Fast Deterministic Sampling of Gaussian Mixture Densities using Projected Cumulative Distributions

Dominik Prossel, **Zhilun Li**, **Petr Novikov**, and **Uwe D. Hanebeck**
Intelligent Sensor-Actuator-Systems Laboratory (ISAS)
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT), Germany
dominik.prossel@kit.edu, zhilun.li@keemail.me, uhnno@student.kit.edu, uwe.hanebeck@kit.edu

***Abstract*—Algorithms based on Projected Cumulative Distributions in combination with a suitable one-dimensional distance measure have been shown to generate high-quality samples from arbitrary densities. This paper investigates their suitability for deterministic sampling from Gaussian mixture densities that arise in a wide range of applications, from nonlinear filtering to model-predictive control. We propose an improved optimization procedure that increases sample quality. To speed up calculation of gradient and Hessian during optimization, we introduce an approximation scheme based on lookup tables. By employing multithreading and GPU-acceleration we are able to take advantage of the highly parallel nature of the algorithm, resulting in a further decrease of its runtime.**

***Index Terms*—Gaussian mixtures, Deterministic sampling, Projected distance, Parallelization.**

I. Introduction

Hybrid filters: Recursive Bayesian estimation of nonlinear dynamical systems generally requires approximation techniques, such as statistical linearization of the system and measurement equation. These approaches perform well for weakly nonlinear systems, but can result in large estimation errors when nonlinearities are strong. Particle filters address this limitation by using a sample-based representation of the state distribution. Particles can be propagated through nonlinear system equations simply by evaluating the system function. However, their filter step can be very challenging relying on resampling to deal with particle degeneration.

These considerations motivate a special class of filtering algorithms. They combine the prediction step of a particle filter to handle strongly nonlinear system equations with the update step of a nonlinear Kalman filter to efficiently incorporate weakly nonlinear measurement information. These hybrid filters continually and alternatingly convert between sample-based and continuous density representations, in order to exploit their respective strengths.

Gaussian Mixture Filters: Filters using Gaussian mixture (GM) densities as their continuous representation are an especially interesting case of these hybrid filters. GMs can approximate any density arbitrarily well by increasing the number of components and can be filtered rather easily on the basis of the Kalman filter and its extensions for nonlinear measurement equations. Some recent examples of GM-based hybrid filters are the Ensemble GM filter introduced in [1] and the one proposed in [2]. Another related field can be found in stochastic model-predictive control, where a GM can occur as the result of the propagation of samples through a Bayesian neural network [3]. This GM is then converted to a simpler representation for further processing, for example, by drawing samples from it. In these methods, the other direction that goes from samples to GMs occurs by design in the prediction step or is performed by density estimation.

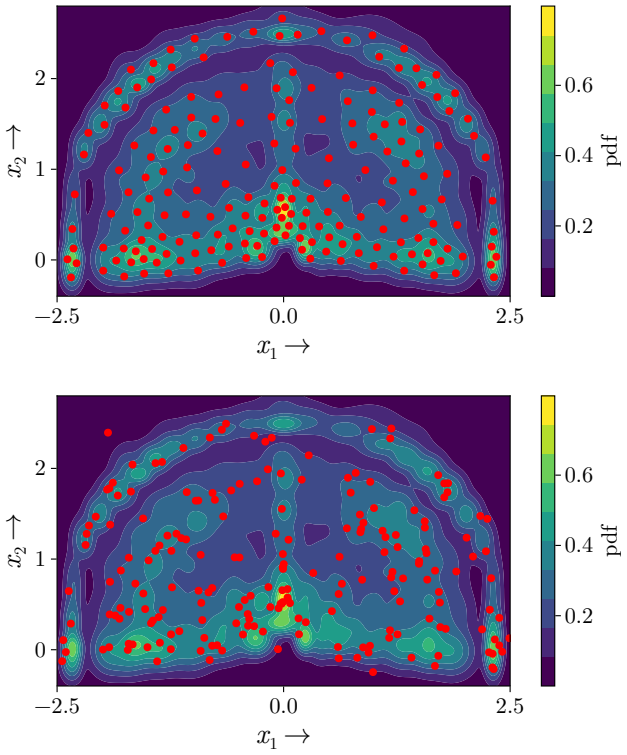


Fig. 1: 200 deterministic samples drawn with the proposed method (top) and 200 random samples (bottom) of a Gaussian mixture with 100 components.

Sampling from a GM: Samples can be drawn from a GM using random sampling or deterministic sampling approaches. Random samples are easy to generate, but many samples are needed to approximate the underlying continuous distribution well. They are best used when the computations performed on

each sample are inexpensive and there is little drawback to using many samples.

Deterministic sample sets are designed to reduce approximation errors with respect to the underlying distribution. Therefore, fewer deterministic samples are needed to achieve the same accuracy as with random sampling [4]. Fig. 1 compares 200 deterministic samples from a GM with 200 randomly drawn samples. The deterministic samples provide an even coverage of the space according to the underlying probability density function (PDF), whereas the random samples exhibit noticeable clustering and gaps. However, generating optimally placed deterministic samples of a GM, as in [5], is generally computationally expensive. They are especially advantageous when each additional sample has a significant impact on the total runtime of an algorithm, for example, in sample-based model-predictive control, where expensive simulations are required for each sample.

To avoid the need for repeated generation of deterministic samples, [6] proposed to use a sample cache, which stores generated samples and reuses them when samples of the same density are requested again. This approach performs well for single Gaussian densities. GMs have many more degrees of freedom, significantly reducing the chance of needing samples of the same GM twice, which makes caching very inefficient. With offline generation and caching out of the question, a fast method for online generation of deterministic samples from GMs is needed. A very promising approach is optimization based on Projected Cumulative Distributions (PCDs), which was first introduced in [7]. It leads to a highly parallelizable algorithm that is suitable for acceleration with multi-threading and GPUs. It also enables a trade-off between accuracy and speed by tuning easily interpretable parameters.

Contributions: This paper is concerned with fast generation of deterministic samples from GM densities based on PCDs. An optimization algorithm using Newton's method is proposed that improves the original algorithm presented in [7] by increasing the accuracy of the density approximation. Several methods are proposed to speed up the evaluation of the gradient and the Hessian during optimization. This includes a sorting-based scheme for fast evaluation of the cumulative distribution function (CDF) of a one-dimensional Dirac mixture (DM) distribution, as well as the introduction of lookup tables to quickly evaluate the CDF and PDF of one-dimensional GM distributions approximately. The resulting algorithm is analyzed with regard to parallelization, and the speedup achieved by multi-threading as well as GPU-acceleration is shown. The implementation used in the experiments is available on GitHub[1].

Outline: In Section II, a short overview of sampling methods for GM distributions is given. The necessary background on PCD-based sampling is given in Section III, though more detailed derivations are available in [7]. The improved optimization method and faster evaluation schemes are presented in Section III-C and Section IV. In Section V, the potential for parallelization of the algorithm is investigated.

II. Related Work

Drawing a random sample from a GM can be done in a two-step procedure by first selecting a component based on the component weights and then drawing a sample from this Gaussian. Depending on the shape of the GM, many random samples are needed to completely cover the underlying distribution and provide an accurate representation. An approach that guarantees that samples are more evenly distributed throughout the density is based on sample reduction. The idea is to first draw samples from each component separately, so there are overall more samples than the final number of samples desired. This may be done either randomly or deterministically. The drawn samples are then reduced to the desired number with a suitable algorithm. For example, with the method based on Localized Cumulative Distribution (LCD) introduced in [8] or with methods from optimal transport [9]. A comparison of different combinations of sampling and reduction algorithms and sampling based on PCDs is available in [1].

When using random samples, the reduction step becomes rather expensive, as there need to be enough random samples to begin with. Choosing this initial number of samples is also a hyperparameter that is challenging to determine automatically. Starting with deterministic samples from each component, the problem lies in the fact that they are drawn independently for each one and then combined into a single set of samples. This leads to artifacts in areas where components overlap, giving suboptimal samples. A way of generating optimal samples of a GM is presented in [5]. They extended the method in [10] for sampling single Gaussians to GM. The samples are placed by minimizing a distance measure based on LCDs. Calculating this distance measure and its gradient requires numerical integration and pairwise distance calculation, slowing down the computation and making it unsuitable for online sample generation. However, it can serve to generate reference solutions for comparison with other algorithms. Sliced optimal transport sampling [11] is a method similar to PCD-based sampling. It uses the sliced Wasserstein loss instead of the Cramér–von Mises-type objective function in [7]. Good results are achieved for sampling uniform densities on different manifolds, but it has not yet been extended to GM.

III. Sampling based on Projected Cumulative Distributions

Sampling of continuous probability distributions based on PCDs was first introduced in [7]. It has also shown good results on non-Euclidean manifolds such as the circle [12]. The key concept behind this method is the Radon transform [13], which transforms a high-dimensional density into infinitely many one-dimensional densities by projecting it onto all possible one-dimensional subspaces. It is applied to both the continuous density and the DM density representing the samples. In practice, only a finite number K of projections is used for the transformation instead of all infinitely many projections. A distance measure between the two densities is defined that accumulates the differences in each of the one-dimensional projections. These differences are measured with a Cramér–von

Mises-type distance measure for one-dimensional distributions. This distance measure is subsequently minimized to find the sample locations.

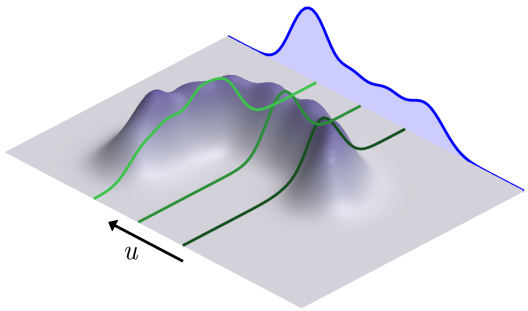


Fig. 2: Different slices of a PDF in green and its projection in blue. The Radon transform can be thought of as taking all slices orthogonal to $\underline{u}$ and integrating over each slice to get the desired projection.

A. Relation to Sliced Distances

A closely related family of distance measures are so-called "sliced" distances. These are especially popular in machine learning with the sliced Wasserstein distance as their most prominent member [14], [15]. The term "sliced" in the name of these methods can be slightly misleading, as they are not based on slices of densities, but rather on projections. The name might come from an interpretation of the Radon transform depicted in Fig. 2. Given a D-dimensional unit vector $\underline{u}$ and a PDF $f(\underline{x})$, it can be thought of as first taking all slices through the PDF orthogonal to $\underline{u}$. This is equivalent to conditioning it on $\underline{u}^\top \underline{x} = z$ for $z \in \mathbb{R}$. Three of these slices for three different values of z are drawn in green in Fig. 2. Integrating over all slices then gives the desired projection $f(z)$ shown in blue

$$f(z) = \int_{\mathbb{R}^D} f(\underline{x} | \underline{u}^\top \underline{x} = z) \tag{1}$$

This is essentially a marginalization of $f(\underline{x})$ with respect to all vectors orthogonal to $\underline{u}$. Note that this transformation only depends on the direction of the unit vector and does not have an anchor point, whereas actual slicing would require one, to define which slice along $\underline{u}$ is taken.

B. Optimization Setup

Consider a continuous D-dimensional distribution with PDF $f(\underline{x})$ and $\underline{x} \in \mathbb{R}^D$ as the target to be sampled. In case this is a GM with C components, the normalization factors $\eta_1, \dots, \eta_C$, and positive weights $v_1, \dots, v_C$ summing to one, this PDF is

$$f(\underline{x}) = \sum_{i=1}^{C} v_i \eta_i \exp\left(-\frac{1}{2}(\underline{x} - \underline{m}_i)^\top \mathbf{C}_i^{-1} (\underline{x} - \underline{m}_i)\right) \ . \tag{2}$$

The goal of optimal sampling is to find the N sample locations collected in $\hat{\underline{X}} = [\hat{\underline{x}}_1^\top, \dots, \hat{\underline{x}}_N^\top]^\top$ that best approximate the target density. To this end, the samples are interpreted as a DM density with PDF

$$\tilde{f}(\underline{x}, \hat{\underline{X}}) = \sum_{i=1}^{N} w_i \delta(\underline{x} - \hat{\underline{x}}_i) \tag{3}$$

with weights $w, \dots, w_N$ that are positive and sum to one.

To compute PCDs of the target and DM density, K unit vectors $\mathcal{U} = \{\underline{u}_1, \dots \underline{u}_K\}$ are uniformly sampled from the hypersphere $\mathbb{S}^{D-1}$ embedded in D-dimensional space. They are used as projection directions. Both densities are projected onto each of the unit vectors, and the one-dimensional CDFs of the projections are computed. The projection of a GM is a one-dimensional GM with projected means $m_{k,i} = \underline{u}_k^\top \underline{m}_i$ and variances $\sigma_{k,i}^2 = \underline{u}_k^\top \mathbf{C}_i \underline{u}_k$, resulting in the PDF

$$f_k(r) = \sum_{i=1}^{C} v_i \frac{1}{\sqrt{2\pi}\sigma_{k_i}} \exp\left(-\frac{1}{2}\frac{(r - m_{k,i})^2}{\sigma_{k,i}^2}\right) \ . \tag{4}$$

Using the complementary error function $\operatorname{erfc}(x)$ the corresponding CDF is

$$F_k(r) = \frac{1}{2} \sum_{i=1}^{C} v_i \operatorname{erfc}\left(\frac{m_{k,i} - r}{\sigma_{k,i}\sqrt{2}}\right) \ . \tag{5}$$

Projecting the DM $\tilde{f}(\underline{x}, \hat{\underline{X}})$ gives the one-dimensional DM $\tilde{f}(r, \hat{\underline{r}}_k)$ with projected locations $\hat{\underline{r}}_k = [\hat{r}_{k,1}, \dots, \hat{r}_{k,N}]^\top$, where $\hat{r}_{k,i} = \underline{u}_k^\top \hat{\underline{x}}_i$. Its CDF is

$$\tilde{F}_k(r, \hat{\underline{r}}_k) = \sum_{i=1}^{N} w_i H(r - \hat{r}_{k,i}) \tag{6}$$

with the Heaviside step function

$$H(x) = \begin{cases} 0, & \text{if } x < 0 \\ 0.5, & \text{if } x = 0 \\ 1, & \text{if } x > 0 \end{cases} \ . \tag{7}$$

See [7] for a detailed derivation of these projections.

Based on the PCDs of both densities, the *one-dimensional* Cramér–von Mises distance with weighting function $1/f_k(r)$

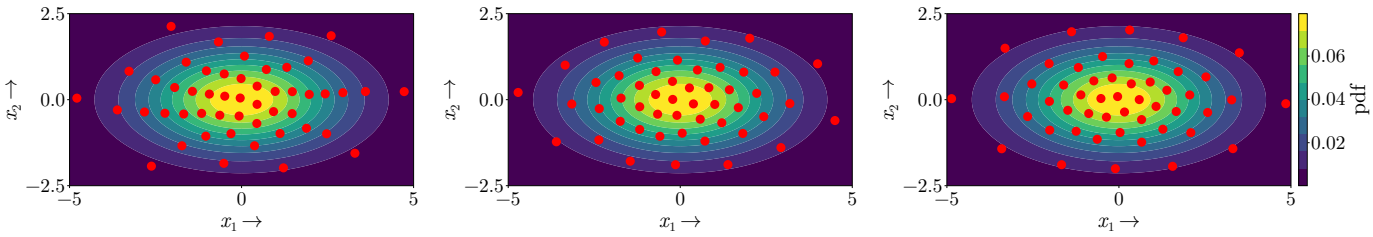
$$d_k(\hat{\underline{r}}_k) = \int_{-\infty}^{\infty} \left(\tilde{F}_k(r, \hat{\underline{r}}_k) - F_k(r)\right)^2 \mathrm{d}r \tag{8}$$

can be computed in each projection. For the final distance measure, all the one-dimensional distances are accumulated to give

$$D_{\mathrm{PCD}}(\hat{\underline{X}}) = \frac{1}{K} \sum_{k=1}^{K} d_k\left(\hat{\underline{r}}_k(\hat{\underline{X}})\right) \ . \tag{9}$$

The optimal sample locations $\hat{\underline{X}}^*$ are then found by minimization as

$$\hat{\underline{X}}^* = \arg\min_{\hat{\underline{X}}} D_{\mathrm{PCD}}(\hat{\underline{X}}) \ . \tag{10}$$



(a) Original [7] (b) Proposed (c) LCD [10]

Fig. 3: Comparison of different optimization methods, for drawing 50 samples of an axis-aligned Gaussian with variance $c_{11} = 4$ along x_1 and $c_{22} = 1$ along x_2.

C. Improved Optimization Procedure

In the original paper [7], an algorithm was proposed to solve the minimization problem in (10). The first step of the algorithm is to draw K projection directions that are fixed during optimization and set an initial solution. Then quasi-Newton updates are applied iteratively while the change in solution is greater than some convergence criterion ϵ. This update is calculated by computing the Newton step in each one-dimensional projection and then backprojecting and accumulating all one-dimensional updates into one update in the original space.

This method already gives very good results shown in Fig. 3, where 50 samples were drawn from a Gaussian density with zero mean and diagonal covariance matrix with entries $c_{11} = 4$ and $c_{22} = 1$. Compared to the optimal LCD-samples drawn with the method in [10], the samples from [7] do not cover the density as evenly and seem to align along radial lines. This effect is more pronounced for distributions that exhibit high anisotropy.

To address this issue, we propose to use Newton's method instead of the quasi-Newton update. To apply the update step in Newton's method

$$\underline{\hat{X}}_{t+1} = \underline{\hat{X}}_t - \mathbf{H}^{-1}\underline{g} \tag{11}$$

the gradient $\underline{g}$ and Hessian $\mathbf{H}$ of the distance measure in (9) with respect to the sample locations are needed. The gradient with respect to the sample location $\underline{\hat{x}}_i$ can be written as

$$\underline{g}_i = \frac{\partial D_{\text{PCD}}(\underline{\hat{X}})}{\partial \underline{\hat{x}}_i} = \frac{1}{K}\sum_{k=1}^{K} \frac{\partial d_k(\underline{\hat{r}}_k)}{\partial \hat{r}_{k,i}} \frac{\partial \hat{r}_{k,i}}{\partial \underline{\hat{x}}_i} \ . \tag{12}$$

With the derivatives of the one-dimensional distance measure derived in [7] and $\partial \hat{r}_{k,i} / \partial \underline{\hat{x}}_i = \underline{u}_k$ this is

$$\underline{g}_i = \frac{2w_i}{K}\sum_{k=1}^{K} \left(\tilde{F}_k(\hat{r}_{k,i}, \underline{\hat{r}}_k) - F_k(\hat{r}_{k,i})\right) \underline{u}_k \ . \tag{13}$$

These partial derivatives are then stacked into a single vector to give the complete gradient $\underline{g} = [\underline{g}_1^\top, \ldots, \underline{g}_N^\top]^\top$.

The Hessian matrix has a block-diagonal structure

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 & & \\ & \ddots & \\ & & \mathbf{H}_N \end{bmatrix} \tag{14}$$

with entries

$$\mathbf{H}_i = -\frac{2w_i}{K}\sum_{k=1}^{K} f_k(\hat{r}_{k,i})\, \underline{u}_k \underline{u}_k^\top \ . \tag{15}$$

This structure of the Hessian makes the Newton step especially easy to apply, as it can be solved independently for each sample. Instead of explicitly inverting each block $\mathbf{H}_i$, it is often more efficient and numerically stable to directly solve the linear system of equations $\mathbf{H}_i \underline{\delta}_i = \underline{g}_i$ for the update $\underline{\delta}_i$. This results in as many D-dimensional linear systems of equations as there are samples.

Comparing the results of this optimization procedure with the original one in Fig. 3, the samples are much more evenly spaced and look very similar to the optimal samples produced by the LCD-based optimization. This is also reflected in the LCD-distance that is used as the loss function in [10]. For the LCD-samples it is approximately 5.8×10^{-4}, for the original PCD-method 8.1×10^{-4}, and the proposed method lies in the middle with 7.1×10^{-4}. The only drawback of this new procedure compared to [7] is that a series of linear systems of equations needs to be solved for the Newton step, which was not the case in the original method.

IV. Efficient Implementation

A naive implementation of our algorithm would first draw a set of projection directions, set an initial solution, and then enter into an optimization loop. In each iteration, it would compute the projections of the densities and evaluate and accumulate the gradient and Hessian in (13) and (14). It then applies the Newton update by solving linear systems of equations and checks for convergence before repeating the loop. Looking closely, there are some simplifications and approximations that can be made to significantly speed up the procedure while maintaining a high level of accuracy of the solution.

The first thing to note is that the projection directions and the target density do not change during optimization. It is therefore not necessary to recompute the projections of the target density in each iteration. This can be done once as part of the initialization and saved for later use. For the Newton update two more minor optimizations are possible. The first one is to cancel the factor $2w_i/K$ in the gradient and Hessian. The second one is to use a solver specialized for symmetric matrices to solve the linear systems of equations. Note that $\mathbf{H}_i$

is a weighted sum of the symmetric rank-one matrices $\underline{u}_k \underline{u}_k^\top$ weighted with $f(\hat{r}_{k,i})$.

The main computational cost of the algorithm lies in the computation of the gradient and Hessian in each iteration, as that requires the evaluation of $\tilde{F}_k(r, \underline{\hat{r}}_k)$, $F_k(r)$ and $f_k(r)$ for each projection and for each sample location $\hat{r}_{k,i}$. This results in NK evaluations per iteration, where each evaluation itself is relatively expensive when done naively.

A. Evaluation of the CDF of DM Densities

The straightforward way to evaluate the CDF of the projected DM $\tilde{F}_k(r, \underline{\hat{r}}_k)$ at the projected sample location $\hat{r}_{k,j}$ is to just calculate the sum in (6). This requires N evaluations of the Heaviside function. While each evaluation is inexpensive, repeating the computation for every projected sample location leads to quadratic scaling in the number of samples. By sorting the pairs of projected samples and weights by position in ascending order and calculating the cumulative sum of the weights, it is possible to evaluate (6) in constant time by directly obtaining the value of $\tilde{F}_k(\hat{r}_{k,j}, \underline{\hat{r}}_k)$ from the cumulative sum.

In the common case that all samples are assigned the same weight of $1/N$, it is enough to have the rank $s_{k,j}$ of a sample $\hat{r}_{k,j}$ to evaluate $\tilde{F}_k(\hat{r}_{k,j}, \underline{\hat{r}}_k)$ as

$$\tilde{F}_k(\hat{r}_{k,j}, \underline{\hat{r}}_k) = (s_{k,j} - 0.5)/N \ . \tag{16}$$

This scheme requires a sorting operation in $\mathcal{O}(N \log(N))$ followed by calculation of (16) for each sample in $\mathcal{O}(N)$. Overall, the quadratic scaling of the naive method is reduced to loglinear scaling with sorting dominating the cost.

Another important observation regarding sorting of the projected samples is that after a few iterations of the algorithm the sample positions do not change much anymore. By saving the ranks of projected samples for each projection for the next iteration, the sorting algorithm can then start with a nearly sorted list of samples. For such lists, simple algorithms like bubble sort or insertion sort can be much faster than divide and conquer algorithms that always sort the complete list [16].

B. Fast Evaluation for GM Densities

Evaluation of $f_k(r)$ and $F_k(r)$ involves the calculation of the exponential function and the error function for each component of the mixture. Compared to, for example, simple polynomials, these are typically expensive to compute. This is especially true for the error function, as there is no closed form solution available. The two basic approaches to accelerate this part of the algorithm are to speed up the computation of these two functions or to reduce the number of times they are computed. This paper takes the second approach by computing lookup tables for $F_k(r)$ and $f_k(r)$ for each projection in the initialization phase of the algorithm.

To create the lookup table, the functions are evaluated at a fixed number M of equally spaced grid points $\xi_1, \ldots, \xi_M$. These are determined by finding the first and last point based on the means and standard deviations of the projected densities as

$$\xi_1 = \min(m_j - 3\sigma_j) \ \forall j = 1, \ldots, C \ , \tag{17}$$

$$\xi_M = \max(m_j + 3\sigma_j) \ \forall j = 1, \ldots, C \ . \tag{18}$$

The remaining points are then placed between the border points, spaced by step size

$$h = \frac{(\xi_M - \xi_1)}{(M-1)} \ . \tag{19}$$

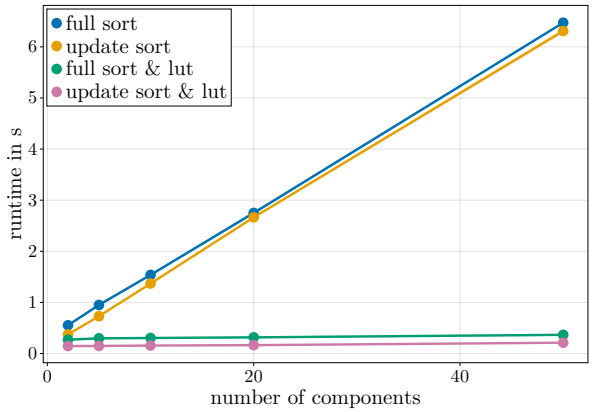
The function values at the grid points are stored as $L_i = F(\xi_i)$ and $l_i = f(\xi_i)$. During optimization, this enables approximate evaluation of these functions in constant time by linear interpolation. If the evaluation point x lies outside the interval covered by the lookup table, 0 is returned for the PDF and 0 or 1 for the CDF, depending on if x is smaller than ξ_1 or bigger than ξ_M. Otherwise, an index $i(x)$ is computed such that $\xi_{i(x)} < x \leq \xi_{i(x)+1}$

$$i(x) = \left\lceil \frac{(x - \xi_1)}{h} \right\rceil \ . \tag{20}$$

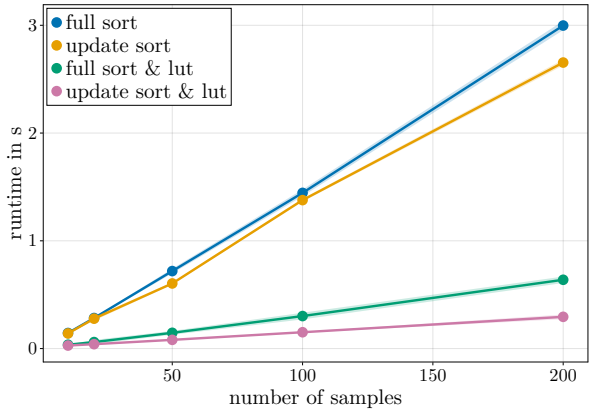
Algorithm 1: Proposed algorithm for sampling equally weighted samples from a Gaussian mixture. The function rank($\underline{\hat{r}}$) computes the rank of each entry in $\underline{\hat{r}}$.

input : Initial solution $\underline{\hat{X}}$, target distribution $f(\underline{x})$, projection directions $\mathcal{U} = \{\underline{u}_1, \ldots \underline{u}_K\}$ and convergence criterion ϵ

$\underline{\delta} \leftarrow \underline{0}$
for $\underline{u}_k \in \mathcal{U}$ **do**
  Compute $f_k(r, \underline{\hat{r}}_k)$ and $F_k(r)$
  Compute lookup tables L_k and l_k
end
while $\|\underline{\delta}\| > \epsilon$ **do**
  $\underline{\delta} \leftarrow \underline{0}$
  for $\underline{u}_k \in \mathcal{U}$ **do**
    $\underline{\hat{r}}_k = [\underline{u}_k^\top \underline{\hat{x}}_1, \ldots, \underline{u}_k^\top \underline{\hat{x}}_N]^\top$
    $\underline{s}_k = \text{rank}(\underline{\hat{r}}_k)$
  end
  for $\underline{x}_i \in \underline{\hat{X}}$ **do**
    $\underline{g} \leftarrow \underline{0}$
    $\mathbf{H} \leftarrow \mathbf{0}$
    for $\underline{u}_k \in \mathcal{U}$ **do**
      $\underline{g} \leftarrow \underline{g} + \left((\underline{s}_{k,i} - 0.5)/N - L_k(\hat{r}_{k,i})\right) \underline{u}_k$
      $\mathbf{H} \leftarrow \mathbf{H} + l_k(\hat{r}_{k,i}) \, \underline{u}_k \underline{u}_k^\top$
    end
    $\underline{\delta}_i \leftarrow \mathbf{H}^{-1} \underline{g}$
    $\underline{\hat{x}}_i \leftarrow \underline{\hat{x}}_i + \underline{\delta}_i$
  end
end
return $\underline{\hat{X}}$



(a) Runtime scaling with the number of GM components



(b) Runtime scaling with the number of samples

Fig. 4: Comparison of the runtime of our algorithm with different techniques for faster gradient evaluation. To evaluate $\tilde{F}(r, \hat{\underline{r}}_k)$, it is either naively sorting the samples (full sort) in each iteration or using some bookkeeping to start with a nearly sorted list (update sort). It is also using or not using lookup tables (lut) to evaluate $f_k(r)$ and $F_k(r)$.

An approximate value of $F(x)$ is then linearly interpolated between the values L_i at $\xi_{i(x)}$ and L_{i+1} at $\xi_{i(x)+1}$

$$F(x) \approx L(x) = L_i + \frac{L_{i+1} - L_i}{h}(x - \xi_i) \ . \tag{21}$$

This works analogously for the PDF.

The setup of all lookup tables for all projections requires KCM evaluations of error functions and exponential functions. During optimization the cost is cut down from KCN evaluations of these functions to KN evaluations of the lookup tables, entirely eliminating the dependency on the number of components in the Gaussian mixture. With respect to memory consumption, the original algorithm needs space for $2C$ values per projection to store mean and variance of each projected Gaussian component. The lookup table version only requires storage of $2M$ values per projection for the CDF and PDF values at all M grid points.

Pseudocode of the final algorithm to draw equally weighted samples with all improvements is shown in Algorithm 1.

V. Parallelization

The algorithm in Algorithm 1 provides several parallelization opportunities. In the setup phase of the algorithm (lines 2–5), the projection of the target density and the computation of the lookup tables can be performed in parallel for each projection vector. During creation of the lookup tables the PDFs and CDFs can also be evaluated simultaneously at all M grid points.

In the optimization loop starting at line 6 the first step is to project the samples. This requires the computation of KN projected samples, which can be fully parallelized. Calculating the ranks of the projected samples can be performed independently in each projection, resulting in K sorting operations in $\mathcal{O}(N \log(N))$. On a GPU this can be further accelerated with parallel sorting algorithms like bitonic sort, but still remains a significant bottleneck. The evaluation of the CDFs needed in line 16 and PDF in line 17 can then again be done in parallel for each of the KN projected samples locations. After multiplying them with the vector $\underline{u}_k$ or respectively $\underline{u}_k \underline{u}_k^\top$ these values are accumulated over all projections, requiring a reduction operation. Due to the block-diagonal structure of the Hessian in (14) the Newton step can be computed independently for each high-dimensional sample in $\mathcal{O}(D^3)$. The resulting Newton updates can also be applied in parallel to each sample.

VI. Numerical Simulations

A Julia implementation of the optimization algorithm in [7] and of our proposed algorithm is available on GitHub[1]. The repository includes multithreaded and GPU-accelerated code for both algorithms based on Julia 1.12 [17] and *CUDA.jl* [18].

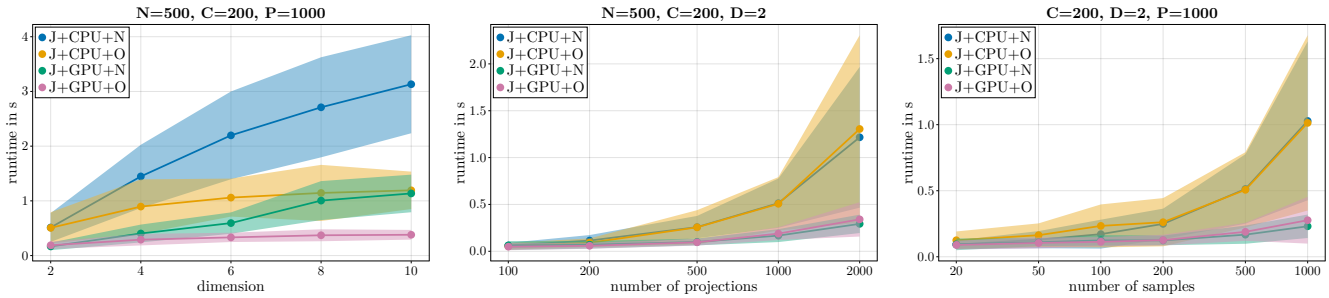
A Python implementation of [7] is also provided[2]. It is implemented using *PyTorch* [19] and can optionally be compiled into multithreaded CPU code or GPU-accelerated code. It does not use most of the techniques discussed in Section IV, and only employs sorting for fast evaluation of $\tilde{F}(r, \hat{r}_k)$.

In the evaluation, we will refer to the different implementations and algorithms by the shorthands "P" for Python and "J" for Julia combined with either "CPU" or "GPU" and "O" for the original algorithm in [7] or "N" for the new optimization procedure described in Section III-C. As an example "J+GPU+N" is short for the GPU-accelerated Julia implementation of the proposed algorithm.

All experiments were conducted inside a Docker container running on an Ubuntu 22.04 host with an Intel Core i9-10980XE CPU with 3.0 GHz base frequency, 18 physical cores and 36 threads. The GPU code was run on an Nvidia GeForce RTX 2080 Ti on the same machine.

All computations used double-precision 64-bit floating point numbers. The number of grid points for the lookup tables was set to $M = 100$, as this provided sufficient accuracy in the scenarios that were tested. A qualitative result of samples drawn with the Julia implementation is shown in Fig. 1.

[1] https://github.com/KIT-ISAS/PCDSampling.jl
[2] https://github.com/KIT-ISAS/PCD_sampling_py



(a) Varying dimension (b) Varying number of projections (c) Varying number of samples

Fig. 5: Average and standard deviation of runtime until reaching the convergence criterion using the different Julia implementations. Each measurement was repeated 40 times for different target densities.

For runtime measurements, the target GM for each run was generated by randomly drawing the mean and standard deviations of each component from a standard normal distribution. The components in these tests only had diagonal covariance matrices, although the implementation can also handle dense matrices.

A. Faster Evaluation

To investigate the speedup achieved with the techniques described in Sections IV-A and IV-B, "J+CPU+N" was run for 100 iterations on a two-dimensional example on a single thread with different combinations of the proposed methods to speed up the evaluation of the gradient. It used lookup tables or (4) and (5) to evaluate $f_k(r)$ and $F_k(r)$. To evaluate $\tilde{F}_k(r, \hat{\underline{r}}_k)$ it sorted the projected samples from scratch in each iteration or saved the ordering from the previous iteration and updated that, starting with a list of nearly sorted values each time.

The number of projections was fixed to $K = 400$. Runtime measurements were repeated 10 times and averaged. The runtime of the different variants for a fixed number of samples $N = 100$ and a varying number of GM components is shown in Fig. 4a. Fig. 4b shows the average runtime for a fixed number of components of $C = 10$ and a varying number of samples.

As expected, the algorithm variants without lookup tables scale linearly in the number of components, because they repeatedly evaluate (4) and (5) in each iteration. The lookup table versions have almost constant runtime as the time spent on the initialization of the lookup tables can be neglected in comparison to the time spent in the optimization loop. The experiments also suggest that keeping track of the ordering of samples over iterations to start sorting with nearly sorted lists speeds up the algorithm, especially for a larger number of samples. When applying these techniques to the algorithm in [7], similar improvements to runtime can be expected.

B. Optimization Procedures

To compare the two different algorithms "N" and "O" and the GPU and CPU parallelization, the Julia versions were run for different problem sizes varying the dimension D, the number of projections K, and the number of samples N. The number of GM components was fixed to $C = 200$, as it does not have a significant impact on runtime, when using lookup tables. The algorithm terminated when $\|\underline{\delta}\|_\infty < 1 \times 10^{-3}$ in each run and 40 runs were executed for each problem size. Fig. 5 shows the average runtime and its standard deviation, keeping two of the parameters fixed and varying the third one.

Increasing the number of dimensions in Fig. 5a shows a big difference between "O" and "N". The original algorithm ("O") does not have a dependence on dimension besides the initial projection of the target density and the projection of the samples in each iteration. Therefore, its runtime increases only slightly for higher dimensions. The improved algorithm ("N") on the other hand needs to solve N D-dimensional linear systems of equations in each iteration. This adds a significant runtime penalty to each iteration in high dimensions but can lead to more accurate results, as shown in Fig. 3. Additional profiling revealed that this is also the bottleneck in the current implementation, taking up a significant portion of the runtime in each iteration.

When the number of samples or the number of projections is increased, the number of iterations until convergence can vary widely depending on the target density and initial solution. Both "O" and "N" seem to scale similarly and also have similar runtimes in the two-dimensional case. In higher dimensions "O" is expected to terminate faster than "N".

C. Comparison of Implementations

Comparing the Julia CPU and GPU implementations, the GPU outperforms the CPU for large numbers of projections and samples. Despite having to solve the Newton step "J+GPU+N" outperforms "J+CPU+O", clearly showing the advantage of GPU-acceleration.

Lastly, the Python implementation "P+GPU+O" is compared to "J+CPU+O" and "J+GPU+O". Their runtime for 100 iterations, including initialization, was measured for 40 runs and its average and standard deviation is shown in Fig. 6. As "P+GPU+O" does not use lookup tables only one GM component is used for this comparison. "J+GPU+O" slightly outperforms it even in this case, where the difference would be much more significant for more components. Both GPU versions scale better than the CPU versions for large numbers of samples.

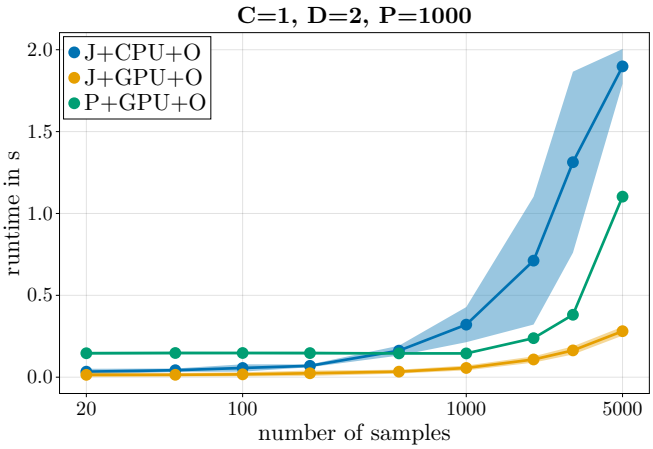


Fig. 6: Runtime for 100 iterations of the algorithm in [7] for different implementations and varying numbers of samples.

VII. Conclusion

This paper introduced key enhancements for PCD-based sampling of GM densities. We proposed an improved optimization procedure, leading to more accurate results as well as techniques for the fast computation of the gradient and the Hessian during optimization. Many of the proposed techniques are not specific to GMs and can easily be transferred to sample from other densities. The potential for parallelization of PCD-based sampling to achieve further speedup and scaling has been shown.

Finally, reference implementations of [7] in Python and of the improved algorithm Algorithm 1 in Julia are provided. Although reasonable effort has been made to optimize these implementations, their performance can likely be further increased by fine-tuning and, for example, applying techniques to optimize cache and memory access.

From a theoretical point of view, it is also worthwhile to investigate more closely the influence of the number of projections on the accuracy of the solution.

References

[1] Uwe D. Hanebeck, Dominik Prossel, Andrey A. Popov, Felipe Giraldo-Grueso, and Renato Zanetti. "Ensemble Gaussian Mixture Filter based on Projected Cramér-von Mises Distance". In: *Proceedings of the 2025 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2025)*. College Station, Texas, Sept. 2025.

[2] Daniel Frisch and Uwe D. Hanebeck. "Gaussian Mixture Particle Filter Step based on Method of Moments". In: *Proceedings of the 27th International Conference on Information Fusion (FUSION 2024)*. Venice, Italy, July 2024.

[3] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

[4] Russel E Caflisch. "Monte Carlo and Quasi-Monte Carlo Methods". In: *Acta numerica* 7 (1998), pp. 1–49.

[5] Igor Gilitschenski, Jannik Steinbring, Uwe D Hanebeck, and Miroslav Simandl. "Deterministic Dirac Mixture Approximation of Gaussian Mixtures". In: *17th International Conference on Information Fusion (FUSION)*. IEEE. 2014, pp. 1–7.

[6] Jannik Steinbring and Uwe D. Hanebeck. "S2KF: The Smart Sampling Kalman Filter". In: *Proceedings of the 16th International Conference on Information Fusion (Fusion 2013)*. Istanbul, Turkey, July 2013.

[7] Uwe D. Hanebeck. "Deterministic Sampling of Multivariate Densities based on Projected Cumulative Distributions". In: *Proceedings of the 54th Annual Conference on Information Sciences and Systems (CISS 2020)*. Princeton, New Jersey, USA, Mar. 2020.

[8] Uwe D. Hanebeck. "Optimal Reduction of Multivariate Dirac Mixture Densities". In: *at – Automatisierungstechnik* 63.4 (Apr. 2015), pp. 265–278.

[9] Felipe Giraldo-Grueso, Andrey A. Popov, Uwe D. Hanebeck, and Renato Zanetti. "Optimal Transport as a Reduction Technique for Deterministic Nonlinear Filtering". In: *Proceedings of the 28th International Conference on Information Fusion (FUSION 2025)*. Rio de Janeiro, Brazil, July 2025.

[10] Uwe D. Hanebeck, Marco F. Huber, and Vesa Klumpp. "Dirac Mixture Approximation of Multivariate Gaussian Densities". In: *Proceedings of the 2009 IEEE Conference on Decision and Control (CDC 2009)*. Shanghai, China, Dec. 2009.

[11] Lois Paulin, Nicolas Bonneel, David Coeurjolly, Jean-Claude Iehl, Antoine Webanck, Mathieu Desbrun, and Victor Ostromoukhov. "Sliced Optimal Transport Sampling." In: *ACM Trans. Graph.* 39.4 (2020), p. 99.

[12] Daniel Frisch and Uwe D. Hanebeck. "Deterministic Sampling on the Circle Using Projected Cumulative Distributions". In: *Proceedings of the 25th International Conference on Information Fusion (Fusion 2022)*. Linköping, Sweden, July 2022.

[13] Johann Radon. "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten". In: *Berichte über die Verhandlungen der Königlich-Sächsischen Gesellschaft der Wissenschaften zu Leipzig. Mathematisch-Physische Klasse* 69 (1917), pp. 262–277.

[14] Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde. "Generalized Sliced Wasserstein Distances". In: *Advances in Neural Information Processing Systems* 32 (2019).

[15] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. "Wasserstein Barycenter and its Application to Texture Mixing". In: *International Conference on Scale Space and Variational Methods in Computer Vision*. Springer. 2011, pp. 435–446.

[16] Curtis R Cook and Do Jin Kim. "Best Sorting Algorithm for Nearly Sorted Lists". In: *Communications of the ACM* 23.11 (1980), pp. 620–624.

[17] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. "Julia: A Fresh Approach to Numerical Computing". In: *SIAM Review* 59.1 (2017), pp. 65–98.

[18] Tim Besard, Christophe Foket, and Bjorn De Sutter. "Effective Extensible Programming: Unleashing Julia on GPUs". In: *IEEE Transactions on Parallel and Distributed Systems* (2018).

[19] Adam Paszke et al. "PyTorch: An Imperative Style, High-performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.