Time-efficient Bayesian Inference for a (Skewed) Von Mises Distribution on the Torus in a Deep Probabilistic Programming Language

Ola Rønning[1], Christophe Ley[2], Kanti V. Mardia[3a,3b] and Thomas Hamelryck[4]

***Abstract*— Probabilistic programming languages (PPLs) are at the interface between statistics and the theory of programming languages. PPLs formulate statistical models as stochastic programs that enable automatic inference algorithms and optimization. Pyro [1] and its sibling NumPyro [2] are PPLs built on top of the deep learning frameworks PyTorch [3] and Jax [4], respectively. Both PPLs provide simple, highly similar interfaces for inference using efficient implementations of Hamiltonian Monte Carlo (HMC), the No-U-Turn Sampler (NUTS), and Stochastic Variational Inference (SVI). They automatically generate variational distributions from a model, automatically enumerate discrete variables, and support formulating deep probabilistic models such as variational autoencoders and deep Markov models.**

The Sine von Mises distribution and its skewed variant are toroidal distributions relevant to protein bioinformatics. They provide a natural way to model the dihedral angles of protein structures, which is important in protein structure prediction, simulation and analysis. We present efficient implementations of the Sine von Mises distribution and its skewing in Pyro and NumPyro, and devise a simulation method that increases efficiency with several orders of magnitude when using parallel hardware (i.e., modern CPUs, GPUs, and TPUs). We demonstrate the use of the skewed Sine von Mises distribution by modeling dihedral angles of proteins using a Bayesian mixture model inferred using NUTS, exploiting NumPyro's facilities for automatic enumeration [5].

I. INTRODUCTION

Until recently, protein structure prediction posed a long-standing open challenge in structural biology. At the fourteenth Critical Assessment of Techniques for Protein Structure Prediction (CASP), [6] the deep learning-based method AlphaFold2 [7], [8] achieved levels of accuracy that enticed the CASP committee to declare the protein structure prediction problem solved. Despite this progress, contemporary methods for protein structure prediction generally do not systematically account for uncertainty. In particular, they fail to model aleatory uncertainty (due to protein dynamics or disorder [9]) and epistemic uncertainty (caused by experimental error, limitations of the probabilistic model and limited amounts of data). As proteins are often parameterized in terms of dihedral angles, directional statistics potentially provides a robust framework for factoring in such uncertainties [10].

Distributions on the 2-torus are attractive for protein structure modeling because a dihedral angle pair, corresponding to a point on the 2-torus, is often used to represent the conformation of the backbone of a single amino acid. [10]. Despite their potential, 2-torus distributions are available in few contemporary inference frameworks. To address this, we introduce efficient implementations of a 2-torus distribution in the deep PPLs Pyro [1], and Numpyro [2]. The distribution is a submodel of the Bivariate von Mises (BvM) distribution [11], known as the sine distribution [12].

The sine distribution is pointwise symmetric. However, it is not immediately evident that the probability distributions of dihedral angles observed in proteins display symmetry. [13] suggests an efficient procedure for skewing a toroidal distribution, which we refer to as sine skewing. Sine skewing works for any dimensionality gives rise to simple simulation procedures and is computationally efficient. In addition to the sine distribution, we also provide implementations of the sine skewing procedure in Pyro and NumPyro.

Pyro and NumPyro are modern deep PPLs with nearly identical user interfaces. However, the two PPLs differ in their computation backend and main inference engine. Pyro uses PyTorch [3] as its backend and is mainly designed for inference with Stochastic Variational Inference [14]. NumPyro is a daughter framework of Pyro that uses Jax [4] as backend. The main inference engine of NumPyro is Hamiltonian Monte Carlo with Iterative NUTS [2], a variant of the No-U-Turn sampler (NUTS) [15]. Iterative NUTS leverages Jax's tracing capabilities for considerable computational gains. Both frameworks provide methods to automatically generate variational distributions called "guides", enumerate (i.e. sum out) discrete variables, and formulate deep probabilistic models. Tailoring the simulation of the sine distribution [16] to use parallelism, we observe a 6000-fold increase in sampling time using a GPU in Pyro over the R-based implementation provided by [16]. These optimizations are not possible in NumPyro; our implementation provides a more modest 3000-fold increase in this framework over [16].

To illustrate the sine distribution and sine skewing, we use both to formulate mixture models of dihedral angles in proteins. We implement and infer the mixture models using NumPyro, but the provided code snippets translate to Pyro with minimal effort. We use iterative NUTS for inference and apply NumPyro's automatic enumeration to obtain differentiable models.

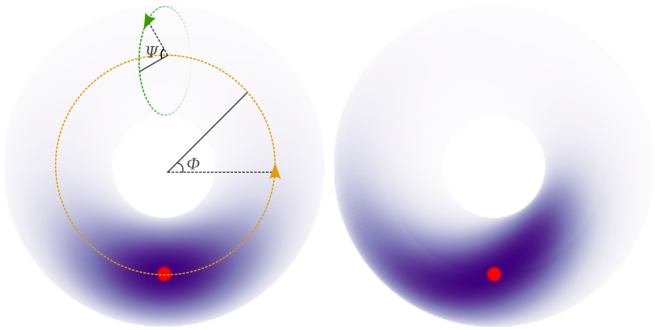
[1]Department of Computer Science, University of Copenhagen, Denmark ola@di.ku.dk

[2]Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium christophe.ley@ugent.be

[3a]School of Mathematics, University of Leeds, LS2 9JT, UK K.V.Mardia@leeds.ac.uk

[3b]Department of Statistics, University of Oxford, OX1 3LB, UK mardia@stats.ox.ac.uk

[4]Department of Biology and Department of Computer Science, University of Copenhagen, Denmark thamelry@bio.ku.dk



(a) $\rho = 0$ (b) $\rho = 1.7$

Fig. 1: (Sine) Distributions over angle pairs (ϕ, ψ) given by pdf $f(\phi, \psi|\boldsymbol{\mu}, \kappa_\phi = 2, \kappa_\psi = 2, \rho)$ projected onto a torus. Here $\kappa_\phi, \kappa_\psi > 0$ denote the concentration (anti-variance), and $\rho \geq 0$ is the correlation between the angles. The density is shaded in purple. The location $\boldsymbol{\mu}$ is marked by the red dot.

The paper proceeds as follows. In Section II we define the sine distribution and the sine skewing procedure and highlight their most prominent features. We provide implementation details in Section III. In Section IV we demonstrate the use of the sine skewed sine distribution in mixture models. Finally, in Section V we summarize and point to future work.

II. Toroidal Distributions

The Bivariate von Mises (BvM) distribution [11] is defined on the unit torus and is point-wise symmetric around its location $\boldsymbol{\mu} = (\mu_\phi, \mu_\psi)$. That is, the density of angle pairs (ϕ, ψ) given by $f(\phi, \psi; \boldsymbol{\theta})$ satisfies $f(\mu_\phi - \phi, \psi; \boldsymbol{\theta}) = f(\mu_\phi + \phi, \psi; \boldsymbol{\theta})$ and $f(\phi, \mu_\psi - \psi; \boldsymbol{\theta}) = f(\phi, \mu_\psi + \psi; \boldsymbol{\theta})$. We have collected all the parameters of f in $\boldsymbol{\theta}$ for conciseness. Fig. 1 illustrates this type of distribution. Because of its redundant number of parameters, various more parsimonious submodels have been proposed in the literature. Here, we focus on the sine distribution of [12], which we shall describe below. Sine skewing is a general approach on a d-torus to skew a given symmetric distribution (such as the BvM), and we have implemented it as such in both Pyro and NumPyro. However, for application purposes we are particularly interested in the 2-D version (i.e., as a distribution over a toroid or 2-torus).

A. Sine Distribution

[11] introduced a circular analogue to the bivariate normal distribution with log probability mass function (pdf)

$$\log f(\phi, \psi|\boldsymbol{\mu}, \boldsymbol{\kappa}, \boldsymbol{A}) \propto \kappa_1 \cos(\phi - \mu_1) + \kappa_2 \cos(\psi - \mu_2) + [\cos(\phi - \mu_1), \sin(\phi - \mu_1)]\boldsymbol{A}[\cos(\psi - \mu_1), \sin(\psi - \mu_1)]^T,$$

where $\kappa_1, \kappa_2 \in \mathbb{R}^+$ are concentration (anti-variation) parameters, $\mu_1, \mu_2 \in [-\pi, \pi)$ are location parameters and $\boldsymbol{A}$ is a real 2-by-2 matrix that captures the correlation between ϕ and ψ. [17] introduces a submodel of the above distribution with two fewer parameters and log pdf

$$\log f(\phi, \psi|\boldsymbol{\mu}, \boldsymbol{\kappa}, \alpha, \beta) \propto \kappa_1 \cos(\phi - \mu_1) + \kappa_2 \cos(\psi - \mu_2) + \alpha \cos(\phi - \mu_1)\cos(\psi - \mu_2) + \beta \sin(\phi - \mu_1)\sin(\psi - \mu_2), \tag{1}$$

where $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$. [17] formulated the normalization constant as a doubly infinite series. [12] resolved the issue with over-parameterization by setting $\alpha = 0$ and $\beta = \rho \in \mathbb{R}$ in Equation (1), yielding the pdf

$$f(\phi, \psi|\boldsymbol{\mu}, \boldsymbol{\kappa}, \rho) = C^{-1} \exp(\kappa_1 \cos(\phi - \mu_1) + \kappa_2 \cos(\psi - \mu_2) + \rho \sin(\phi - \mu_1)\sin(\psi - \mu_2)). \tag{2}$$

The ρ parameters control statistical dependence between the angles ϕ and ψ, with $\rho = 0$ meaning ϕ and ψ are independent and eq. (2) reducing two two von Mises distributions. [12] also formulated the calculation of the normalization constant C as the infinite series

$$\frac{1}{C} = (2\pi)^2 \sum_{i=0}^{\infty} \binom{2i}{i} \left(\frac{\rho^2}{4\kappa_1\kappa_2}\right)^2 I_i(\kappa_1) I_i(\kappa_2), \tag{3}$$

where I_i is the $i-$th order modified Bessel function of the first kind. [18] considered another subclass of Equation (1) by setting $\alpha = \beta = -\kappa_3$. To distinguish the two subclasses, [18] named Equation (2) the *sine model* and their own model, the Cosine model. [18] further proved that Equation (2) is unimodal if $\kappa_1\kappa_2 > \rho^2$ and bimodal if $\kappa_1\kappa_2 < \rho^2$, with $\kappa_1, \kappa_2 > 0$ and $\rho \in \mathbb{R}$.

B. Sine Skewing Procedure

The sine and cosine distributions are symmetric. However, for applications in bioinformatics, symmetry is not necessarily desirable. To address the issue of asymmetry, [13] introduced a general way to turn any symmetric density $f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta})$ into a skewed version

$$g(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{\theta}) = f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta}) \times (1 + \lambda_1 \sin(\phi - \mu_1) + \lambda_2 \sin(\psi - \mu_2)), \tag{4}$$

where $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)^T$ with $|\lambda_i| \leq 1$ and $|\lambda_1| + |\lambda_2| \leq 1$.

The resulting asymmetric distribution is prefixed with "sine-skewed", so skewing the sine distribution yields the sine skewed sine distribution. [13] shows that this results in several desirable properties. Most notable is that the normalization constant remains the same. They also show that the skewing procedure preserves the computability of any trigonometric moments of the underlying toroidal distribution. Finally, [13] provides a simple and efficient simulation scheme for the skewed distributions.

III. Implementation

In Pyro and Numpyro, the implementation of a distribution needs to provide four attributes and two methods. The attributes are

1) the batch shape, which denotes the shape of a batch of conditionally independent random variables;

2) the event shape, which denotes the shape of a single random variable;
3) the support (e.g., reals, positive, interval) and its dimensionality; and
4) the parameter types and their dimensionality.

The two methods implement i) calculating the log probability of a batch of events and ii) simulation, which given a desired sample shape must adhere to both the batch shape and the event shape of the distribution.

We can expand an instantiated distribution to an arbitrary batch shape and declare a given number of these dimensions part of an event. We call a distribution instantiated once its parameters are fixed. Computationally, an instantiated distribution is represented as a single object on a hardware device (i.e., GPU or CPU). It is common when running inference to use both devices when a GPU is available. Pyro's computational backend, PyTorch, provides simple methods for checking the current device of an objects and moving it to another device. NumPyro's backend, Jax, handles moving objects between devices automatically, but provides methods to manually intervene.

A. Batch Sampling the Sine Distribution

The support of the sine distribution is the 2-torus, and thus an event always comprises two dependent angles. In Pyro and NumPyro, a Python tuple is used to specify the event shape. Thus, the event shape of the sine distribution is always (2,), using the syntax of a singleton Python tuple.

To sample from the distribution in Pyro, we extend the work of [16] to parallel sampling of arbitrary batch- and sample-shapes. [16] provides a bivariate rejection-sampling for (among others) the sine distribution using an Angular Central Gaussian (ACG) distribution as envelope.

Our extension samples the marginal distribution over the first angle. This requires the sampling method to track and sample missing samples in parallel while avoiding branching and overusing pseudo-random numbers. To track the missing samples, we flatten[1] the sample- and batch-shape and allocate an empty piece of memory $\boldsymbol{M}$ for storing the samples with the flattened shape. Flattening has the advantages of simplifying indexing and providing a coalesced memory layout for GPU computation, needed for high utilization.

To minimize the number of pseudo-random numbers, we sample the smallest number of missing samples from the batches that are not filled, which yields proposals $\boldsymbol{P}$. In order to place the accepted proposals correctly into $\boldsymbol{M}$ (where samples are accumulated) we apply a $n \times m$ Boolean mask matrix $\boldsymbol{B}$, where n is the number of samplers and m total number of desired samples. The entries of $\boldsymbol{B}$ are given by

$$\boldsymbol{B}_{(i,j)} = \left[0 \leq j - \sum_{j'=1}^{j} 1_{\boldsymbol{M}_{(i,j')}} < \sum_{k=1}^{m} 1_{\boldsymbol{M}_{(i,k)}} \right]_{(i,j)}, \quad (5)$$

where $1_{\boldsymbol{A}_{(i,j)}}$ is the indicator function for an accepted samples in $\boldsymbol{M}$ at i, j and similarly $1_{\mathbf{P}_{(i,j)}}$ is the indicator for accepted proposals at i, j. Using a Boolean mask to select where accepted proposals are placed we avoid copying already accepted samples and branching on the Boolean values in the mask. Pyro avoids branching due to the tensor abstraction deployed in PyTorch, which provides primitives for using sub-tensors as references.

Because in NumPyro all tensors are immutable with concrete shapes, the shape of tensors cannot be a function of the state of execution. This means that the optimizations possible in Pyro are not possible in NumPyro. Therefore, the implementation in NumPyro is a straightforward translation of the R method provided in [16].

B. Sine Skewing

Sine skewing [13] is a procedure for breaking the symmetry of a given toroidal distribution. Treating sine skewing as a higher-order method, which takes a distribution and yields its skewed variate, allows for a simple implementation that works for any toroidal distribution with a simulation method.

The simulation method for the skewed distribution follows three steps outlined in [13]. For some k-dimensional base density f with location $\boldsymbol{\mu}$, parameters $\boldsymbol{\theta}$, and skewness $\boldsymbol{\lambda}$ their sampling scheme is

1) Sample $\boldsymbol{y}$ from base density f;
2) Sample u from the uniform density over $[0, 1]$;
3) Generate skewed samples $\boldsymbol{x}$ by

$$\boldsymbol{x} = \begin{cases} \boldsymbol{y} & \text{if } u \leq \frac{1}{2}\left(1 + \sum_{i=1}^{k} \sin(\boldsymbol{y}_i - \boldsymbol{\mu}_i)\right) \\ -\boldsymbol{y} + 2\boldsymbol{\mu} & \text{otherwise,} \end{cases}$$

where $\boldsymbol{y} \in [-\pi, \pi)^k$ is a draw from f and $\boldsymbol{x} \in [-\pi, \pi)^k$ is the corresponding skewed sample.

To compute the log probability density, we simply take the log of Equation (4), which yields

$$\log(f(\phi, \psi; \boldsymbol{\mu}, \boldsymbol{\theta})) + \log(1 + \lambda_\phi \sin(\phi - \mu_\phi) + \lambda_\psi \sin(\psi - \mu_\psi)),$$

for a density f on the 2-torus.

C. Benchmarking

We benchmark the sampling time for varying sampling sizes on CPU[2] and GPU[3]. Dispatching to the computation device (i.e., GPU or CPU) is declared in logic and handled by the backend frameworks, so the same high-level code (our implementations) is executed regardless of the device. We compare our implementation of the sine distribution to the implementation provided as a supplement to [16] in R. We are unaware of publicly available implementations of the sine skewed sine distribution with simulation methods and so did not benchmark our implementation of sine skewing against any reference. The timings are given in log (base ten) scale in Fig. 2.

For Pyro, the average time of ten repetitions is presented along with the associated standard deviation; only the average timings are given for the reference R implementation.

[1] $(2, 3, 4)$ becomes $(2 * 3 * 4) = (24,)$

[2] Intel® Xeon® Silver 4114 Processor (13.75M Cache 2.20 GHz).

[3] NVIDIA Quadro RTX 6000.

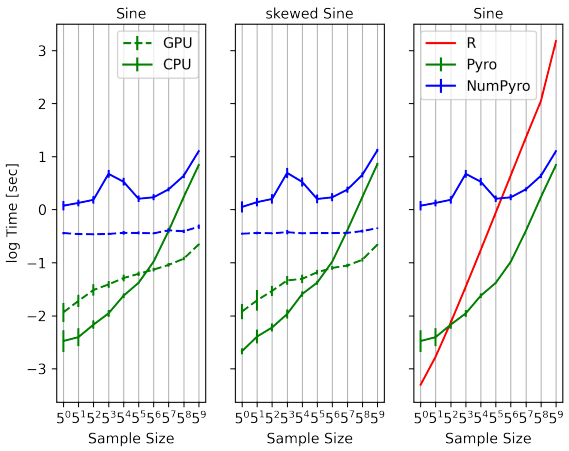


Fig. 2: Average timings in seconds for differing sample sizes. We compare sampling in R [16], with our sampling implementations in Pyro and NumPyro.

For NumPyro we run eleven repetitions and dismiss the first call which takes between two and seven seconds. We then report the average and deviation for the remaining ten repetitions. We do this because Jax can amortize the high cost of the first iteration (where the python program is traced) upon repeated computation. As expected, we observe that for Pyro, sampling is faster on CPU for a low number of samples (i.e., less than around three thousand); see the solid green lines in the left and middle plots of Fig. 2. Note that the simulation with skewing adds negligible overhead. The faster simulation for small sample size on the CPU is because i) the CPU does not incur overhead from transferring memory between devices and ii) small sample sizes provide limited parallelism for the GPU to exploit. The same plots for NumPyro (cross markers) show the effect of the different computational backend (i.e. Jax). Once the sampling scheme is traced, we observe close to constant computation time for exponentially growing samples sizes on GPU and only a single order of magnitude increase for CPU. The Pyro implementation is faster due to its use of the extended sampling scheme.

For large sample sizes we see a significant 3114-fold speedup for the GPU in the Pyro framework; and a 6785-fold speedup compared to the reference R implementation. NumPyro provides a more modest 3000-fold speedup compared to the R implementation.

The right plot of Fig. 2 shows that the reference implementation of the sine distribution is faster in R for sample sizes smaller than 25, which is due to the overhead associated with Pyro's PyTorch backend and its use of parallel subroutines. This also causes the improved performance of Pyro for larger sample sizes. NumPyro on CPU needs substantially larger samples sizes before outperforming the R implementation.

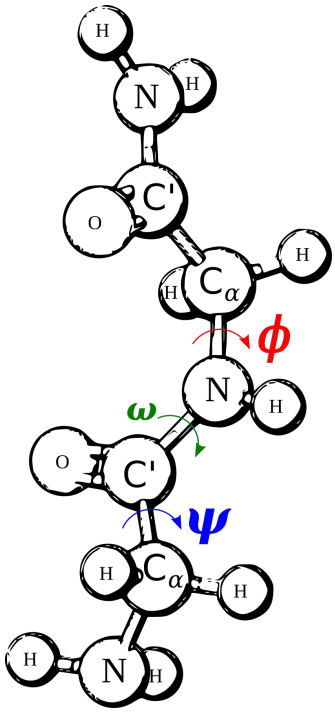


Fig. 3: A protein is a polymer of amino acids, whose geometry can be quantified by their dihedral or torsion angles. The ϕ, ψ, and ω triplet denotes the torsion angles associated with the $C' - N - C_\alpha - C'$ bonds, the $N - C_\alpha - C' - N$ bonds, and the $C_\alpha - C' - N - C_\alpha$ bonds, respectively. Assuming ideal bond angles and bond lengths, these angles suffice to describe the geometry of the protein's backbone. The ω angle tends to be very close to either pi or (in rare cases) 0. Not all triplet values can occur in amino acids, mainly because of steric clashes between atoms. The top row of Fig. 4 (known as Ramachandran plots) shows empirically measured angles pairs for serine, glycine, and proline. The angle pairs of serine are representative for the rest of the amino acids.

IV. Application

To illustrate our implementations, we model the dihedral angles pairs, see Fig. 3, of serine (S), glycine (G), and proline (P). The models are Bayesian mixture models with sine skewed sine distributions as likelihoods. We infer model parameters with Iterative NUTS in NumPyro. Following the analysis of [19] (Table 5), we use ten mixture components for glycine, nine for serine, and seven for proline. The angle pairs are taken from the 9mer dataset described in [20]. We restrict our attention to these three amino acids as they are representative of all amino acids.

A. Data

We obtained our dihedral angle data set from a previously described protein fragment data set [20]. The data set consists of protein fragments of length 9 (9mers) derived from the 3733 proteins selected with cullpdb [21]. All included proteins have i) resolution less than 1.6 ångström, ii) R-factor less than 0.25, iii) sequence identity below twenty

percent, and iv) sequence identity below 20 percent with proteins used in CASP13. All fragments consist of angles from the allowed region on the Ramachandran plot [22], with outliers removed using the PHENIX software [23]. We use the 60/20/20 split provided in the dataset and pool the angles by amino acid type. There are 47130 angles for proline, 74830 for glycine, and 59304 for serine. However, we only use a random subsample of one thousand angle pairs for each amino acid. We do this to avoid running long chains with enumeration, which scales linearly with the number of mixture components. Further, this alleviates the density peaks present in the data, which are challenging for both HMC and NUTS.

B. Mixture Model

To model the dihedral angles we consider mixture models of the form

$$\sum_{i=1}^{n} w_i \, \mathrm{g_{SS}}(\phi, \psi | \boldsymbol{\theta}_i),$$

where $0 < w_i < 1, \sum_i w_i = 1$ are mixture weights and $\boldsymbol{\theta}_i$ denoted all parameters of sine skewed sine distribution $\mathrm{g_{SS}}$ with pdf given by Equation (4) and antecedent pdf given by Equation (2). We refer to the mixture as the Skewed model.

Our approach is close to [24] who considers a mixture of sine distributions for modeling the entire Ramachandran plot. Also, like our approach, [13] modeled angle pairs for individual amino acids. However, they use only a two-component mixture model for comparing a range of toroidal distributions to their skewed variants. Furthermore, unlike [24], [13] our approach is Bayesian, which allows quantifying the uncertainty of the posterior.

For the concentration of the sine distribution we use a constant scaled Beta prior. As it is not obvious whether to favour high or low concentrations we introduce a hyper prior on the Beta prior. The hyper prior controls the mean $\beta_\mu \sim \mathtt{Uni}(0, 1)$, where $\mathtt{Uni}$ denotes the uniform distribution, and virtual count $\beta_c \sim \mathtt{Gam}(1, 1/20)$, where $\mathtt{Gam}$ is the gamma distribution.

The Beta prior is given by $\mathtt{Beta}(\beta_\mu \beta_c, (1 - \beta_\mu)\beta_c)$. To see this, consider a random variable $X \sim \mathtt{Beta}(A, B)$ whose expectation is $\beta_\mu = \mathbb{E}[X] = A/\beta_c = A/(A + B)$. Rearranging yields $\mathtt{Beta}(\beta_\mu \beta_c, (1 - \beta_\mu)\beta_c)$. The scaling of the concentration is fixed to seventy, and thus $0 \leq \phi, \psi \leq 70$.

Recall that the location of the sine distribution denote is $\boldsymbol{\mu} = (\mu_\phi, \mu_\psi) \in [-\pi, \pi)^2$. $\boldsymbol{\kappa} = (\kappa_\phi, \kappa_\psi) \in [0, \infty)^2$ denotes its concentration. For the correlation we scale $\sqrt{\kappa_\phi \kappa_\psi}$ by $\rho \in [0, 1]$, which ensures we obtain mixtures of uni-modal sine distributions.

We use uni-variate von Mises distributions on the locations, denoted as $\mathtt{vM}(\mu, \kappa)$. Like the sine distribution the von Mises distribution is parameterized by a location $\mu \in [-\pi, \pi)$ and concentration $\kappa \in [0, \infty)$. We use the prior $\mathtt{vM}(\pi, 2)$ on μ_ψ to avoid the forbidden region of the Ramachandran plot. For μ_ψ, we choose a flat von Mises prior.

For skewness we use a uniform prior. Recall the skewness is constrained to $|\lambda_\phi| + |\lambda_\psi| \leq 1$. We accommodate this constraint by first sampling λ_ϕ uniformly from $[-1, 1]$ and then using the sampled λ_ϕ to decide the support interval (i.e., $[-1 + |\lambda_\phi|, 1 - |\lambda_\phi|]$) for the uniform sample of λ_ψ.

For correlation we put a $\mathtt{Beta}(2, 10)$ prior on the scale factor ρ. The prior favors low correlation to maintain relatively high sample efficiency for the posterior predictive samples.

Finally, we put a Dirichlet distribution, denoted Dir, of our mixture weights.

Bringing it all together, we can write the Skewed model as

Algorithm 1 Skewed Model

1: $\beta_{\mu_\phi} \sim \mathtt{Uni}\,(0, 1)$
2: $\beta_{c_\phi} \sim \mathtt{Gam}\,(1, 1/20)$
3: $\beta_{\mu_\psi} \sim \mathtt{Uni}\,(0, 1)$
4: $\beta_{c_\psi} \sim \mathtt{Gam}\,(1, 1/20)$
5: **for** $i \in [1, \ldots, m]$ **do** ▷ Mixing Plate
6: $\quad \boldsymbol{\mu}_\phi^{(i)} \sim \mathtt{vM}\,(\pi, 2)$
7: $\quad \boldsymbol{\mu}_\psi^{(i)} \sim \mathtt{vM}\,(0, 0.1)$
8: $\quad \boldsymbol{\kappa}_\psi^{(i)} \sim \mathtt{Beta}(\beta_{\mu_\psi}\beta_{c_\psi}, (1 - \beta_{\mu_\psi})\beta_{c_\psi})$
9: $\quad \boldsymbol{\kappa}_\phi^{(i)} \sim \mathtt{Beta}(\beta_{\mu_\phi}\beta_{c_\phi}, (1 - \beta_{\mu_\phi})\beta_{c_\phi})$
10: $\quad \boldsymbol{\rho}^{(i)} \sim \mathtt{Beta}(2, 10)$
11: $\quad \boldsymbol{\lambda}_\phi^{(i)} \sim \mathtt{Uni}(-1, 1)$
12: $\quad \boldsymbol{\lambda}_\psi^{(i)} \sim \mathtt{Uni}(-1 + |\boldsymbol{\lambda}_\phi^{(i)}|, 1 - |\boldsymbol{\lambda}_\phi^{(i)}|)$
13: $\quad \boldsymbol{\pi}^{(i)} \sim \mathrm{Dir}(1)$
14: $\quad \boldsymbol{\theta}^{(i)} \leftarrow (\boldsymbol{\mu}_\phi, \boldsymbol{\mu}_\psi, 70\boldsymbol{\kappa}_\phi, 70\boldsymbol{\kappa}_\psi, \boldsymbol{\rho} \circ \sqrt{\boldsymbol{\kappa}_\phi \circ \boldsymbol{\kappa}_\psi}, \boldsymbol{\lambda}_\phi, \boldsymbol{\lambda}_\psi)$
15: **end for**
16: **for** $j \in [1, \ldots, D]$ **do** ▷ Observation Plate
17: $\quad (\boldsymbol{\phi}^{(j)}, \boldsymbol{\psi}^{(j)}) \sim \sum_{i=1}^{m} \boldsymbol{\pi}^{(i)} \, \mathrm{g_{SS}}^{(i)} \left(\phi, \psi; \boldsymbol{\theta}^{(i)}\right)$
18: **end for**

where m is number of mixture components, D is number angle pairs observations, and $\circ$ is the Hardamard product between real vectors. The Skewed model is presented graphically in Fig. 5. Note the mixture plate (line 3) corresponds to the outer rectangle in Fig. 5 and the observation plate (line 13) to the inner square in Fig. 5.

The informative priors used in are somewhat ad-hoc, and more structured investigations into what constitutes good/optimal priors would be of interest for both the fields of Bioinformatics and Directional Statistics.

NumPyro caches support transformations (e.g., $h : [-1, 1] \to \mathbb{R}$) for distributions with constrained support. The caching poses a challenge for the uniform skewness prior. To see the problem, note that the support of λ_ψ in Alg. 1 line 9 depends on the current value of λ_ϕ. Therefore, after the first accepted proposal the support transformation is stale, i.e. referring to an earlier computation state which is invalid for the current state. We opted for the equivalent $\lambda_\psi \sim (1 - |\lambda_\phi|)\,\mathtt{Uni}(-1, 1)$ in code as it makes the support static (i.e., always $[-1, 1]$). [4]

[4]During integration into Pyro, Du Phat (a core Pyro-PPL developer) noted the stick-breaking transformation [25] can be used for a more elegant and generalized solution.

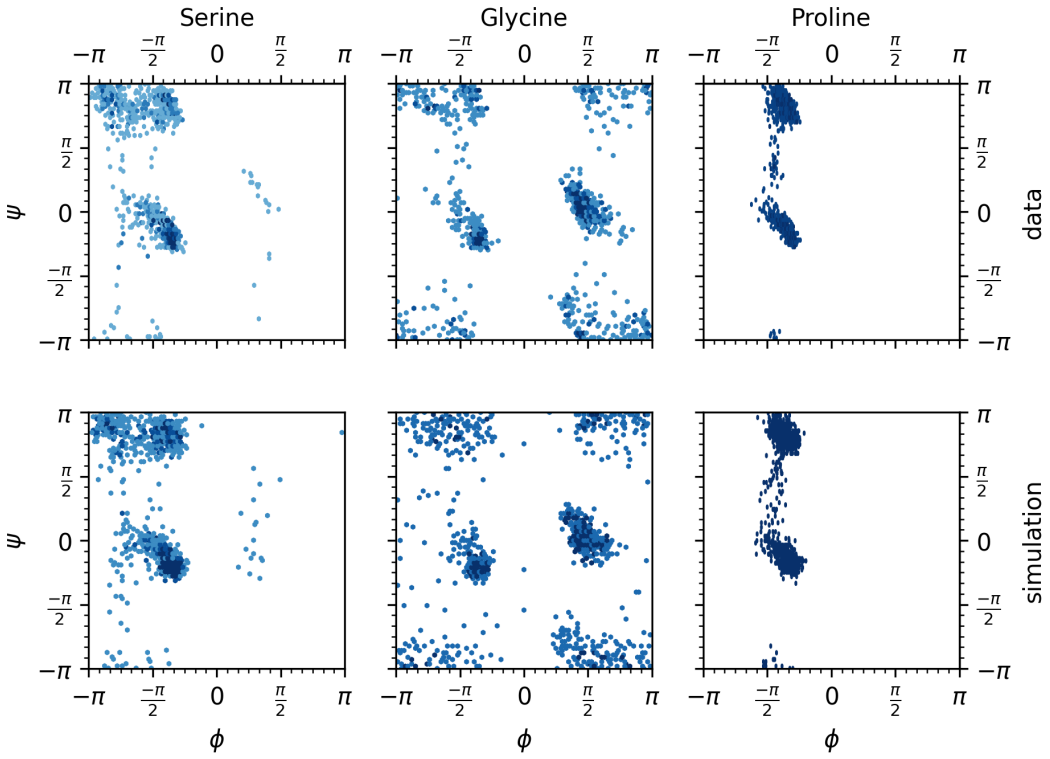


Fig. 4: The top row shows the Ramachandran plots (see Fig. 3 for details) of the data used for inference. The bottom row shows the Ramachandran plots of the posterior predictive samples from the Skewed mixture model, see Fig. 5.

The full code example is distributed with the NumPyro framework.

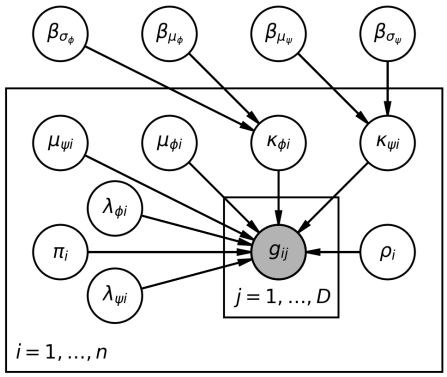


Fig. 5: Probabilistic graphical model representation of the hierarchical Bayesian mixture model using the sine skewed sine distribution. The large plate (shown as rectangle) applies to the n mixture components and the small plate applies to the D (i.e., 1000) angles.

C. Inference

We infer parameters using iterative NUTS in NumPyro. NUTS was proposed by [15] and introduces a stop criterion for the trajectory of the Hamiltonian dynamics in HMC. Iterative NUTS is an extension to NUTS by [2] which introduce a data-structure on the trajectory that makes NUTS work well with Jax.

We initialize the sine distribution locations with k-means clustering. We set the max tree depth for NUTS to seven, use target probabilty 0.8 and initial step length 1.71×10^{-2}. We run the chain for one thousand five hundred samples with five hundred burn-in, called warmup, samples. The time take to sample the chains (inference time) is about seven minutes on our CPU[5]. The mean MH-ratio, called acceptance probability, is slightly greater than our target probability (0.8). However, both chain diagnostics (ESS and $\hat{R}$) indicate the chains have converged. The diagnostics are detailed in Section V-A. All chain statistics are gathered in Table I.

D. Results

Fig. 4 shows that simulation from the posterior predictive distribution (bottom row) is visually close to the observation angles (top row). For glycine (middle row) the simulation has put some density in the forbidden region of Ramachandran plot. The local density in the Ramachandran plot can be extremely peaked, which poses a challenge for iterative NUTS. We handle this here by subsampling the data.

V. Discussion

We presented efficient implementations of the sine distribution and the sine skewing procedure for the 2-torus in the

[5]Intel® Xeon® Silver 4114 Processor (13.75M Cache, 2.20 GHz).

Amino Acid	ESS	$\hat{R}$	Divergences	warmup	samples	Inference time	Accept prob
Serine	1575.88	1	0	500	1500	7m23s	0.91
Glycine	1448.33	1	0	500	1500	6m40s	0.90
Proline	1384.20	1	0	500	1500	6m25s	0.93

TABLE I: Summary statistics of Markov chains for models trained on individual amino acids. Statistics shown include mean effective sample size (ESS), mean Gelman-Rubin diagnostic $\hat{R}$, number of divergent samples, number of warmup samples, total number of samples, total time for sampling, and mean acceptance probability.

PPLs Pyro and NumPyro. Currently, efficient implementations of directional distributions are generally scarce, with the univariate von Mises distribution as the notable exception. Therefore, we believe there is considerable untapped potential in making these methodologies more generally available to modern software running on parallel hardware. Another promising direction of research is tailoring inference algorithms to work on non-Euclidean manifolds. In particular, implementing the so-called reparameterization trick [26] for toroidal distributions would make these distributions available as directional priors in variational autoencoders [27].

APPENDIX

A. Pyro PPL automatic MCMC summary statistics

Below is a summary of Pyro's and NumPyro's summary statistics for the MCMC chain run for 3000 steps with a mixture of fifteen sine models on the serine angle-pairs. The Rubin-Gelman Diagnostic $\hat{R}$ [28] for two MCMC chains x and y of length n as

$$\hat{R} = \sqrt{\frac{1/n \sum_{i=1}^{n} \text{Var}[(x_i, y_i)]}{(n-1)/n^2 \sum_{i=1}^{n} \text{Var}[(x_i, y_i)] + \text{Var}[(x_j + y_j)_{j=1}^{n}]}}.$$

For a single chain $\hat{R}$ is computed by comparing the first half to the second half of the chain. $1 \leq \hat{R} < 1.1$ is generally considered an indicator of convergence.

The effective sample size, n_eff, measure the increase in uncertainty due to within chain auto-correlation. For m chains of length n the effective sample size can be estimated by

$$\text{n_eff} = \frac{mn}{-1 + 2\sum_{t=0}^{m}(\rho_{2t} + \rho_{2t+1})}$$

where ρ_t is the estimated auto-correlation with lag t.

Divergence are registered when the energy difference between the head of the chain and the proposal is greater than one thousand.

5.0% and 95.0% denote the boundaries of the posterior density with 90% of the probability mass for each variable.

	mean	std	median	5.0%	95.0%	n_eff	$\hat{R}$
beta_mean_phi	0.98	0.03	0.99	0.94	1.00	7.01	1.22
beta_mean_psi	0.99	0.01	0.99	0.98	1.00	33.65	1.02
beta_prec_phi	18.11	18.51	12.27	0.44	47.61	9.83	1.08
beta_prec_psi	23.22	16.91	18.89	1.93	44.83	55.77	1.02
corr_scale[0]	0.02	0.01	0.02	0.00	0.04	81.95	1.00
corr_scale[1]	0.02	0.02	0.02	0.00	0.04	21.03	1.05
corr_scale[2]	0.01	0.01	0.01	0.00	0.02	126.25	1.00
corr_scale[3]	0.00	0.00	0.00	0.00	0.01	108.85	1.00
corr_scale[4]	0.08	0.04	0.08	0.01	0.13	17.91	1.01
corr_scale[5]	0.07	0.04	0.07	0.01	0.13	100.84	1.00
corr_scale[6]	0.05	0.03	0.05	0.00	0.09	46.31	1.09
corr_scale[7]	0.00	0.00	0.00	0.00	0.01	13.63	1.01

...

Number of Hamiltonian trajectory divergences: 0

ACKNOWLEDGMENT

We would like to thank the Pyro PPL developers Fritz Obermeyer and Du Phat for valuable feedback and discussion about our implementations. We acknowledge support from the Independent Research Fund Denmark (DFF) under the grant "Deep Probabilistic Programming for Protein Structure Prediction". K. V. Mardia acknowledges the Leverhulme Trust for the Emeritus Fellowship. Christophe Ley's research is supported by the FWO Krediet aan Navorsers grant with reference number 1510391N.

REFERENCES

[1] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

[2] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: Composable transformations of Python+NumPy programs. `http://github.com/google/jax`, 2018.

[5] Fritz Obermeyer, Eli Bingham, Martin Jankowiak, Du Phan, and Jonathan P. Chen. Functional tensors for probabilistic programming. *arXiv preprint arXiv:1910.10775*, 2020.

[6] John Moult, Jan T Pedersen, Richard Judson, and Krzysztof Fidelis. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 23(3):ii–iv, 1995.

[7] Mohammed AlQuraishi. Machine learning in protein structure prediction. *Current Opinion in Chemical Biology*, 65:1–8, 2021.

[8] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[9] Joan-Emma Shea, Robert B Best, and Jeetain Mittal. Physics-based computational and theoretical approaches to intrinsically disordered proteins. *Current Opinion in Structural Biology*, 67:219–225, 2021.

[10] Thomas Hamelryck, Kanti V Mardia, and Jesper Ferkinghoff-Borg. *Bayesian Methods in Structural Bioinformatics*. Springer, 2012.

[11] Kanti V Mardia. Statistics of directional data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 37(3):349–371, 1975.

[12] Harshinder Singh, Vladimir Hnizdo, and Eugene Demchuk. Probabilistic model for two dependent circular variables. *Biometrika*, 89(3):719–723, 2002.

[13] Jose Ameijeiras-Alonso and Christophe Ley. Sine-skewed toroidal distributions and their application in protein bioinformatics. *Biostatistics*, 2021 (to appear).

[14] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[15] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[16] John T Kent, Asaad M Ganeiber, and Kanti V Mardia. A new unified approach for the simulation of a wide class of directional distributions. *Journal of Computational and Graphical Statistics*, 27(2):291–301, 2018.

[17] Louis-Paul Rivest. A distribution for dependent unit vectors. *Communications in Statistics-Theory and Methods*, 17(2):461–483, 1988.

[18] Kanti V Mardia, Charles C Taylor, and Ganesh K Subramaniam. Protein bioinformatics and mixtures of bivariate von mises distributions for angular data. *Biometrics*, 63(2):505–512, 2007.

[19] Christopher Bystroff, Vesteinn Thorsson, and David Baker. Hmmstr: a hidden markov model for local sequence-structure correlations in proteins. *Journal of molecular biology*, 301(1):173–190, 2000.

[20] Christian B. Thygesen, Ahmad S. Al-Sibahi, Christian S. Steenmanns, Lys S. Moreta, Anders B. Sørensen, and Thomas Hamelryck. Efficient generative modelling of protein structure fragments using a deep markov model. In *International Conference on Machine Learning*. PMLR, 2021 (to appear).

[21] Carol A Rohl, Charlie EM Strauss, Kira MS Misura, and David Baker. Protein structure prediction using rosetta. *Methods in Enzymology*, 383:66–93, 2004.

[22] Gopalasamudram N Ramachandran, Chandrasekharan Ramakrishnan, and Viswanathan Sasisekharan. Stereochemistry of polypeptide chain configurations. *Journal of Molecular Biology*, 7:95–99, 1963.

[23] Dorothee Liebschner, Pavel V Afonine, Matthew L Baker, Gábor Bunkóczi, Vincent B Chen, Tristan I Croll, Bradley Hintze, L-W Hung, Swati Jain, Airlie J McCoy, et al. Macromolecular structure determination using x-rays, neutrons and electrons: Recent developments in phenix. *Acta Crystallographica Section D: Structural Biology*, 75(10):861–877, 2019.

[24] Kanti V Mardia. Statistical approaches to three key challenges in protein structural bioinformatics. *Journal of the Royal Statistical Society: SERIES C: Applied Statistics*, pages 487–514, 2013.

[25] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994.

[26] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

[27] Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. *arXiv preprint arXiv:1808.10805*, 2018.

[28] Andrew Gelman, Donald B Rubin, et al. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.