter in both computations.




**(a)** Functions $\hat{\phi}_0$ and $\phi_1$, general algorithm

**(b)** Functions $\hat{\phi}_0$ and $\phi_1$, Gaussian algorithm

**(c)** KL-divergence per iteration, general algorithm

**(d)** KL-divergence per iteration, Gaussian algorithm

**Figure 5.2:** Results of both algorithms for $\mu = 0.5$, $\sigma_0^2 = 0.25$, $\sigma_1^2 = 0.16$, and $\sigma^2 = 0.25$

| computation | 1 | 2 |
|---|---|---|
| General algorithm | 1,80101307771638 | 1,80101303126115 |
| Gaussian algorithm | 1,80101303564490 | 1,80101303126114 |
| Error | 4,207E-08 | 9,992E-15 |

**Table 5.1:** Values of the KL-divergence in the final iteration of computations 1 and 2

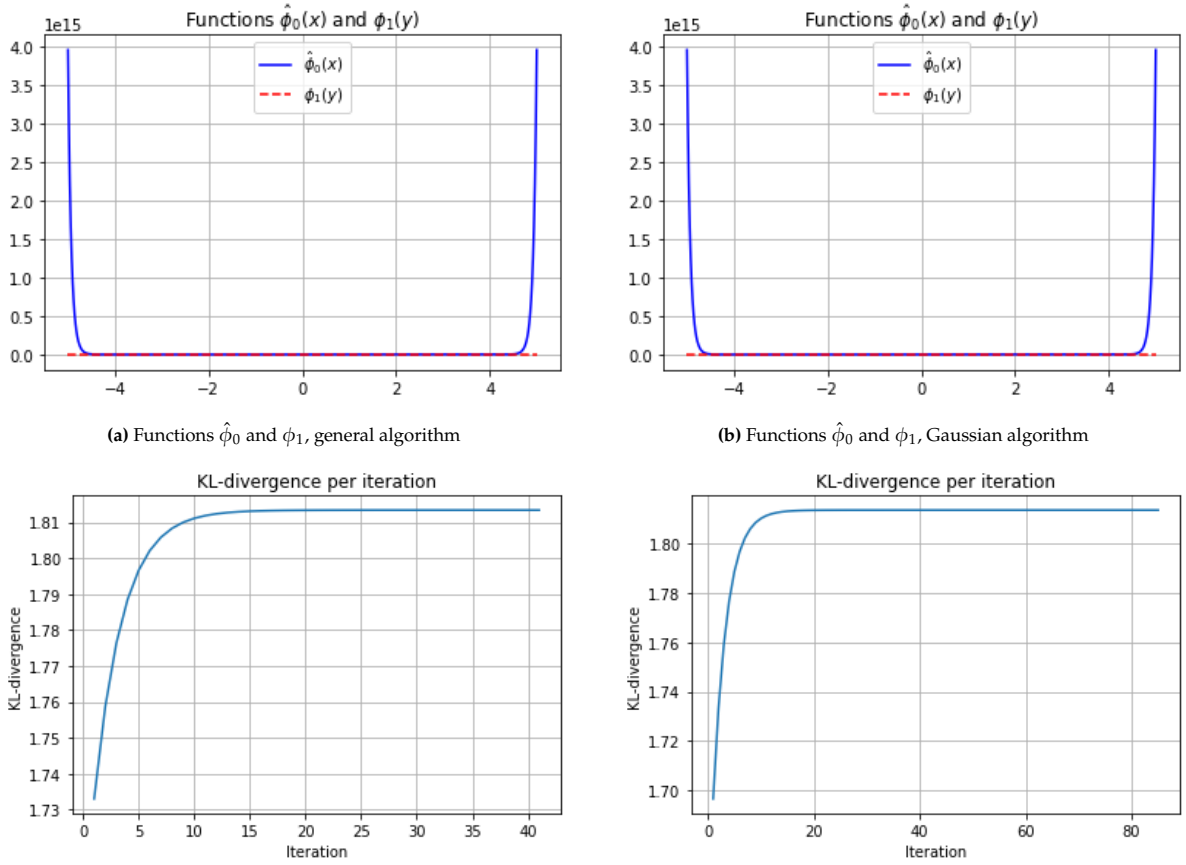
## 5.3. Discussion

**Computation 3:** $\mu = 0$ **,** $\sigma^2 = 0.04$

Going back to having a zero mean, but altering the value of $\sigma^2$ to $0.2^2 = 0.04$, the results vary significantly to those of computation 1. The small value of $\sigma^2$, which behaves similarly to the entropic term $\varepsilon$, causes unstable results: The functions $\hat{\phi}_0$ and $\phi_1$ explode, and the KL-divergence does not converge to a minimum, as can be seen in figure 5.3.

An interesting result is that the function values still converge over the iterations for both algorithms, although slower than in computation 1. The general algorithm converges numerically after 41 iterations, and the Gaussian zero mean algorithm does so after 85 iterations. Both algorithms produce near-identical results for this rather unstable solution, both for the functions $\hat{\phi}_0$ and $\phi_1$ as for the KL-divergence.

It is, however, possible to derive the analytical closed form equation of the KL-divergence for Gaussians. This could be implemented in the computer algorithm, in an attempt to find results that are less affected by the numerical instability.




**(a)** Functions $\hat{\phi}_0$ and $\phi_1$, general algorithm

**(b)** Functions $\hat{\phi}_0$ and $\phi_1$, Gaussian algorithm

**(c)** KL-divergence per iteration, general algorithm

**(d)** KL-divergence per iteration, Gaussian algorithm

**Figure 5.3:** Results of both algorithms for $\mu = 0$, $\sigma_0^2 = 0.25$, $\sigma_1^2 = 0.16$, and $\sigma^2 = 0.04$

Overall, it can be seen that both algorithms produce very similar results for all shown computations. The general algorithm converges faster in each of the computations done in this section, which could potentially make it preferable to the Gaussian algorithms.

# 6

# Visualisation of the Schrödinger Bridge

## 6.1. Methodology of bridge visualisation

With the computational algorithms introduced in chapter 5, the Schrödinger Bridge can now be visualised in a similar manner as is done for figure 2.1. By using the function $\pi$ that is obtained through these algorithms, as well as the Brownian motion kernel $q_{\sigma^2}$, the intermediate distributions $\rho_\tau$ can be computed for times $\tau \in (0,1)$. These distributions demonstrate the evolution from initial distribution $\rho_0$ to target distribution $\rho_1$.

As both algorithms in chapter 5 produce extremely similar functions $\hat{\phi}_0$ and $\phi_1$, and therefore near-identical functions $\pi$, the bridge shall only be computed for the general algorithm - the algorithm made for the Gaussian cases would produce much the same figures.

In order to compute distributions $\rho_\tau$, one must shift their focus to finding the optimal 'tripling', instead of the coupling from definition 3. This means that not only the locations $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ are considered, but also an intermediate location, which is denoted $z_\tau \in \mathcal{Z}$. Here $\mathcal{Z}$ is a random variable of all possible intermediate locations. According to Benamou [1], the goal is to find an expression for $\pi(x, z_\tau, y)$, from which $\rho_t$ can be derived by integration.

**Definition 6.** *(Tripling set)*
*For probability density functions $(\alpha, \beta, \gamma)$ on $\mathbb{R}$ of random variables $(\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3)$, the set $\mathcal{V}$ of joint probabilities $\pi_3$ with marginals $\alpha$, $\beta$ and $\gamma$ is called a tripling set, and is defined by:*

$$\mathcal{V}(\alpha,\beta,\gamma) := \left\{ \pi_3(x_1,x_2,x_3) : \mathbb{R}^3 \to \mathbb{R}^3 : \int_{\mathbb{R}\times\mathbb{R}} \pi_3 \mathrm{d}x_2\mathrm{d}x_3 = \alpha, \int_{\mathbb{R}\times\mathbb{R}} \pi_3 \mathrm{d}x_1\mathrm{d}x_3 = \beta, \int_{\mathbb{R}\times\mathbb{R}} \pi_3 \mathrm{d}x_1\mathrm{d}x_2 = \gamma \right\}.$$

*If $\pi_3 \in \mathcal{V}(\alpha,\beta,\gamma)$, then $\pi_3$ is called a tripling of $\alpha$, $\beta$ and $\gamma$.*

The tripling $\pi(x, z_\tau, y) \in \mathcal{V}(\rho_0, \rho_\tau, \rho_1)$, as defined in definition 6, is used to obtain the following result:

**Proposition 8.** *(Intermediate densities)*
*For Schrödinger Bridges with marginals $\rho_0$ and $\rho_1$, optimal probability density function $\pi(x,y)$, and Brownian motion kernel $q_{\sigma^2}(s,x,t,y)$, intermediate distributions $\rho_\tau$ are given by:*

$$\rho_\tau(z_\tau) = \int_{\mathbb{R}\times\mathbb{R}} \frac{q_{\sigma^2}(0,x,\tau,z_\tau) q_{\sigma^2}(\tau,z_\tau,1,y)}{q_{\sigma^2}(0,x,1,y)} \pi(x,y)\mathrm{d}x\mathrm{d}y, \qquad \tau \in (0,1). \tag{6.1}$$

*Proof.* Note that by using conditional probabilities:

$$\pi(x, z_\tau, y) = \pi(z_\tau | x, y)\pi(x, y). \tag{6.2}$$

For $\tau \in (0.1)$, the optimal transport of the intermediate distributions, given the initial and target distributions, is induced by Brownian motion and thus:

$$\pi(z_\tau|x,y) = q_{\sigma^2}(z_\tau|x,y) = \frac{q_{\sigma^2}(x,z_\tau,y)}{q_{\sigma^2}(x,y)}, \tag{6.3}$$

using the short-form notation $q_{\sigma^2}(x,y) = q_{\sigma^2}(0,x,1,y)$.

Therefore, by definition 6:

$$\rho_\tau(z_\tau) = \int_{\mathbb{R}\times\mathbb{R}} \frac{q_{\sigma^2}(x,z_\tau,y)}{q_{\sigma^2}(x,y)}\pi(x,y)\mathrm{d}x\mathrm{d}y. \tag{6.4}$$

Now, $q_{\sigma^2}((x,z_\tau,y)$ is the Brownian motion kernel describing the movement from $x$ to $y$, by first moving from $x$ to the intermediate stop $z_\tau$ in a time $\tau$, and then moving from $z_\tau$ to y using the remaining time $1-\tau$. By the Markov property of the Brownian motion, these increments are independent and thus it immediately follows that:

$$q_{\sigma^2}(z_\tau,x,y) = q_{\sigma^2}(0,x,\tau,z_\tau)q_{\sigma^2}(\tau,z_\tau,1,y), \tag{6.5}$$

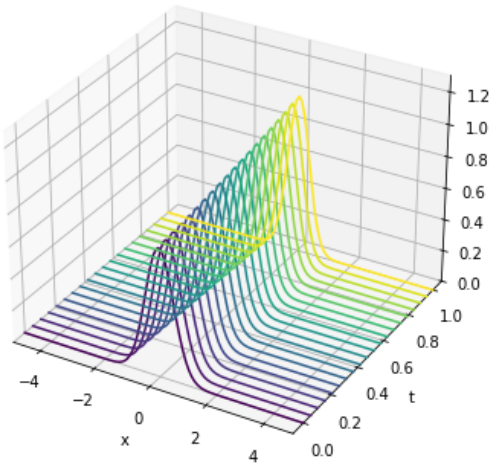
from which the result of equation 6.1 is obtained. □

## 6.2. Bridges for Gaussian marginals

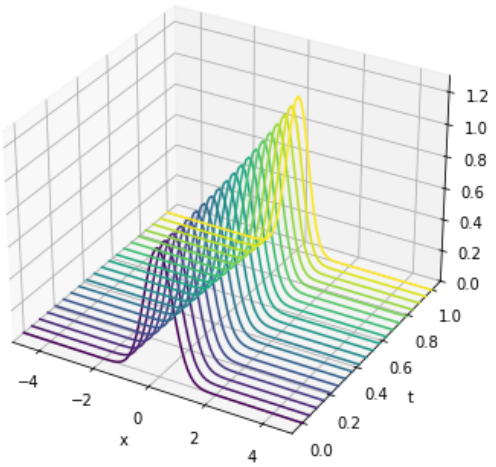
With the result of proposition 8, the Schrödinger Bridges of computations 1,2 and 3 were visualised using an extension to the general algorithm, for which the Python code can be found in appendix C.2. A time grid was made to find the intermediate distributions for twenty equally spaced values of $\tau \in (0, 1)$, that is, $\tau_i = \frac{i}{19}$, $i = 0, ....., 19$.

In figures 6.1 and 6.2, these visualisations are presented. Here, the same initial and target distributions $\rho_0$ and $\rho_1$ as in chapter 5 are used, with varying values of $\sigma$ of the kernel $q_{\sigma^2}$ used in each subfigure. in particular, subfigures 6.1a and 6.1b represent the bridges of computations 3 and 1 respectively, and the bridge of computation 2 is visualised in subfigure 6.2b. Note that figures 6.1 and 6.2 look alike, but that the mean of $\rho_0$ is shifted in figure 6.2 to a value $\mu = 0.5$, whereas in figure 6.1, $\rho_0$ has a zero mean.

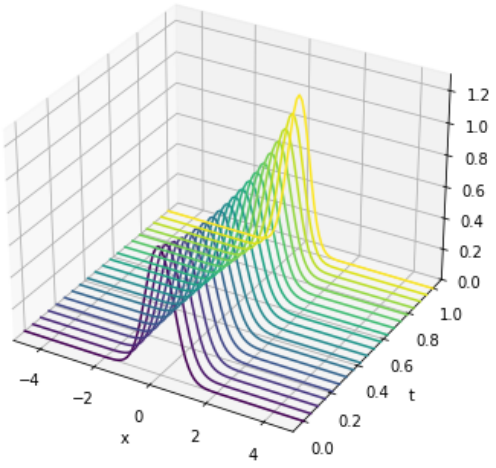
It can be seen that for both figures 6.1 and 6.2, increasing the value of $\sigma$ creates more dampening in the bridge, and, as seen in chapter 5, uses a higher rate of approximation. The general effect of $\sigma$ on computational results shall be discussed further in section 6.4.



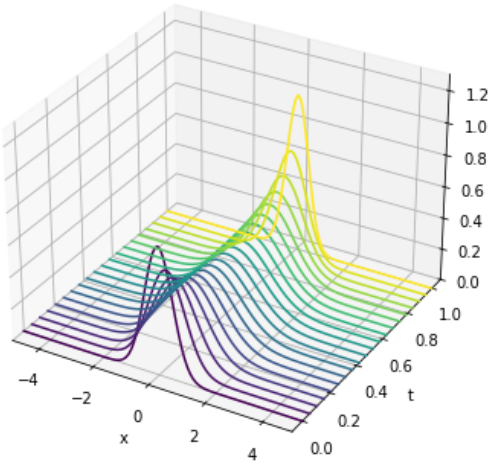

**(a)** $\sigma = 0.2$




**(b)** $\sigma = 0.5$

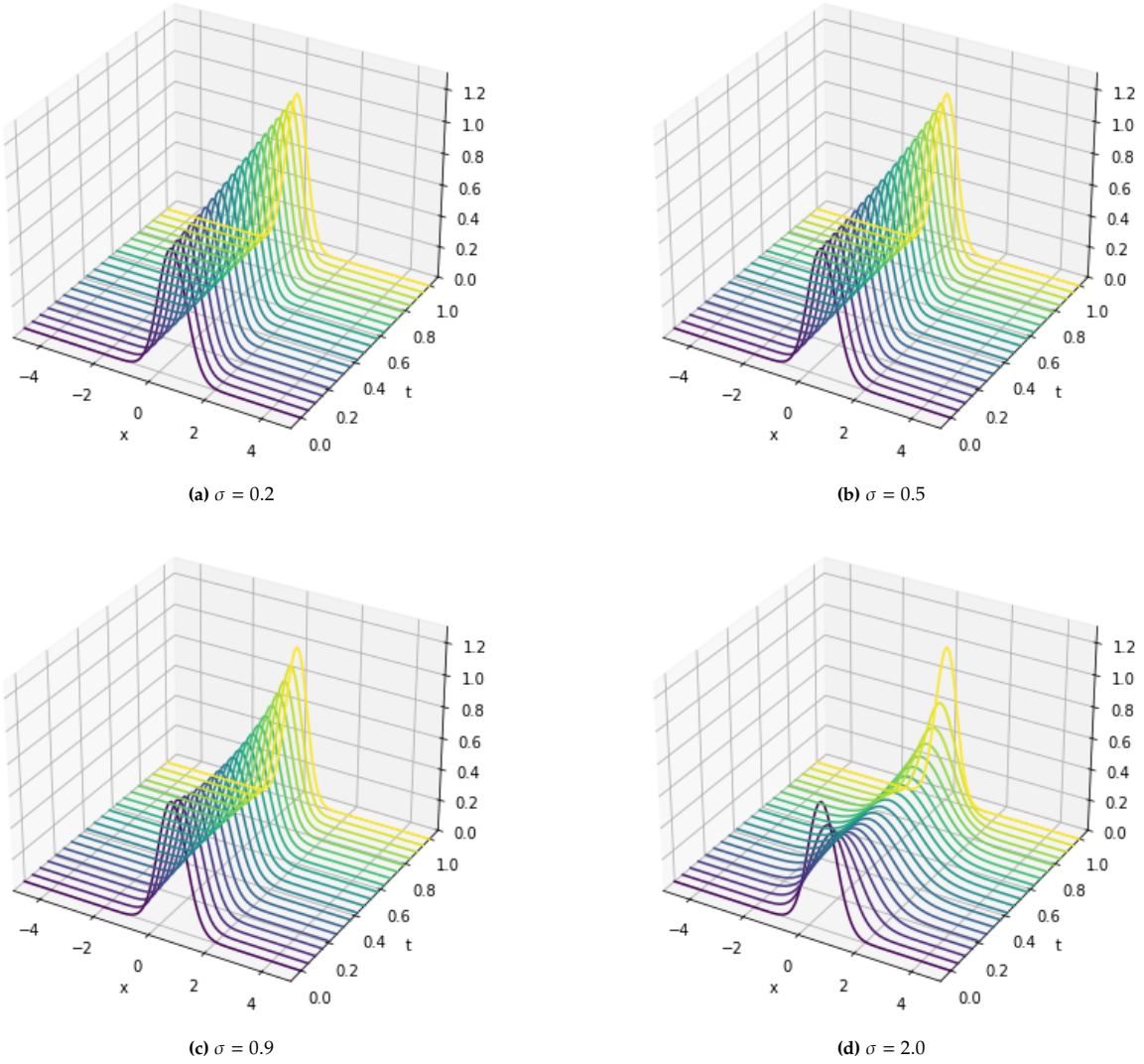



**(c)** $\sigma = 0.9$




**(d)** $\sigma = 2.0$

**Figure 6.1:** Schrödinger bridges for $\rho_0 \sim \mathcal{N}(0, 0.5^2)$, $\rho_1 \sim \mathcal{N}(0, 0.4^2)$, using Brownian motion kernel $q_{\sigma^2}$ with varying values of $\sigma$.




**(a)** $\sigma = 0.2$

**(b)** $\sigma = 0.5$

**(c)** $\sigma = 0.9$

**(d)** $\sigma = 2.0$

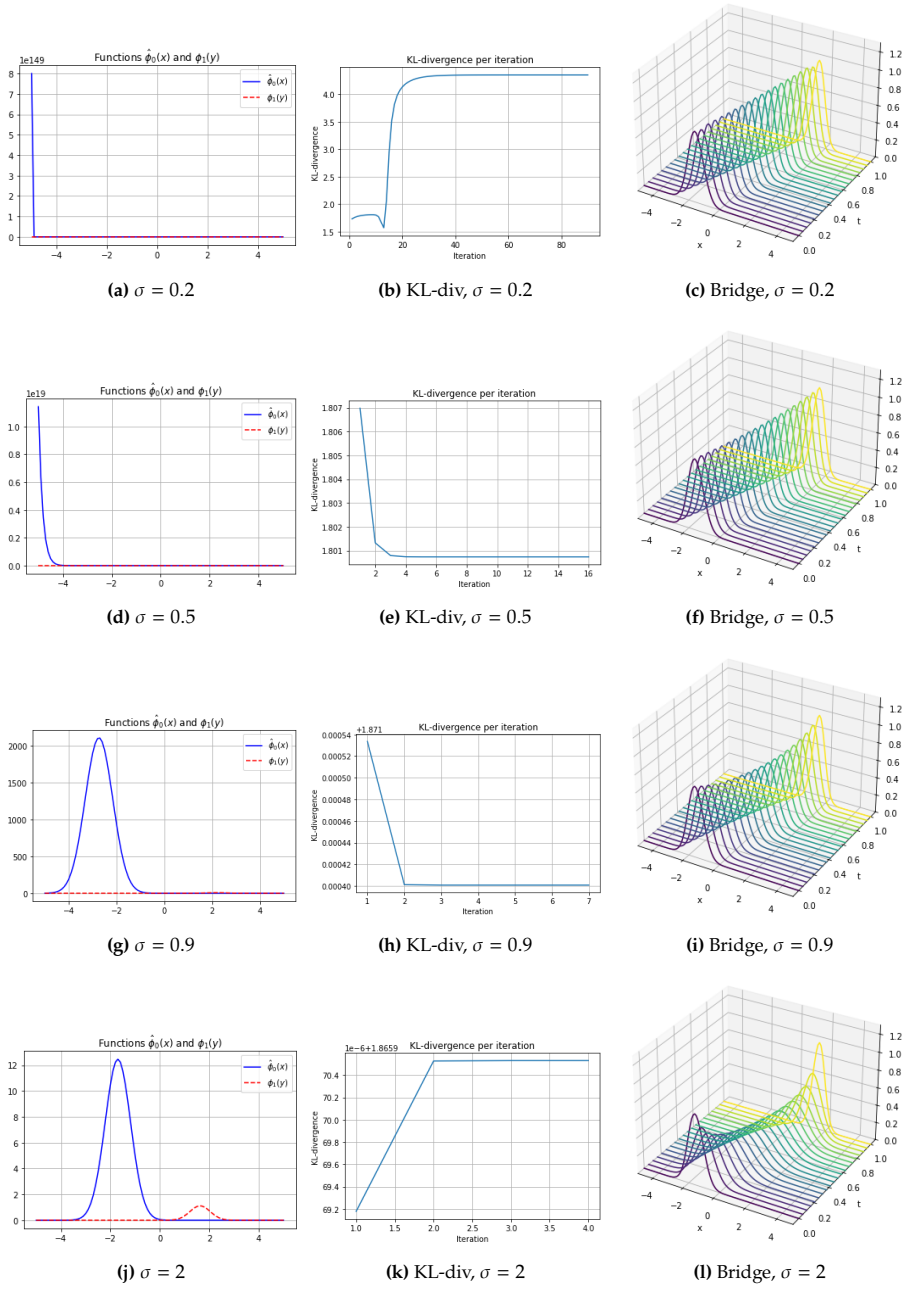
**Figure 6.2:** Schrödinger bridges for $\rho_0 \sim \mathcal{N}(0.5, 0.5^2)$, $\rho_1 \sim \mathcal{N}(0, 0.4^2)$, using Brownian motion kernel $q_{\sigma^2}$ with varying values of $\sigma$.

Interestingly enough, for the value $\sigma^2 = 0.2^2$, still a neat Bridge is plotted, both for $\mu = 0$ and $\mu - 0.5$. Despite the fact that the algorithm fails in minimising the KL-divergence for such a small value of $\sigma^2$, it seems to still find a nicely scaled optimal function $\pi(x, y)$, from which the Schrödinger Bridge can be computed.

## 6.3. Bridges for various marginals

Naturally, the general algorithm of theorem 1 with its code from appendix C can produce Schrödinger Bridges that are not only (zero mean) Gaussian. In this section, four more computations are presented, which are example computations that the Gaussian algorithms of propositions 5 and 6 are unable to create.

**Computation 4 (Two shifted Gaussians)**
The first of these computations is one that still uses two Gaussian marginals, with the difference that now both marginals have a shifted, non-zero mean. Using $\rho_0 \sim \mathcal{N}(-1.5, 0.5^2)$, $\rho_1 \sim \mathcal{N}(1.5, 0.4^2)$, results are presented in figure 6.3.




**(a)** $\sigma = 0.2$ **(b)** KL-div, $\sigma = 0.2$ **(c)** Bridge, $\sigma = 0.2$

**(d)** $\sigma = 0.5$ **(e)** KL-div, $\sigma = 0.5$ **(f)** Bridge, $\sigma = 0.5$

**(g)** $\sigma = 0.9$ **(h)** KL-div, $\sigma = 0.9$ **(i)** Bridge, $\sigma = 0.9$

**(j)** $\sigma = 2$ **(k)** KL-div, $\sigma = 2$ **(l)** Bridge, $\sigma = 2$

**Figure 6.3:** Computations for $\rho_0 \sim \mathcal{N}(-1.5, 0.5^2)$, $\rho_1 \sim \mathcal{N}(1.5, 0.4^2)$, using kernel $q_{\sigma^2}$ with varying values of $\sigma$.

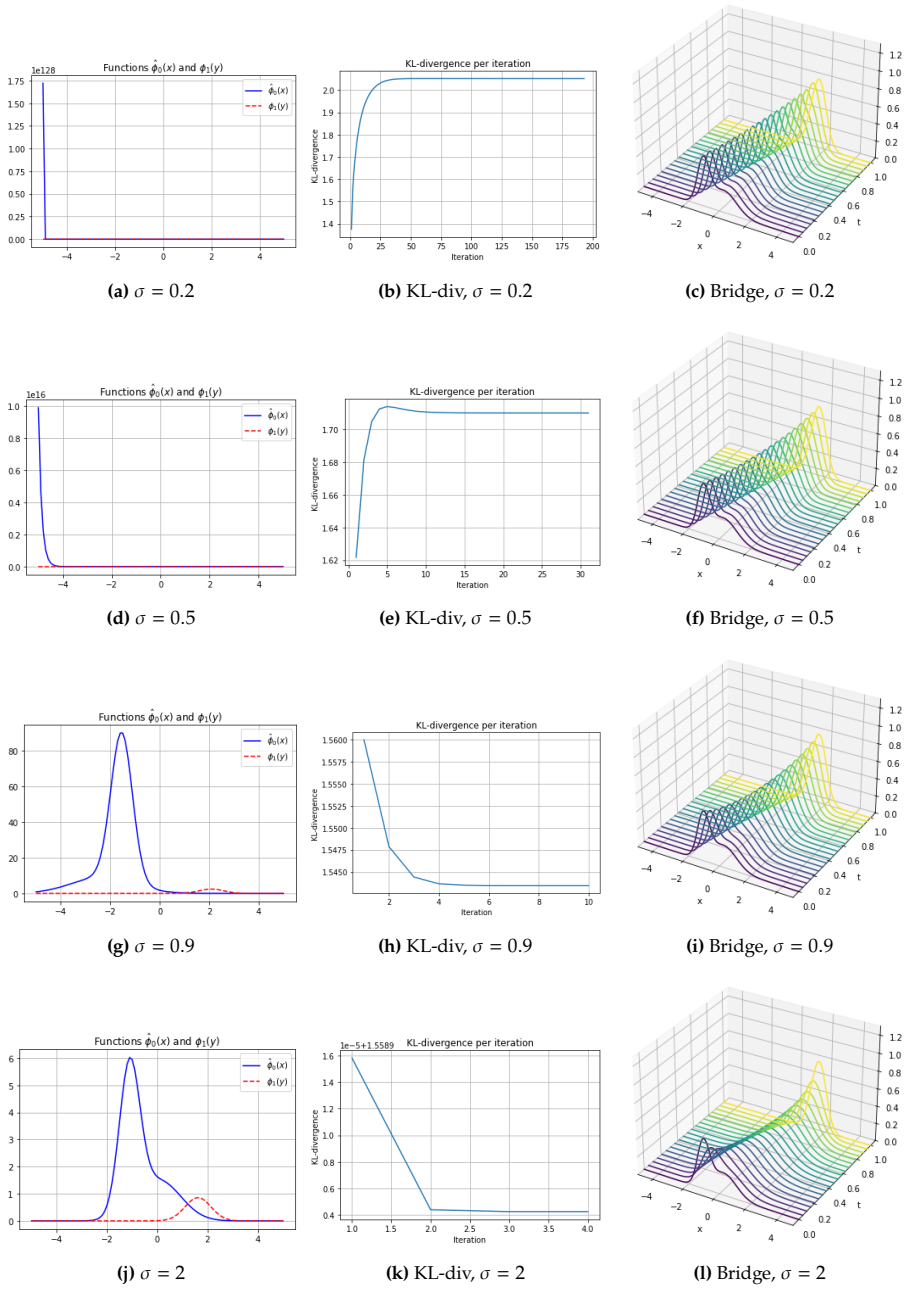
It is clear that these marginals show similar behaviour as the before seen marginals of the first three

computations, where KL-divergence does not decrease over the iterations whenever $\sigma^2$ is too close to zero, and functions $\hat{\phi}_0$ and $\phi_1$ are oddly scaled for these small values of $\sigma^2$. The bridges are "neat" for any value of $\sigma$, with dampening occurring as the value of $\sigma$ increases.

It seems however, that the computation experiences numerical errors again whenever $\sigma^2$ becomes too large, as the KL-divergence does not decrease whenever $\sigma = 2$, which is a different result than what can be concluded from figures 6.1 and 6.2.

**Computation 5 (Sum of Gaussians)**

The next computation uses a sum of Gaussian distributions for $\rho_0$. Here $\rho_0 \sim \frac{1}{2}\mathcal{N}(-1, 0.4^2) + \frac{1}{2}\mathcal{N}(0.3, 0.8^2)$ is used, whilst maintaining the shifted Gaussian distribution $\rho_1 \sim \mathcal{N}(1.5, 0.4^2)$. The results are visualised in figure 6.4.




**(a)** $\sigma = 0.2$ **(b)** KL-div, $\sigma = 0.2$ **(c)** Bridge, $\sigma = 0.2$

**(d)** $\sigma = 0.5$ **(e)** KL-div, $\sigma = 0.5$ **(f)** Bridge, $\sigma = 0.5$

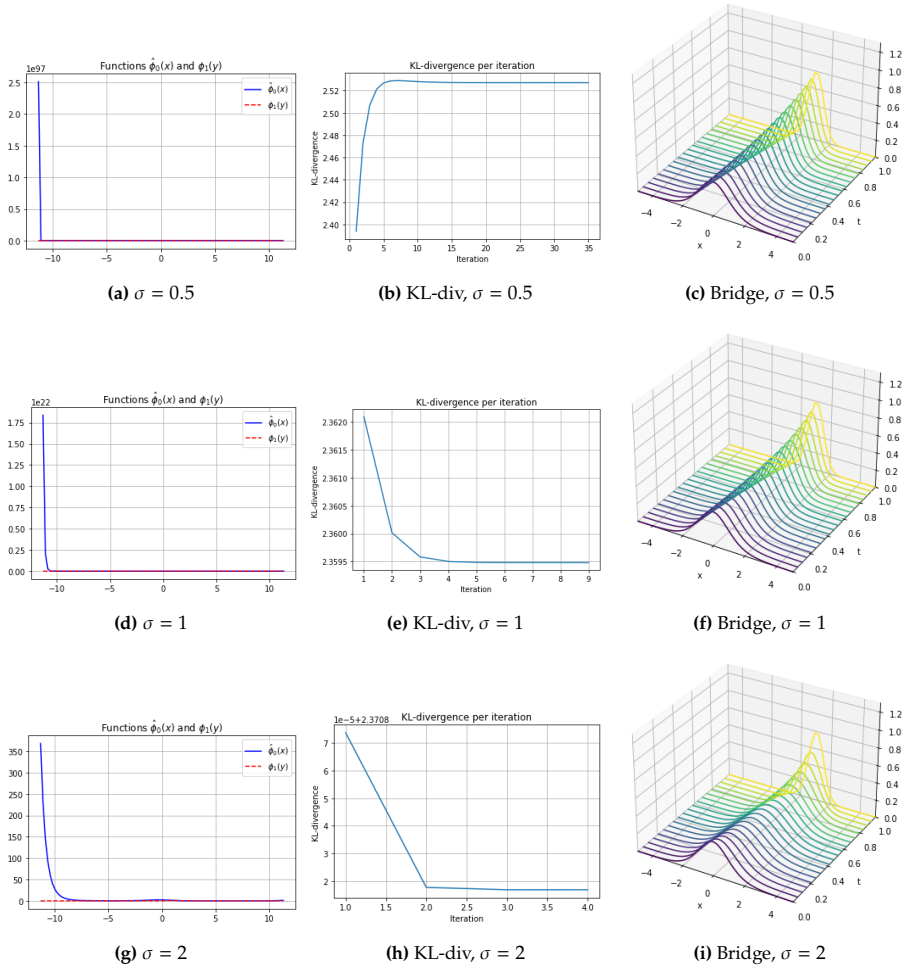
**(g)** $\sigma = 0.9$ **(h)** KL-div, $\sigma = 0.9$ **(i)** Bridge, $\sigma = 0.9$

**(j)** $\sigma = 2$ **(k)** KL-div, $\sigma = 2$ **(l)** Bridge, $\sigma = 2$

**Figure 6.4:** Computations for $\rho_0 \sim \frac{1}{2}\mathcal{N}(-1, 0.4^2) + \frac{1}{2}\mathcal{N}(0.3, 0.8^2)$, $\rho_1 \sim \mathcal{N}(1.5, 0.4^2)$, using kernel $q_{\sigma^2}$ with varying values of $\sigma$.

This computation shows expected, similar behaviour as to what has been seen before in previous computations. Again, increasing the value of $\sigma$ stabilises the solution. For these marginals $\rho_0$ and $\rho_1$, a larger value of $\sigma$ is needed to find a decreasing curve of the KL-divergence than before. It can be seen that for $\sigma = 0.5$, the decrease does not occur yet, as it has done for previous computations.

**Computation 6 (Student-t distribution - with varying $\sigma$)**
The next computation uses a non-Gaussian distribution for $\rho_0$. It instead uses a *student-t* distribution, which is fairly similar, but has a longer "tail", producing significantly different results for the computations.




**(a)** $\sigma = 0.5$ **(b)** KL-div, $\sigma = 0.5$ **(c)** Bridge, $\sigma = 0.5$

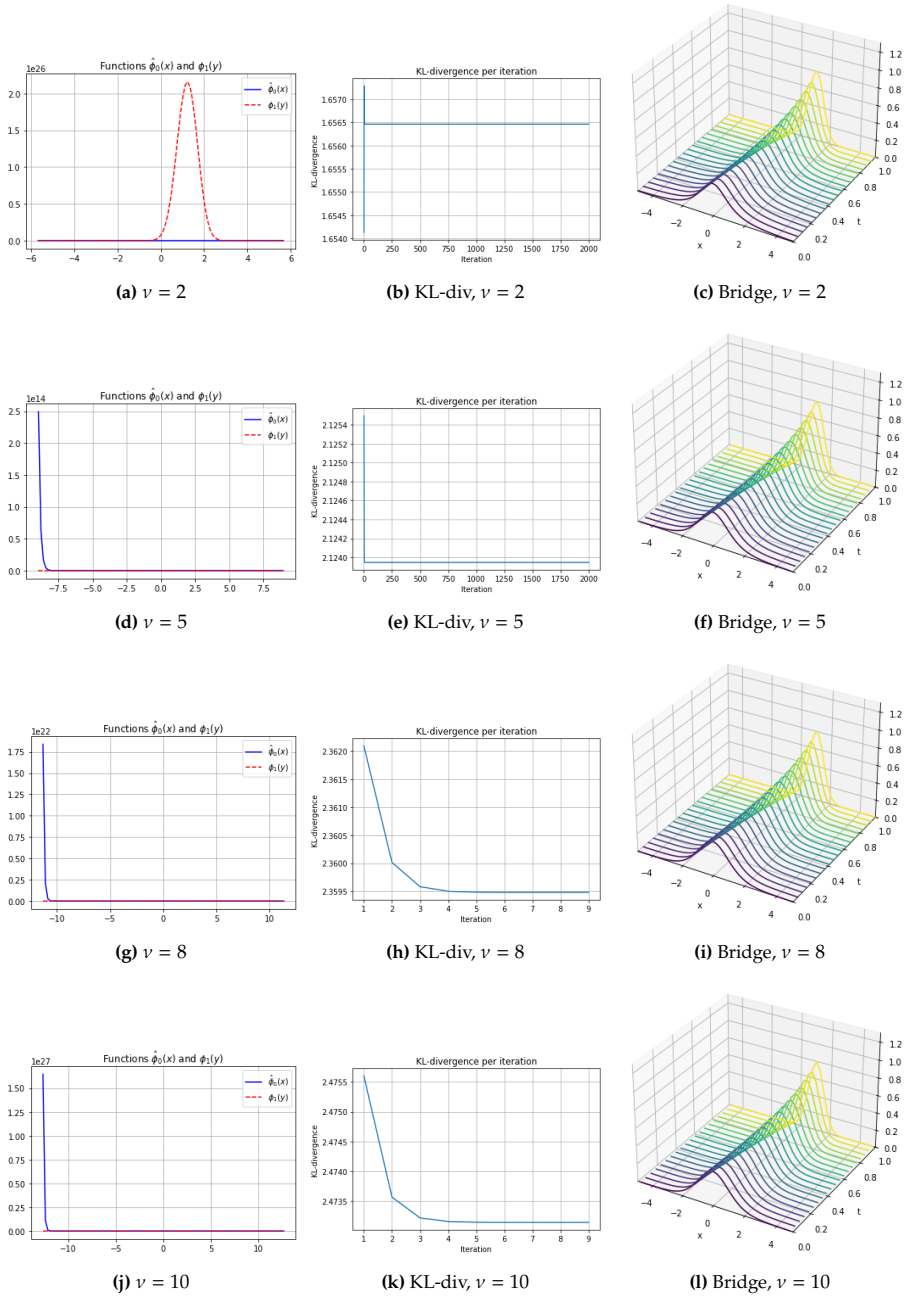
**(d)** $\sigma = 1$ **(e)** KL-div, $\sigma = 1$ **(f)** Bridge, $\sigma = 1$

**(g)** $\sigma = 2$ **(h)** KL-div, $\sigma = 2$ **(i)** Bridge, $\sigma = 2$

**Figure 6.5:** Computations for $\rho_0 \sim t_8$, $\rho_1 \sim \mathcal{N}(1, 0.5^2)$, using kernel $q_{\sigma^2}$ with varying values of $\sigma$

Choosing $\rho_0$ as a student-t distribution with eight degrees of freedom, that is, $\rho_0 \sim t_8$, and $\rho_1 \sim \mathcal{N}(1, 0.5^2)$ a shifted Gaussian distribution, figure 6.5 is produced through the computational algorithm. Clearly, taking this student-t distribution creates oddly scaled functions $\hat{\phi}_0$ and $\phi_1$, and a larger value of $\sigma$ is needed to obtain a decreasing curve of the KL-divergence. The bridges still seem to be visualised nicely, regardless of their computations of the update functions and KL-divergence.

**Computation 7 (Student-t distribution - with varying $\nu$)**
In the last computation, $\rho_0$ is again a student-t distribution, for which now there will be variated over the degrees of freedom $\nu$. Here, $\rho_1 \sim \mathcal{N}(1, 0.5^2)$ is maintained and the kernel $q_{\sigma^2}$ is used with a constant value $\sigma^2 = 1$. The influence of $\nu$ on the output of the algorithm is explored.




**(a)** $\nu = 2$ **(b)** KL-div, $\nu = 2$ **(c)** Bridge, $\nu = 2$

**(d)** $\nu = 5$ **(e)** KL-div, $\nu = 5$ **(f)** Bridge, $\nu = 5$

**(g)** $\nu = 8$ **(h)** KL-div, $\nu = 8$ **(i)** Bridge, $\nu = 8$

**(j)** $\nu = 10$ **(k)** KL-div, $\nu = 10$ **(l)** Bridge, $\nu = 10$

**Figure 6.6:** Computations for $\rho_0 \sim t_\nu$ with varying values of $\nu$, $\rho_1 \sim \mathcal{N}(1, 0.5^2)$, using kernel $q_1$.

As is shown in figure 6.6, increasing the degrees of freedom $\nu$ stabilises the solution. Although functions $\hat{\phi}_0$ and $\phi_1$ are oddly scaled for every shown value of $\nu$, the KL-divergence does decrease for the values $\nu = 5, 8$ and 10. Again, the bridges are visualised neatly regardless of the value of $\nu$ or $\sigma$.

What is rather interesting about these computations is that for $\nu$ being too small, the algorithm does not seem to converge. The relative errors do decrease over the iterations, but they stall at $RE = 0.03$ for $\nu = 2$ and $RE = 2,9 * 10^{-4}$ for $\nu = 5$. This suggests that small values of $\nu$ might be extremely sensitive to numerical underflow or overflow.

## 6.4. Notes on the bridge visualisation

In this section, some notes are made on the results that are found in chapters 5 and 6. Though various results have already been discussed for individual computations, the following comments offer more general remarks:

- Figures 6.1 and 6.2 share a lot of similarity with figure 2.1, showing that increasing the value of $\varepsilon$ or $\sigma$ causes more dampening, which is further supported throughout computations 4-7. Indeed, $\varepsilon$ and $\sigma$ have a similar effect: enlarging the value of $\sigma$ causes the algorithm to converge faster, but also creates less accurate results. This aligns with the effect of $\varepsilon$ on equation 2.1: as the value of $\varepsilon$ increases, the entropy regulariser starts to dominate. Consequently, results with greater entropic noise, and thus a higher approximation rate, are found.
- Since Chen et al. [4] uses a different transition kernel than the one used throughout this paper, it seems that $\varepsilon$ has a more extreme dampening effect than $\sigma^2$ has in the computations of this paper. However, by the use of non-identical kernels, these two parameters cannot be compared directly and thus no further conclusion on their similarity can be made here.
- A subject of further research could be the exploration of the interesting fact that bridge visualisations remain neat, even if the minimum value of the KL-divergence is not found through the computational algorithm.
- What additionally seems to be the case is that the computational algorithm becomes more sensitive to numerical errors as marginals stray further away from the "standard" Gaussian zero mean distributions. One could examine the numerical stability of the algorithm for different marginal distributions, as well as the reason as to why the algorithm fails to minimise the Schrödinger bridge problem for certain input values.

# 7

# Conclusion

In chapter 2, it was presented that the Optimal Transport problem can be approximated with an entropy term, leading to it's equivalence with the Schrödinger Bridge problem.

By regarding the originally physical density constraints as marginal probability densities, it has been established that solving the optimal transport problem is equivalent to minimising the relative entropy, or KL-divergence, between two probability laws $\mathcal{P}$ and $\mathcal{Q}$. Here, $\mathcal{Q}$ is the known probability law induced by a kernel $q_{\sigma^2}$ that describes Brownian motion. $\mathcal{P}$ acts as the minimiser, and represents the most likely way to go from one density to the other.

The unique optimal probability law $\mathcal{P}$ exists under the correct assumptions, and can be found algorithmically: in chapter 3 a derivation of the general algorithm is found, using the Lagrangian. The similarities between the SB problem and the discrete entropic OT problem have been made clear: both are solved by an algorithm that is induced from finding the stationary points of the Lagrangian, and have converging update steps.

One difference between the two solution methods is that the discrete counterpart is solved using matrix algebra, whereas the solution to the SB problem is found by integrating. Another important difference is that the solution to the discrete OT problem is found directly, while in the continuous case it needs to be solved from the perspective of a SB problem to find the appropriate update steps.

From the general algorithm found in chapter 3, analytical expressions were derived for finding the solution whenever marginals are Gaussian. For Gaussian marginals with both zero and non-zero mean, algorithms that update coefficients within the functions $\hat{\phi}_0$ and $\phi_1$, rather than updating these functions as a whole, were obtained. Using this algorithm, numerical integration is largely omitted, and thus it creates results for which less numerical approximation is needed.

When comparing the performance of both computational algorithms, in the form of codes found in appendices C.1, D and E, it can be seen that the general algorithm produces nearly the exact same results as both the algorithms for Gaussian marginals, where all algorithms fail if the entropy term $\sigma^2$ is too close to zero. It can still be tested if the algorithm of appendix E returns the same results found in computation 1 of section 5.2, whenever the input $\mu = 0$ is given.

An interesting result is that the value of the KL-divergence seems to not, or barely, depend on the value of the mean $\mu$, as computations 1 and 2 produce almost the exact same value for the KL-divergence. The precise influence of $\mu$ on the solution to the SB problem can be a relevant topic for further research.

Since the general algorithm overall converges faster than the ones for the Gaussians, along with the fact that this algorithm can also be used for non-Gaussian marginals, it can be considered as the preferable algorithm for further computations.

For both Gaussian and non-Gaussian marginals, bridges have been made with the code of appendix C.2: these visualise intermediate distributions $\rho_t$, that show how the initial distribution changes into the target one over time. It has been revealed that increasing the value of $\sigma^2$ in the Brownian motion

kernel $q_{\sigma^2}$ causes the algorithm to converge faster, but also uses a larger rate of approximation, which is illustrated as dampening in the bridge visualisations.

For values of $\sigma^2$ that are too close to zero, the algorithm again fails to minimise the KL-divergence for non-Gaussian marginals, likely due to numerical instabilities. It appears that minimal value of $\sigma^2$ needed for the algorithm to work increases with the computational complexity of the marginals. In other words, the more intricate the marginal distribution is, the more "well-behaved", or favourable, conditions are needed for the algorithm to be reliable.

Overall, the algorithm found in chapter 3 produces a functional computational algorithm under the right conditions. Clear visualisations of the Optimal Transport route for any given initial and target distribution are computed, provided that these are represented as a probability density functions.

# References


[1] J.D. Benamou. *Optimal Transportation, modelling and numerical simulation*. Acta Numerica, Cambridge University Press (CUP), 2021.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambrigde University Press, 2004.

[3] C. Brezinski and L. Wuytack. *Numerical Analysis: Historical Developments in the 20th Century*. Elsevier, 2001.

[4] Y. Chen, T.T. Georgiou, and M. Pavon. "Optimal transport in systems and control". In: *Annual Review of Control, Robotics, and Autonomous Systems.* (2020).

[5] M. Cuturi. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport". In: *Graduate School of Informatics, Kyoto University* (n.d.).

[6] M. Essid and M. Pavon. "Traversing the Schrödinger Bridge strait: Robert Fortet's marvelous proof redux". In: *Courant Institute of Mathematical Sciences at NYU and Dipartimento di Matematica Tullio Levi-Civita at Università di Padova* (2018).

[7] R. Fortet. "Résolution d'un système d'équations de M. Schrödinger." In: *J. Math. Pures Appl.* (1940).

[8] G. Monge. "Mémoire sur la théorie des déblais et des remblais". In: *Paris: De l'Imprimerie Royale* (1781).

[9] G. Peyré and M. Cuturi. *Computational Optimal Transport*. Foundations and Trends in Machine Learning, 2019.

[10] L.C. Torres, L. M. Pereira, and M. Hadi Amini. "A Survey on Optimal Transport for Machine Learning: Theory and Applications". In: *Florida International University* (2021).

[11] C. Vuik et al. *Numerical Methods for Ordinary Differential Equations*. TU Delft OPEN, 2023.

# A

# Derivation of the strict convexity of $f(P) = \langle P, C \rangle - \varepsilon H(P)$

This appendix focuses on deriving strict convexity of the objective function:

$$f(P) = \langle P, C \rangle - \varepsilon H(P) = \sum_{i,j} C_{i,j} P_{i,j} + \varepsilon \sum_{i,j} P_{i,j}(\log(P_{i,j}) - 1), \tag{A.1}$$

for any value of $\varepsilon > 0$. This property of strict convexity is used for showing the existence of a unique solution to the discrete OT problem, as described in chapter 3. Do find the derivation, $f(P)$ is split into two sums of functions:

$$f(P) = \sum_{i,j} g_{i,j}(P_{i,j}) + \varepsilon \sum_{i,j} h(P_{i,j}), \tag{A.2}$$

$$with:$$

$$g_{i,j}(x) = C_{i,j} x; \tag{A.3}$$

$$h(x) = x(\log(x) - 1). \tag{A.4}$$

Since $g_{i,j}(x)$ is linear for all values $C_{i,j}$, it follows immediately from the definition of convex functions that $g_{i,j}(x)$ is convex for all $x \in \mathbb{R}, i, j, \in \{1, ...., n\}$. By the note that a sum of convex functions is again convex, one obtains that $\sum_{i,j} g_{i,j}(P_{i,j})$ is convex for any matrix $P$.

The convexity of $h(x)$ is established by showing its second derivative is positive on the chosen domain. By taking its derivative with respect to $x$ twice:

$$h'(x) = \frac{\mathrm{d}h(x)}{\mathrm{d}x} = \log(x) - 1 + x\frac{1}{x} = \log(x) - 1 + 1 = \log(x). \tag{A.5}$$

And thus:

$$h''(x) = \frac{\mathrm{d}h'(x)}{\mathrm{d}x} = \frac{\mathrm{d}\log(x)}{\mathrm{d}x} = \frac{1}{x}, \tag{A.6}$$

exists and is positive whenever $x \in \mathbb{R}_{>0}$.

One of the constraints for the optimal transport problem is that $P$ is everywhere positive, and so consequently:

$$h''(P_{i,j}) > 0 \qquad \forall i, j \in \{1, ...., n\}. \tag{A.7}$$

By the notion of the strict inequality, strict convexity holds. For strict convexity, a sum of its functions is once again strictly convex, which gives $\sum_{i,j} h(P_{i,j})$ the desired property.

With the obtained results, the strict convexity of $f(P)$ for $\varepsilon > 0$ is shown by its definition. First note that for any two matrices $P$ and $R$ with only positive entries, and $t \in (0, 1)$:

$$\sum_{i,j} g_{i,j}(tP_{i,j} + (1-t)Q_{i,j}) = t \sum_{i,j} g_{i,j}(P_{i,j}) + (1-t) \sum_{i,j} g_{i,j}(Q_{i,j}); \qquad \textit{(by linearity)} \tag{A.8}$$

$$\sum_{i,j} h(tP_{i,j} + (1-t)Q_{i,j}) < t \sum_{i,j} h(P_{i,j}) + (1-t) \sum_{i,j} h(Q_{i,j}). \qquad \textit{(by strict convexity)} \tag{A.9}$$

Therefore, for $\varepsilon > 0$:

$$\begin{aligned} f(tP + (1-t)Q) &= \textstyle\sum_{i,j} g_{i,j}(tP_{i,j} + (1-t)Q_{i,j}) + \varepsilon \sum_{i,j} h(tP_{i,j} + (1-t)Q_{i,j}) \\ &= t \textstyle\sum_{i,j} g_{i,j}(P_{i,j}) + (1-t) \sum_{i,j} g_{i,j}(Q_{i,j}) + \varepsilon \sum_{i,j} h(tP_{i,j} + (1-t)Q_{i,j}) \\ &< t \textstyle\sum_{i,j} g_{i,j}(P_{i,j}) + (1-t) \sum_{i,j} g_{i,j}(Q_{i,j}) + \varepsilon \left( t \sum_{i,j} h(P_{i,j}) + (1-t) \sum_{i,j} h(Q_{i,j}) \right) \\ &= t \left( \textstyle\sum_{i,j} g_{i,j}(P_{i,j}) + \varepsilon \sum_{i,j} h(P_{i,j}) \right) + (1-t) \left( \sum_{i,j} g_{i,j}(Q_{i,j}) + \varepsilon \sum_{i,j} h(Q_{i,j}) \right) \\ &= t f(P) + (1-t) f(Q). \end{aligned} \tag{A.10}$$

And thus $f(P)$ is strictly convex for all positive values of $\varepsilon$.

# B

# Evaluation of integrals in the 1D Bernstein case

## B.1. Evaluation of $\int_{\mathbb{R}} \exp(\frac{-(y-x)^2}{2\sigma^2})\mathrm{d}x$

To find equation $\hat{\phi}_1^1$ of the SBP algorithm in the analytical Gaussian cases, as described in chapter 4, the integral is evaluated using change of variables three times.

Recall that $\hat{\phi}_1^1$ is defined by equation 4.10 as:

$$\hat{\phi}_1^1(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{\mathbb{R}} \exp(\frac{-(y-x)^2}{2\sigma^2})\mathrm{d}x. \tag{B.1}$$

First, the substitution $u := x - y$ and consequently $\mathrm{d}u = \mathrm{d}x$ is made, and the integral is evaluated by taking its square:

$$\begin{aligned} & (\textstyle\int_{\mathbb{R}} \exp(\frac{-u^2}{2\sigma^2})\mathrm{d}u)^2 \\ = \quad & \textstyle\int_{\mathbb{R}} \exp(\frac{-u^2}{2\sigma^2})\mathrm{d}u \int_{\mathbb{R}} \exp(\frac{-v^2}{2\sigma^2})\mathrm{d}v \\ = \quad & \textstyle\int_{\mathbb{R}^2} \exp(\frac{-(u^2+v^2)}{2\sigma^2})\mathrm{d}u\mathrm{d}v. \end{aligned} \tag{B.2}$$

Using change of variables a second time to evaluate the integral using polar coordinates $r \in [0, \infty)$ and $\theta \in [0, 2\pi]$, where:

$$\begin{array}{llll} u & := r\cos\theta & \text{and} & v & := r\sin\theta, \\ \mathrm{d}u\mathrm{d}v & = r\mathrm{d}r\mathrm{d}\theta, & \text{and consquently} & u^2+v^2 & = r^2. \end{array} \tag{B.3}$$

Equation B.2 can then be solved as follows:

$$\begin{aligned} (\textstyle\int_{\mathbb{R}} \exp(\frac{-u^2}{2\sigma^2})\mathrm{d}u)^2 &= \textstyle\int_0^{2\pi}\int_0^{\infty} \exp(\frac{-r^2}{2\sigma^2})r\mathrm{d}r\mathrm{d}\theta \\ &= \textstyle\int_0^{2\pi}(\int_0^{\infty} \exp(\frac{-r^2}{2\sigma^2})r\mathrm{d}r)\mathrm{d}\theta. \end{aligned} \tag{B.4}$$

In order to solve the inner integral of equation B.4, one last change of variables is made:

$$\begin{array}{lrcl} & s & := & \frac{r^2}{2\sigma^2}. \\ \text{Then:} & \mathrm{d}s & = & \frac{r}{\sigma^2}\mathrm{d}r, \\ \text{and thus:} & r\mathrm{d}r & = & \sigma^2\mathrm{d}s. \end{array} \tag{B.5}$$

Substituting this into the inner integral of B.4 gives:

$$\begin{aligned} \int_0^\infty \exp(\tfrac{-r^2}{2\sigma^2}) r \mathrm{d}r &= \int_0^\infty \exp(-s) * \sigma^2 \mathrm{d}s \\ &= \sigma^2 \int_0^\infty \exp(-s)\mathrm{d}s \\ &= \sigma^2[-\exp(-s)]_0^\infty \\ &= \sigma^2(-0+1) \\ &= \sigma^2. \end{aligned} \tag{B.6}$$

Thus we obtain:

$$\begin{aligned} (\int_{\mathbb{R}} \exp(\tfrac{-u^2}{2\sigma^2})\mathrm{d}u)^2 &= \int_0^{2\pi} \sigma^2 \mathrm{d}\theta \\ &= \sigma^2 \int_0^{2\pi} 1 \mathrm{d}\theta \\ &= \sigma^2[\theta]_0^{2\pi} \\ &= \sigma^2(2\pi - 0) \\ &= 2\pi\sigma^2. \end{aligned} \tag{B.7}$$

Therefore, the solution for $\hat{\phi}_1^1$ is found as described in equation 4.10,and it is found that $\hat{\phi}_1^1 = 1$.

## B.2. Evaluation of $\int_{\mathbb{R}} \exp\big(-\frac{1}{2}\frac{1}{A_0}(y-A_1)^2 + A_2\big)\mathrm{d}y$

Evaluating the more general integral $\int_{\mathbb{R}} \exp\big(-\frac{1}{2}\frac{1}{A_0}(y-A_1)^2 + A_2\big)\mathrm{d}y$ is done using similar substitutions as for the integral in section B.1, and is as follows:

First note that $\exp(A_2)$ is constant w.r.t. $y$ and thus:

$$\int_{\mathbb{R}} \exp\big(-\frac{1}{2}\frac{1}{A_0}(y-A_1)^2 + A_2\big)\mathrm{d}y = \exp(A_2) \int_{\mathbb{R}} \exp\big(-\frac{1}{2}\frac{1}{A_0}(y-A_1)^2\big)\mathrm{d}y. \tag{B.8}$$

Then using the substitution $u = y - A_1$ and thus $\mathrm{d}u = \mathrm{d}y$, and evaluating the square of the integral:

$$\begin{aligned} & \big(\int_{\mathbb{R}} \exp\big(-\tfrac{1}{2}\tfrac{1}{A_0}(y-A_1)^2\big)\mathrm{d}y\big)^2 \\ = \; & \big(\int_{\mathbb{R}} \exp(\tfrac{-1}{2A_0}u^2)\mathrm{d}u\big)^2 \\ = \; & \int_{\mathbb{R}} \exp(\tfrac{-1}{2A_0}u^2)\mathrm{d}u \int_{\mathbb{R}} \exp(\tfrac{-1}{2A_0}v^2)\mathrm{d}v \\ = \; & \int_{\mathbb{R}^2} \exp(\tfrac{-1}{2A_0}((u^2+v^2))\mathrm{d}u\mathrm{d}v. \end{aligned} \tag{B.9}$$

Then again, by using polar coordinates $r \in [0, \infty)$ and $\theta \in [0, 2\pi]$, equation B.9 becomes:

$$\begin{aligned} (\int_{\mathbb{R}} \exp(\tfrac{-1}{2A_0}u^2)\mathrm{d}u)^2 &= \int_0^{2\pi}\int_0^\infty \exp(\tfrac{-1}{2A_0}r^2) r \mathrm{d}r\mathrm{d}\theta \\ &= \int_0^{2\pi}(\int_0^\infty \exp(\tfrac{-1}{2A_0}r^2) r \mathrm{d}r)\mathrm{d}\theta. \end{aligned} \tag{B.10}$$

Making the final substituition $s = \frac{1}{2}r^2$ and thus $\mathrm{d}s = r\mathrm{d}r$, equation B.9 is evaluated by:

$$\begin{aligned} (\int_{\mathbb{R}} \exp(\tfrac{-1}{2A_0}u^2)\mathrm{d}u)^2 &= \int_0^{2\pi}\int_0^\infty \exp(\tfrac{-s}{A_0})\mathrm{d}s\mathrm{d}\theta \\ &= \int_0^{2\pi}[-A_0\exp(\tfrac{-s}{A_0})]_0^\infty \mathrm{d}\theta \\ &= \int_0^{2\pi}\big(-A_0(0-1)\big)\mathrm{d}\theta \\ &= A_0\int_0^{2\pi} 1 \mathrm{d}\theta \\ &= 2\pi A_0. \end{aligned} \tag{B.11}$$

Therefore, by equations B.8 and B.11 it can be concluded that:

$$\int_{\mathbb{R}} \exp\big(-\frac{1}{2}\frac{1}{A_0}(y-A_1)^2 + A_2\big)\mathrm{d}y = \exp(A_2)\sqrt{2\pi A_0}. \tag{B.12}$$

# C

# Code for general marginals

The code used to find the numerical solutions to the SB problem with general marginals is given below. Note that in this code, $\rho_0$ and $\rho_1$ are coded as Gaussians, and $q$ is the Brownian motion kernel. However, this code can easily be adjusted by altering only these functions, so that it can be used for any set of marginals and kernels.

Section C.1 provides the code that produces the functions $\hat{\phi}_0$ & $\phi_1$ as well as the value of the KL-divergence per iteration. In section C.2, an extension to this code is given, used for making 3D plots of the intermediate distributions. This creates the so-called bridge.

## C.1. General code for computation $\pi$ and the KL-divergence

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 29 11:10:03 2025

@author: lunah
"""

#Schrodinger Bridge algorithm for general marginals

#Return a measure P01 on R X R that solves the Schrodinger Bridge
#P01 = integral(q(0,x,1,y)*phihat0(x)*phi1(y) dxdy)

import math
import sympy as smp
import numpy as np
import matplotlib.pyplot as plt
from sympy.abc import x,y
from scipy.interpolate import interp1d
#from scipy.special import gamma    activate for student-t or gamma distributions



def SBeqs(rho0_sym,rho1_sym, q_sym):
    #symbolise the variables for x and y:
    x=smp.Symbol("x")
    y=smp.Symbol("y")

    #rewrite the symoblic functions to evaluate numerically:
    rho0=smp.lambdify(x,rho0_sym, 'numpy')
    rho1 = smp.lambdify(y, rho1_sym, 'numpy')


    #define the grid that adapts bounds based on standard deviations:
    a = min(mu0 - 4*sigma0, mu1 - 4*sigma1) #lower bound
    b = max(mu0 + 4*sigma0, mu1 + 4*sigma1) #upper bound
    N=101                                   #nr. of grid points
```

```
    xvals=np.linspace(a,b,N)
    yvals=np.linspace(a,b,N)
    X,Y=np.meshgrid(xvals, yvals, indexing='ij')

    #rewriting symbolic kernel as a 2D function:
    q = smp.lambdify((x, y), q_sym, 'numpy')
    q_vals=q(X,Y)

    #Initialise the value of phihat0 and phi1:
    phihat0_xvals=np.ones_like(xvals)
    phi1_yvals = np.ones_like(yvals)
    #this will be adapted in the algorithm

    #initialising KL-divergence
    KL=[]

    #The following lines in a for-loop until convergence:
    tolerance = 1e-5
    max_iter=2000
    for i in range(max_iter):

        #save old values of phihat0 and phi1 in order to calculate RE
        phihat0_old = phihat0_xvals.copy()
        phi1_old = phi1_yvals.copy()

        #Compute phihat1(y) using trapezoidal method integral
        #phihat1(y)= integral(q(0,x,1,y)*phihat0(x)dx)
        #integrand=q(xvals,y)*phihat0(xvals)
        phihat1_yvals = np.array(
            [np.trapezoid(q(xvals,y)*phihat0_xvals, xvals) for y in yvals])


        #Compute phi(y) with phihat1(y)
        phi1_log =np.log(np.clip(rho1(yvals), 1e-300, None)) - np.log(np.clip(phihat1_yvals,
            1e-300, None))
        phi1_yvals=np.exp(phi1_log)

        #Compute phi0(x) using trapezoidal method integral
        #phi0(x) = integral(q(0,x,1,y)*phi1(y)dy)
        #integrand = q(x,yvals)*phi1(yvals)
        phi0_xvals=np.array(
            [np.trapezoid(q(x,yvals)*phi1_yvals, yvals) for x in xvals])


        #Update phihat0(x)
        phihat0_log = np.log(np.clip(rho0(xvals), 1e-300, None)) - np.log(np.clip(phi0_xvals,
             1e-300, None))
        phihat0_xvals=np.exp(phihat0_log)


        #joint distribution function pi(x,y)
        eps=1e-300
        pi_log=np.log(q_vals+eps)+np.log(phihat0_xvals+eps)+np.log(phi1_yvals+eps)
        pi=np.exp(pi_log)

        #sanity checking if pi and g are PDFs
        pi_integral=np.trapezoid(np.trapezoid(pi,yvals, axis=1),xvals)
        q_integral=np.trapezoid(np.trapezoid(q_vals,yvals, axis=1),xvals)

        #normalisation so that pi and g are considered PDFs
        pi=pi/pi_integral
        q_vals=q_vals/q_integral

        #defining the integrand and avoiding log(0) or devision by zero
        log_ratio = np.log(pi+eps) -np.log(q_vals+eps)
        KL_integrand = pi*log_ratio

        #double integrating to obtain KL divergence
        KL_inner_int=np.trapezoid(KL_integrand, xvals)
        KL_total_int=np.trapezoid(KL_inner_int, yvals)
```

```
        #adding the value to the list
        KL.append(KL_total_int)

        #sanity checking if rho0 and rho1 are weighted properly
        #integrals must be 1
        rho0_int = np.trapezoid(phihat0_xvals*phi0_xvals, xvals)
        rho1_int = np.trapezoid(phihat1_yvals*phi1_yvals, yvals)

        if not (0.95 < rho0_int <1.05):
            print(f"Unbounded function rho0 after {i+1} iterations.")
            break

        if not (0.95 < rho1_int <1.05):
            print(f"Unbounded function rho1 after {i+1} iterations.")
            break

        #if not: normalise

        #checking that there are no negative values in phihat0 and phi1:
        if np.any(phihat0_xvals < 0):
            print("Error: negative values in phihat0." )
            break

        if np.any(phi1_yvals < 0):
            print("Error: negative values in phi1." )
            break


        #Calculate relative errors
        REx=np.linalg.norm(phihat0_xvals-phihat0_old)/np.linalg.norm(phihat0_old)
        REy=np.linalg.norm(phi1_yvals-phi1_old)/np.linalg.norm(phi1_old)

        #stopping condition
        if REx < tolerance and REy < tolerance:
            print(f"Converged in {i} iterations.")
            break

    #rewriting final phihat0 and phi1 back into symoblic expression approx. in exponential
        form

    #first make sure log(0) does not happen
    phihat0_clipped = np.clip(phihat0_xvals, 1e-10, None)
    log_phihat0 = np.log(phihat0_clipped)

    phi1_clipped=np.clip(phi1_yvals, 1e-10, None)
    log_phi1 = np.log(phi1_clipped)

    #fit an polynomial of degree 2  (for ex.) to the log of the function
    coeffs0=np.polyfit(xvals, log_phihat0, deg=2)
    coeffs1=np.polyfit(yvals, log_phi1, deg=2)

    #rounding and thresholding the coefficients:
    rounded_coeffs0 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs0]
    rounded_coeffs1 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs1]

    #creating a symbolic expression in an approximation of an exponential
    phihat0_sym=sum(c*x**i for i, c in enumerate(reversed(rounded_coeffs0)) if c!=0)
    phihat0_approx=smp.exp(phihat0_sym)

    phi1_sym=sum(c*y**i for i, c in enumerate(reversed(rounded_coeffs1)) if c!= 0)
    phi1_approx=smp.exp(phi1_sym)

    print("\n phihat0(x) = \n", phihat0_approx,
          "\n phi1(y) = \n", phi1_approx)
    return phihat0_xvals, phi1_yvals, rho0_int, rho1_int, KL

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

#initialising marginals rho0, rho1 and kernel q

#this is used for normal distributions, can be adapted
```

```
mu0=float(input("choose a value for _0 : "))
mu1=float(input("choose a value for _1 : "))

#nu0 = float(input("Choose degrees of freedom  _0 : ")) for student-t

sigma0=float(input("choose a value for _0 :" ))
sigma1=float(input("choose a value for _1 :" ))
sigma=float(input("choose a value for  :" ))


rho0 =  1/smp.sqrt(2*math.pi*sigma0**2)*smp.exp(-(x-mu0)**2/(2*sigma0**2))

        #for student-t: (gamma((nu0 + 1) / 2) /
        #               (smp.sqrt(nu0 * math.pi) * gamma(nu0 / 2))) * \
        #               (1 + ((x - mu0) ** 2) / nu0) ** (-(nu0 + 1) / 2)

rho1=1/smp.sqrt(2*math.pi*sigma1**2)*smp.exp(-(y-mu1)**2/(2*sigma1**2))
q=1/smp.sqrt(2*math.pi*sigma**2)*smp.exp(-(x - y)**2/(2*sigma**2))



#finding phihat0, phi1 and the KL-divergence with the above function
[phihat0, phi1, rho0_int, rho1_int, KL_div]=SBeqs(rho0,rho1,q)

#plotting the new found functions phihat0(x) and phi1(y):


#define the grid that adapts bounds based on standard deviations:
a = min(mu0 - 4*sigma0, mu1 - 4*sigma1) #lower bound
b = max(mu0 + 4*sigma0, mu1 + 4*sigma1) #upper bound
N=101                                   #nr. of grid points

x_plot=np.linspace(a,b, N)
y_plot=np.linspace(a,b, N)
X,Y = np.meshgrid(x_plot,y_plot, indexing = 'ij')


#creating the plots:
plt.plot(x_plot, phihat0, label="$\hat\phi_0(x)$", color='blue')
plt.plot(x_plot, phi1, label=r'$\phi_1(y)$', color='red', linestyle='--')
#plt.axis((-10,10,0,1.4))
#plt.xlabel("x")
#plt.ylabel("Value")
plt.title("Functions $\hat\phi_0(x)$ and $\phi_1(y)$")
plt.legend()
plt.grid(True)
plt.show()


#plotting KL-divergence for every iteration:
x_numbers = list(range(1, len(KL_div)))

plt.plot(x_numbers, KL_div[1:])
plt.title("KL-divergence per iteration")
plt.grid(True)
#plt.axis((1,len(KL), ymin, ymax))
plt.xlabel("Iteration")
plt.ylabel("KL-divergence")
plt.show()



##################################################
#sanity check: double integral of pi over R^2 should return 1
def P01_trapezoid(phihat0,phi1,q_sym):
    #Return a measure P01 on R X R that solves the Schrodinger Bridge
    #Using linear interpolation trapezoidal method to find the double integral

    #Input:
    #phihat0(x): first of the pair of functions for the solution of the SBP.
    #phi1(y):    second of the pair of functions for the solution of the SBP.
```

```
    #q(0,x,1,y): the Markov transition probability kernel.

    #symbolise the variables for x and y:
    x=smp.Symbol("x")
    y=smp.Symbol("y")

    #making sure that q can be evaluated numerically:
    q = smp.lambdify((x, y), q_sym, 'numpy')

    #define the grid, that adapts bounds based on standard deviations:
    a = min(mu0 - 4*sigma0, mu1 - 4*sigma1)
    b = max(mu0 + 4*sigma0, mu1 + 4*sigma1)

    N=101
    xvals=np.linspace(a,b,N)
    yvals=np.linspace(a,b,N)

    phihat0_func = interp1d(xvals, phihat0, kind='linear', bounds_error=False, fill_value
        =0.0)
    phi1_func = interp1d(yvals, phi1, kind='linear', bounds_error=False, fill_value=0.0)


    integrand=np.zeros((N,N))
    for i, xi in enumerate(xvals):
        for j, yj in enumerate(yvals):
            integrand[i, j] = q(xi, yj) * phihat0_func(xi) * phi1_func(yj)


    #inner integrand = q(0,x,1,y)*phihat0(x)*phi1(y) integrated over x
    inner_int=np.trapezoid(integrand, xvals) #integrate over x
    total_int=np.trapezoid(inner_int, yvals) #integrate over y
    return total_int

P01_R=P01_trapezoid(phihat0,phi1,q)
```

## C.2. Code extension for Schrödinger Bridge visualisation

```
##########################################
#Visualisation of the SB

#defining a time grid:
t_0 = 0                             #starting time
t_n = 1                             #stopping time
n = 20                              #nr of time points
times = np.linspace(t_0, t_n, n)    #time grid
dt = times[1]-times[0]              #time interval

#defining spatial grid
x_vals = np.linspace(a,b,N)
y_vals=x_vals
z_vals=x_vals
dx=x_vals[1]-x_vals[0]
dy=y_vals[1]-y_vals[0]
X,T = np.meshgrid(x_vals, times, indexing ='ij' )

def compute_pi_t(x_vals,y_vals, phihat0, phi1, q_sym):

    #Input:
    #x,y:        values x,y over which the function pi is evaluated.
    #phihat0(x): first of the pair of functions for the solution of the SBP.
    #phi1(y):    second of the pair of functions for the solution of the SBP.
    #q(0,x,1,y): the Markov transition probability kernel.


    #defining the 2D grid:
    X,Y =np.meshgrid(x_vals, y_vals, indexing ='ij')

    #making sure that q can be evaluated numerically:
    x_sym=smp.Symbol("x")
    y_sym=smp.Symbol("y")
```

```
    q_func = smp.lambdify((x_sym, y_sym), q_sym, 'numpy')
    q_vals = q_func(X,Y)

    #computing pi
    eps=1e-300
    pi_vals_log = np.log(phihat0 + eps)[:, None] + np.log(q_vals + eps) + np.log(phi1 + eps)[None, :]
    pi_vals_log = np.clip(pi_vals_log, a_min=-300, a_max=None)
    pi_vals = np.exp(pi_vals_log- pi_vals_log.max())

    #normalise pi:
    norm =np.trapezoid(np.trapezoid(pi_vals, y_vals, axis=1),x_vals)

    pi_vals=pi_vals/norm

    return pi_vals


pi=compute_pi_t(x_vals, y_vals, phihat0, phi1, q)

#defining rho_0 and rho_1 for initial and target distribution
rho_0_vals = 1/np.sqrt(2*np.pi*sigma0**2)*np.exp(-(x_vals-mu0)**2/(2*sigma0**2))
            #for student-t: (gamma((nu0 + 1) / 2) /
            #               (np.sqrt(nu0 * math.pi) * gamma(nu0 / 2))) * \
            #               (1 + ((x_vals - mu0) ** 2) / nu0) ** (-(nu0 + 1) / 2)

rho_1_vals = 1/np.sqrt(2*np.pi*sigma1**2)*np.exp(-(x_vals-mu1)**2/(2*sigma1**2))

#normalisation
rho_0_vals=rho_0_vals/np.trapezoid(rho_0_vals, x_vals)
rho_1_vals=rho_1_vals/np.trapezoid(rho_1_vals, y_vals)

#initialsing 3D plot with colour gradient
fig=plt.figure(figsize=(10,6))
ax=fig.add_subplot(111, projection='3d')
facecolors = plt.colormaps['viridis'](np.linspace(0, 1, len(range(n))))


# Restricting to the fixed x-domain [-5, 5] for the 3D plot
x_plot_mask = (x_vals >= -5) & (x_vals <= 5)
x_plot_vals = x_vals[x_plot_mask]


#initialising the plot for rho0
ax.plot(x_plot_vals, np.zeros_like(x_vals)[x_plot_mask], rho_0_vals[x_plot_mask], color = facecolors[0])

tolerance=1e-100

#initialising the plot for rho_t for each value in times
for t in range(1,n-1):

    SB_vals_t = np.zeros_like(x_vals)


    for i, z in enumerate(x_vals):
        q_start = (1 / np.sqrt(2*np.pi*sigma**2 * times[t])) * \
                  np.exp(- (x_vals - z)**2 / (2*sigma**2 * times[t]))  # shape: (len(x_vals),)

        q_end = (1 / np.sqrt(2*np.pi*sigma**2 * (1 - times[t]))) * \
                np.exp(- (y_vals - z)**2 / (2*sigma**2 * (1 - times[t])))  # shape: (len(y_vals),)

        numerator = np.outer(q_start, q_end)  # shape: (len(x_vals), len(y_vals))

        q_01 = (1 / np.sqrt(2*np.pi*sigma**2)) * \
                np.exp(- (y_vals[None, :] - x_vals[:, None])**2 / (2*sigma**2))  # shape: (len(x_vals), len(y_vals))

        log_integrand = np.log(pi + tolerance) + np.log(numerator + tolerance) - np.log(q_01
```

```
            + tolerance)
        integrand = np.exp(log_integrand)

        SB_vals_t[i] = np.trapezoid(np.trapezoid(integrand, y_vals, axis=1), x_vals)

    SB_vals_t_plot = SB_vals_t[x_plot_mask]
    ax.plot(x_vals[x_plot_mask], [times[t]]*len(x_plot_vals), SB_vals_t_plot, color=
        facecolors[t])



#initialsing the plot for rho_1
ax.plot(x_vals[x_plot_mask], np.ones_like(x_plot_vals), rho_1_vals[x_plot_mask], color =
    facecolors[n-1])

#plotting the SB
ax.set(xlabel='x', ylabel='t', zlabel='␣', xlim=(-5,5), ylim=(0,1), zlim=(0,1.3))
plt.show()
```

# D

# Code for Gaussian marginals with zero mean

The code used to find the numerical solutions to the SB problem with Gaussian marginals with zero mean is given below. The algorithm produces functions $\hat{\phi}_0$ & $\phi_1$, and the value of the KL-divergence per iteration, which are plotted as graphs.

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 13 13:29:57 2025

@author: lunah
"""

#1D Bernstein SBP algorithm
#based on analytical evaluations
#using rho0(x)=1/sqrt(2*pi*sigma_0^2)exp(-x^2/2sigma_0^2)
#using rho1(y)=1/sqrt(2*pi*sigma_1^2)exp(-y^2/2sigma_1^2)
#using g(x,y)=q(0,x,1,y)= 1/sqrt(2*pi*sigma^2)exp(-(x-y)^2/2sigma^2)

import math
import numpy as np
import sympy as smp
import matplotlib.pyplot as plt
from sympy.abc import x,y


################################################
#creating fucntions phihat0 and phi1

#initialsing values for the variances
sigma_0= float(input("choose a value for _0 :" ))
sigma_1=float(input("choose a value for _1 :" ))
sigma=float(input("choose a value for  :" ))


#initialsing coefficients a,b,c and d
a_vals=[1]
bfrac_vals=[0]
c_vals=[0]
dfrac_vals=[0]

#define tolerance for stopping condition
tolerance=1e-6

#define maximum number of iterations
max_iter=100

#creating for loop for the coefficients
```

```
for i in range(1,max_iter):
    c_vals.append(math.sqrt(2*math.pi*sigma_1**2)/(a_vals[i-1]*math.sqrt(1 + (bfrac_vals[i
        -1]*sigma**2))))
    dfrac_vals.append(1/(sigma_1**2) - 1/(sigma**2) +1/(sigma**2+(sigma**4)*bfrac_vals[i-1]))

    a_vals.append(math.sqrt(2*math.pi*sigma_0**2)/(c_vals[i]*math.sqrt(1 + (dfrac_vals[i]*
        sigma**2))))
    bfrac_vals.append(1/(sigma_0**2) - 1/(sigma**2) +1/(sigma**2+(sigma**4)*dfrac_vals[i]))

    #stopping conditions for convergence:
    REa = abs((a_vals[i]-a_vals[i-1])/max(a_vals[i-1], 1e-10))
    REb = abs((bfrac_vals[i]-bfrac_vals[i-1])/max(bfrac_vals[i-1], 1e-10))

    REc = abs((c_vals[i]-c_vals[i-1])/max(c_vals[i-1], 1e-10))
    REd = abs((dfrac_vals[i]-dfrac_vals[i-1])/max(dfrac_vals[i-1], 1e-10))

    if  REa < tolerance and REb < tolerance and REc < tolerance and REd < tolerance:
        print(f"Converged␣in␣{i}␣iterations.")
        break

#defining the grid
lb=-5                              #left boundary
rb=5                               #right boundary
N=1001                             #nr of grid points
xvals=np.linspace(lb,rb,N)         #grid for x values
yvals=np.linspace(lb,rb,N)         #grid for y values
dx=xvals[1]-xvals[0]
dy=yvals[1]-yvals[0]

#defining phihat0 and phi1 so that it can be plotted
phihat0_xvals=1/a_vals[-1]*np.exp(-xvals**2*bfrac_vals[-1]/2)
phi1_yvals = 1/c_vals[-1]*np.exp(-yvals**2*dfrac_vals[-1]/2)

#rewriting phihat0 and phi1 back into symoblic expression approx. in exponential form

#first make sure log(0) does not happen
phihat0_clipped = np.clip(phihat0_xvals, 1e-10, None)
log_phihat0 = np.log(phihat0_clipped)

phi1_clipped=np.clip(phi1_yvals, 1e-10, None)
log_phi1 = np.log(phi1_clipped)

#fit an polynomial of degree 2  (for ex.) to the log of the function
coeffs0=np.polyfit(xvals, log_phihat0, deg=2)
coeffs1=np.polyfit(yvals, log_phi1, deg=2)

#rounding and thresholding the coefficients:
rounded_coeffs0 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs0]
rounded_coeffs1 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs1]

#creating a symbolic expression in an approximation of an exponential
phihat0_sym=sum(c*x**i for i, c in enumerate(reversed(rounded_coeffs0)) if c!=0)
phihat0_approx=smp.exp(phihat0_sym)

phi1_sym=sum(c*y**i for i, c in enumerate(reversed(rounded_coeffs1)) if c!= 0)
phi1_approx=smp.exp(phi1_sym)

print("\n␣phihat0(x)␣=␣\n", phihat0_approx,
      "\n␣phi1(y)␣=␣\n", phi1_approx)

#plotting phihat0x and phi1y
plt.plot(xvals, phihat0_xvals, label="$\hat\phi_0(x)$", color='blue')
plt.plot(yvals, phi1_yvals, label=r'$\phi_1(y)$', color='red', linestyle='--')
plt.title("Functions␣$\hat\phi_0(x)$␣and␣$\phi_1(y)$")

plt.legend()
plt.grid(True)
plt.show()

##############################################
#Sanity check that P01(R) =1
```

```
P_coeff = 2*math.pi/(a_vals[-1]*c_vals[-1]*math.sqrt(2*math.pi*sigma**2))
P_root = (1/(sigma**2) + bfrac_vals[-1])*(1/(sigma**2) +dfrac_vals[-1] - 1/(sigma**2 + (
    bfrac_vals[-1]*sigma**4)))
P01_R=P_coeff/math.sqrt(P_root)


################################################3
#KL-divergence

# initialising KL-divergence
KL=[]

#2D grid for x and y
X,Y = np.meshgrid(xvals, yvals, indexing ='ij')

#defining g as a function of x and y
g=1/(sigma*math.sqrt(2*math.pi))*np.exp(-(X-Y)**2/(2*sigma**2))

#normalisation so that g is considered a PDF
g_integral=np.trapezoid(np.trapezoid(g,yvals, axis=1),xvals)
g=g/g_integral


#creating function pi for each iteration
for j in range(1,len(a_vals)):

   #defining phihat0(x) and phi1(y) for each iteration
   phihat0_per_it = 1/a_vals[j]*np.exp(-xvals**2*bfrac_vals[j]/2)
   phi1_per_it = 1/c_vals[j]*np.exp(-yvals**2*dfrac_vals[j]/2)

   Z = np.trapezoid(
       np.trapezoid(phihat0_per_it* g * phi1_per_it, yvals, axis=1),
       xvals)

   phihat0_per_it /= np.sqrt(Z)
   phi1_per_it /= np.sqrt(Z)


   #joint distribution pi(x,y)
   pi= g*phihat0_per_it*phi1_per_it

   # Normalize pi so that it integrates to 1
   pi_integral_y = np.trapezoid(pi, yvals, axis=1)
   pi_integral = np.trapezoid(pi_integral_y, xvals)

   if not np.isclose(pi_integral, 1.0, rtol=1e-4):
       print(f"[Iter␣{j}]␣Warning:␣        ␣=␣{pi_integral:.6f},␣renormalizing␣  .")


   #normalisation so that pi is considered a PDF
   pi=pi/pi_integral

   #defining our marginals directly from pi
   rho0=np.trapezoid(pi,yvals)
   rho1=np.trapezoid(pi,xvals)

   #normalisation of rho0 and rho1
   rho0_int = np.trapezoid(rho0, xvals)
   rho1_int = np.trapezoid(rho1, yvals)

   rho0=rho0/rho0_int
   rho1=rho1/rho1_int

   #computing KL without division by zero or log zero
   eps = 1e-300

   #computing KL-divergence directly with its formula:

   #computing log ratio without division by zero
   log_ratio = np.log(pi+eps) -np.log(g+eps)
   KL_integrand = pi*log_ratio
```

```
182
183
184     #double integrating to obtain KL divergence
185     KL_inner_int=np.trapezoid(KL_integrand, xvals)
186     KL_total_int=np.trapezoid(KL_inner_int, yvals)
187
188     #adding the value to the list
189     KL.append(KL_total_int)
190
191
192 #Plotting KL-divergence
193 x_numbers = list(range(1, len(KL)+1))
194
195 plt.plot(x_numbers, KL)
196 plt.title("KL-divergence␣per␣iteration")
197 plt.grid(True)
198 #plt.axis((1,len(KL), ymin, ymax))
199 plt.xlabel("Iteration")
200 plt.ylabel("KL-divergence")
201 plt.show()
```

# E

# Code for Gaussian marginals with non-zero mean

The code used to find the numerical solutions to the SB problem with Gaussian marginals with non-zero mean is given below. The algorithm produces functions $\hat{\phi}_0$ & $\phi_1$, and the value of the KL-divergence per iteration, which are plotted as graphs.

```
    # -*- coding: utf-8 -*-
"""
Created on Tue May 13 18:13:58 2025

@author: lunah
"""

#1D Gaussian nonzero meean SBP algorithm
#based on analytical evaluations
#using rho0(x)=1/sqrt(2*pi*sigma_0^2)exp(-(x-mu)^2/2sigma_0^2)
#using rho1(y)=1/sqrt(2*pi*sigma_1^2)exp(-y^2/2sigma_1^2)
#using g(x,y)=q(0,x,1,y)= 1/sqrt(2*pi*sigma^2)exp(-(x-y)^2/2sigma^2)

import math
import numpy as np
import sympy as smp
import matplotlib.pyplot as plt
from sympy.abc import x,y

################################################
#creating fucntions phihat0 and phi1

#initialsing values for the variances and mean
mu=float(input("choose a value for  :"))
sigma_0= float(input("choose a value for _0 :" ))
sigma_1=float(input("choose a value for _1 :" ))
sigma=float(input("choose a value for  :" ))

#initialsing coefficients a,b,c and d,e,f
max_iter=100
a_vals=[]
bfrac_vals=[]
c_vals=[]

d_vals=[0]
efrac_vals=[0]
f_vals=[0]

#defining intial values for the coefficients
a_vals.append(1)
bfrac_vals.append(0)
c_vals.append(0)
```

```

#define tolerance for stopping condition
tolerance=1e-6

#creating for loop for the coefficients
for i in range(1,max_iter):
    efrac_vals.append(1/(sigma_1**2)-1/(sigma**2) +1/(sigma**2 + bfrac_vals[i-1]*(sigma**4)))
    f_vals.append(-(1/(efrac_vals[i])*bfrac_vals[i-1]*c_vals[i-1]/((bfrac_vals[i-1]*sigma**2)
        +1)))

    coeff_d_next = math.sqrt(1 +sigma**2*bfrac_vals[i-1])/(a_vals[i-1]*sigma_1*math.sqrt(2*
        math.pi))
    exp_d_next = efrac_vals[i]*(f_vals[i]**2)/2 -((bfrac_vals[i-1]**2)*c_vals[i-1]**2)/(2*((
        bfrac_vals[i-1])+1/(sigma**2)))+ bfrac_vals[i-1]*(c_vals[i-1]**2)/2
    d_vals.append(coeff_d_next*math.exp(exp_d_next))

    bfrac_vals.append(1/(sigma_0**2)-1/(sigma**2) +1/(sigma**2 + efrac_vals[i]*(sigma**4)))
    c_vals.append((1/bfrac_vals[i])*(mu/(sigma_0**2) - f_vals[i]/((sigma**2)+(1/efrac_vals[i
        ])))))

    coeff_a_next = math.sqrt(1 +sigma**2*efrac_vals[i])/(d_vals[i]*sigma_0*math.sqrt(2*math.
        pi))
    exp_a_next = bfrac_vals[i]*(c_vals[i]**2)/2 -(f_vals[i]**2)/(2*((1/efrac_vals[i])+1/(((
        efrac_vals[i])**2)*(sigma**2))))+ efrac_vals[i]*(f_vals[i]**2)/2 -mu**2/(2*(sigma_0
        **2))
    a_vals.append(coeff_a_next*math.exp(exp_a_next))

    #stopping conditions for convergence:
    REa = abs((a_vals[i]-a_vals[i-1])/max(a_vals[i-1], 1e-10))
    REb = abs((bfrac_vals[i]-bfrac_vals[i-1])/max(bfrac_vals[i-1], 1e-10))
    REc = abs((c_vals[i]-c_vals[i-1])/max(c_vals[i-1], 1e-10))

    REd = abs((d_vals[i]-d_vals[i-1])/max(d_vals[i-1], 1e-10))
    REe = abs((efrac_vals[i]-efrac_vals[i-1])/max(efrac_vals[i-1], 1e-10))
    REf = abs((f_vals[i]-f_vals[i-1])/max(f_vals[i-1], 1e-10))

    if REa < tolerance and REb < tolerance and REc < tolerance and REd < tolerance and REe <
        tolerance and REf < tolerance:
        print(f"Converged␣in␣{i}␣iterations.")
        break



#defining the grid
lb=-5                               #left boundary
rb=5                                #right boundary
N=1001                               #nr of grid points
xvals=np.linspace(lb,rb,N)           #grid for x values
yvals=np.linspace(lb,rb,N)           #grid for y values

#defining phihat0 and phi1 so that it can be plotted
phihat0_xvals=a_vals[-1]*np.exp(-bfrac_vals[-1]*(xvals-c_vals[-1])**2/2)
phi1_yvals = d_vals[-1]*np.exp(-efrac_vals[-1]*(yvals-f_vals[-1])**2/2)

#rewriting phihat0 and phi1 back into symoblic expression approx. in exponential form

#first make sure log(0) does not happen
phihat0_clipped = np.clip(phihat0_xvals, 1e-10, None)
log_phihat0 = np.log(phihat0_clipped)

phi1_clipped=np.clip(phi1_yvals, 1e-10, None)
log_phi1 = np.log(phi1_clipped)

#fit an polynomial of degree 2  (for ex.) to the log of the function
coeffs0=np.polyfit(xvals, log_phihat0, deg=2)
coeffs1=np.polyfit(yvals, log_phi1, deg=2)

#rounding and thresholding the coefficients:
rounded_coeffs0 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs0]
rounded_coeffs1 = [round(c, 4) if abs(c) >= 1e-5 else 0 for c in coeffs1]
```

```
106
107 #creating a symbolic expression in an approximation of an exponential
108 phihat0_sym=sum(c*x**i for i, c in enumerate(reversed(rounded_coeffs0)) if c!=0)
109 phihat0_approx=smp.exp(phihat0_sym)
110
111 phi1_sym=sum(c*y**i for i, c in enumerate(reversed(rounded_coeffs1)) if c!= 0)
112 phi1_approx=smp.exp(phi1_sym)
113
114 print("\n␣phihat0(x)␣=␣\n", phihat0_approx,
115       "\n␣phi1(y)␣=␣\n", phi1_approx)
116
117 #plotting phihat0x and phi1y
118 plt.plot(xvals, phihat0_xvals, label="$\hat\phi_0(x)$", color='blue')
119 plt.plot(yvals, phi1_yvals, label=r'$\phi_1(y)$', color='red', linestyle='--')
120 plt.title("Functions␣$\hat\phi_0(x)$␣and␣$\phi_1(y)$")
121 #plt.axis((-10,10,0,4))
122 plt.legend()
123 plt.grid(True)
124 plt.show()
125
126
127 ##################################################
128 #Sanity check that P01(R) =1
129 q_0=1/(1/(sigma**2)+efrac_vals[-1]-1/((sigma**2)+(sigma**4)*bfrac_vals[-1]))
130 q_1=q_0*(f_vals[-1]*efrac_vals[-1] + bfrac_vals[-1]*c_vals[-1]/(1+ (sigma**2)*bfrac_vals[-1])
    )
131
132 P_coeff = math.sqrt(2*math.pi*q_0)*a_vals[-1]*d_vals[-1]/(math.sqrt(1 + (sigma**2)*bfrac_vals
    [-1]))
133 P_exp = q_1**2/(2*q_0) + (bfrac_vals[-1]**2)*(c_vals[-1]**2)/(2*(1/(sigma**2)+bfrac_vals[-1])
    ) -efrac_vals[-1]*(f_vals[-1]**2)/2 -bfrac_vals[-1]*(c_vals[-1]**2)/2
134
135 P01_R=P_coeff*math.exp(P_exp)
136
137 ##################################################
138 #KL-divergence
139
140 # initialising KL-divergence
141 KL=[]
142
143 #2D grid for x and y
144 X,Y = np.meshgrid(xvals, yvals, indexing ='ij')
145
146 #defining g as a function of x and y:
147 g_coeff = 1/(sigma*np.sqrt(2*np.pi))
148 g_exp=-(X-Y)**2/(2*sigma**2)
149 g=g_coeff*np.exp(g_exp)
150
151 #normalisation so that g is considered a PDF
152 g_integral=np.trapezoid(np.trapezoid(g,yvals, axis=1),xvals)
153 g=g/g_integral
154
155 #creating function pi for each iteration
156 for j in range(1,len(a_vals)):
157
158     #defining phihat0(x) and phi1(y) for each iteration
159     phihat0_per_it = a_vals[j]*np.exp(-bfrac_vals[j]*(xvals-c_vals[j])**2/2)
160     phi1_per_it = d_vals[j]*np.exp(-efrac_vals[j]*(yvals-f_vals[j])**2/2)
161
162     #normalization of phihat0 and phi1
163     Z = np.trapezoid(
164         np.trapezoid(phihat0_per_it* g * phi1_per_it, yvals, axis=1),
165         xvals)
166
167     phihat0_per_it /= np.sqrt(Z)
168     phi1_per_it /= np.sqrt(Z)
169
170     pi_log=np.log(g)+np.log(phihat0_per_it)+np.log(phi1_per_it)
171     pi=np.exp(pi_log)
172
173     # Normalize    so that it integrates to 1
```

```
    pi_integral_y = np.trapezoid(pi, yvals, axis=1)
    pi_integral = np.trapezoid(pi_integral_y, xvals)


    if not np.isclose(pi_integral, 1.0, rtol=1e-4):
        print(f"[Iter {j}] Warning:         = {pi_integral:.6f}, renormalizing   .")

    #normalisation so that pi is considered a PDF
    pi=pi/pi_integral


    #computing KL-divergence directly with its formula:
    #computing log ratio without division by zero
    eps = 1e-300
    log_ratio = np.log(pi+eps) -np.log(g+eps)
    KL_integrand = pi*log_ratio


    #double integrating to obtain KL divergence
    KL_inner_int=np.trapezoid(KL_integrand, yvals, axis =1)
    KL_total_int=np.trapezoid(KL_inner_int, xvals)

    #adding the value to the list
    KL.append(KL_total_int)


#Plotting KL-divergence
x_numbers = list(range(1, len(KL)+1))
plt.plot(x_numbers, KL)
plt.title("KL-divergence per iteration")
plt.grid(True)
#plt.axis((1,len(KL), 0, 5))
plt.xlabel("Iteration")
plt.ylabel("KL-divergence")
plt.show()
```