



HAL
open science

Fast Python sampler of the von Mises Fisher distribution

Carlos Pinzón

► **To cite this version:**

| Carlos Pinzón. Fast Python sampler of the von Mises Fisher distribution. 2023. hal-04004568v1

HAL Id: hal-04004568

<https://hal.science/hal-04004568v1>

Preprint submitted on 24 Feb 2023 (v1), last revised 3 Aug 2023 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Public Domain

Fast Python sampler of the von Mises Fisher distribution

Carlos Pinzón
Inria, LIX

Abstract

This paper implements a method for sampling from the d -dimensional Von Mises Fisher distribution using NumPy, focusing on speed and readability. The complexity of the algorithm is $O(nd)$ for n samples, which is theoretically optimal taking into account that nd is the output size.

1 Introduction

The von Mises Fisher distribution (VMF), named after Richard von Mises [vM81] and Ronald Fisher [Fis53], is a probability distribution used in directional statistics and directional privacy mechanisms that generalizes the von Mises distribution (VM) for larger dimensions. The VM distribution has a circumference as domain, or equivalently the interval $[-\pi, \pi]$, while the domain of the d -dimensional VMF distribution is the $(d-1)$ -sphere S^{d-1} , hence VM corresponds to VMF when $d = 2$.

For given parameters of mean direction $\mu \in S^{d-1}$ and concentration $\kappa \in (0, \infty)$, the density function of the VMF is given, for each point $x \in S^{d-1}$, by

$$p(x \mid \mu, \kappa) = C_d(\kappa) \exp(\kappa \mu \cdot x),$$

where the \cdot denotes the dot product, i.e. the standard inner product in d -dimensions, and $C_d(\kappa)$ is a normalization constant that depends only on the dimensionality d and the concentration κ .

When $d = 2$, the domain is S^1 , i.e. the circumference of radius 1 in \mathbb{R}^2 , and letting θ_0 and θ be the angles that define μ and x respectively, the density function is given by $p(\theta \mid \theta_0, \kappa) = C_2(\kappa) \exp(\kappa \cos(\theta_0 - \theta))$, because $\mu \cdot x = |\mu| |x| \cos(\theta_0 - \theta) = \cos(\theta_0 - \theta)$. This formula coincides with that of the VM distribution in the form that it is typically presented, with its parameter varying in $[-\pi, \pi]$.

There exist implementations for VM in the main scientific Python packages SciPy [Sci23] and NumPy [Num23], but they do not implement the VMF in general (for $d \neq 2$). In this paper, we propose a fast implementation of the VMF that runs in $\Theta(nd)$, which is optimal, since $n \times d$ is also the size of the output.

2 Related work

There are two Python implementations for sampling from the VMF distribution [ten23, Whi23]. Before introducing the main difference between our algorithm and these implementations, let us describe in detail the principles they follow.

◦ **Principle 1 (rotation).** Due to the rotational symmetry, it suffices to have a sampler that handles only $\mu = \hat{e}_d = (0, 0, \dots, 0, 1)^\top$, because the resulting samples can be rotated afterwards towards the true μ . This rotation is carried out using a rotation matrix, which can be computed using Givens rotations and Householder reflections [Hou58].

◦ **Principle 2 (decomposition).** The sampling problem can be decomposed into computing the random angular deviation $\theta \in [0, \pi]$ between the output x and μ , and the random direction z in which the angle occurs.

Figure 2.1, taken from Weggenmann and Kerschbaum [WK21], depicts the decomposition principle for the case $d = 3$. The figure shows the angle θ between μ and x , and all the remaining orientation information, is captured in the $(d - 1)$ -dimensional unitary vector ξ . Since μ is fixed to be $\mu = \hat{e}_d = (0, 0, 1)$, we can embed $\xi = (\xi_1, \xi_2)$ in 3 dimensions as $z = (\xi_1, \xi_2, 0)$, so that it is perpendicular to μ . The problem of sampling

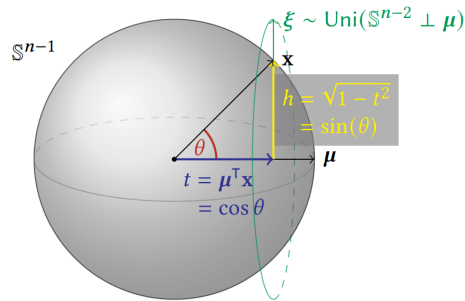


Figure 2.1: Decomposition into θ and ξ .

from VMF can therefore be reduced to sampling θ and ξ separately, and letting the output (assuming $\mu = \hat{e}_d$) be $x = \cos(\theta)\mu + \sin(\theta)z$.

◦ **Principle 3 (Muller).** Sampling ξ can be done using the algorithm by Muller [Mul59]. More precisely, if $Y_1, \dots, Y_{d-1} \sim \mathcal{N}(0, 1)$ are independent, and we let $Y = (Y_1, \dots, Y_{d-1})^\top$, it is known that the unitary vector $\xi := Y/|Y|$ is uniformly distributed on the $(d - 2)$ -sphere. For instance, when $d = 3$, ξ is a 2-dimensional vector taken from the 1-sphere (the circumference), and when $d = 2$, ξ is a 1-dimensional vector taken from the 0-sphere (the set $\{-1, 1\}$).

◦ **Principle 4 (Ulrich-Woods).** Sampling θ can be done using the algorithm by Ulrich [Ulr84], which was later revised and improved by Woods [Woo94]. It can be shown from the definition of the VMF distribution that the marginal distribution of the angle $\theta \in [0, \pi]$ is given for some constant $C'_d(\kappa)$ by

$$p(\theta \mid \mu, \kappa) = C'_d(\kappa) (\sin \theta)^{d-2} \exp(\kappa \cos \theta).$$

Ulrich and Woods propose a rejection method for generating samples θ that follow this distribution.

3 Algorithm

We propose to avoid the rotation step and to work entirely in the d -dimensional space. This means that the Muller method needs to be adjusted to produce vectors uniformly in $\{z \in S^{d-1} : z \perp \mu\}$ instead of $\{\xi \in S^{d-2}\}$. The Ulrich-Woods method, however, is left untouched, although it is vectorized to increase speed. The implementation is shown below.

```
import numpy as np

def random_VMF(mu, kappa, size=None):
    """
    Von Mises-Fisher distribution sampler with
    mean direction mu and concentration kappa
    """
    # parse input parameters
    n = 1 if size is None else np.product(size)
    shape = () if size is None else tuple(np.ravel(size))
    mu = np.asarray(mu)
    mu = mu / np.linalg.norm(mu)
    (d,) = mu.shape
    # z component: radial samples perpendicular to mu
    z = np.random.normal(0, 1, (n, d))
    z /= np.linalg.norm(z, axis=1, keepdims=True)
    z = z - (z @ mu[:, None]) * mu[None, :]
    z /= np.linalg.norm(z, axis=1, keepdims=True)
    # sample angles (in cos and sin form)
    cos = _random_VMF_cos(d, kappa, n)
    sin = np.sqrt(1 - cos**2)
    # combine angles with the z component
    x = z * sin[:, None] + cos[:, None] * mu[None, :]
    return x.reshape((*shape, d))

def _random_VMF_cos(d: int, kappa: float, n: int):
    """
    Generate n iid samples t with density function given by
    p(t) = someConstant * (1-t**2)**((d-2)/2) * exp(kappa*t)
    """
    # b = Eq. 4 of https://doi.org/10.1080/03610919408813161
    b = (d - 1) / (2 * kappa + (4 * kappa**2 + (d - 1)**2)**0.5)
    x0 = (1 - b) / (1 + b)
    c = kappa * x0 + (d - 1) * np.log(1 - x0**2)
    found = 0
    out = []
    while found < n:
        m = min(n, int((n - found) * 1.5))
        z = np.random.beta((d - 1) / 2, (d - 1) / 2, size=m)
        t = (1 - (1 + b) * z) / (1 - (1 - b) * z)
        test = kappa * t + (d - 1) * np.log(1 - x0 * t) - c
        accept = test >= -np.random.exponential(size=m)
        out.append(t[accept])
        found += len(out[-1])
    return np.concatenate(out)[:n]
```

References

- [Fis53] Ronald Aylmer Fisher. Dispersion on a sphere. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 217(1130):295–305, 1953.
- [Hou58] Alston S Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4):339–342, 1958.
- [Mul59] Mervin E Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Communications of the ACM*, 2(4):19–20, 1959.
- [Num23] Numpy.org. Von mises. <https://numpy.org/doc/stable/reference/random/generated/numpy.random.vonmises.html>, 2023. Accessed: 2023-02-16.
- [Sci23] Scipy.org. Von mises. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.vonmises.html>, 2023. Accessed: 2023-02-16.
- [ten23] tensorflow.org. Von mises fisher. https://github.com/tensorflow/probability/blob/v0.19.0/tensorflow_probability/python/distributions/vonmises_fisher.py#L44-L519, 2023. Accessed: 2023-02-16.
- [Ulr84] Gary Ulrich. Computer generation of distributions on the m-sphere. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 33(2):158–163, 1984.
- [vM81] Richard von Mises. Uber die” ganzzahligkeit” der atomgewicht und verwandte fragen. *Physikal. Z.*, 19:490–500, 1981.
- [Whi23] Daniel Whittenbury. Von mises. <https://dlwhittenbury.github.io/ds-2-sampling-and-visualising-the-von-mises-fisher-distribution-inp-dimensions.html>, 2023. Accessed: 2023-02-16.
- [WK21] Benjamin Weggenmann and Florian Kerschbaum. Differential privacy for directional data. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1205–1222, 2021.
- [Woo94] Andrew TA Wood. Simulation of the von mises fisher distribution. *Communications in statistics-simulation and computation*, 23(1):157–164, 1994.