An R Package for Fast Sampling from von Mises Fisher Distribution

Elysée Aristide Houndetoungan

2020-09-11

Context

I build an R package named **vMF** which samples from the von Mises-Fisher distribution ($\mathcal{M}$) as **movMF**. However, unlike the **movMF** package (Hornik and Grün, 2018) which also simulates and estimates mixtures of $\mathcal{M}$, **vFM** instead focuses on fast sampling from $\mathcal{M}$. The package also computes the density and the normalization constant of the von Mises-Fisher distribution.

Note that **movMF** is more general and can be used for many other purposes. It cannot be replaced by **vMF** which is more specific.

The von Mises Fisher distribution is used to model coordinates on a hypersphere of dimension $p \geq 2$. It can be considered as the equivalent of the normal distribution on a hypersphere. The von Mises Fisher distribution is characterized by two parameters. The location (or mean directional) parameter $\boldsymbol{\mu}$ around which simulations from the distribution will be concentrated and the intensity parameter η which measures the intensity of concentration of the simulations around $\boldsymbol{\mu}$. The higher η, the more the simulations are concentrated around $\boldsymbol{\mu}$. Comparing to the normal distribution, $\boldsymbol{\mu}$ is similar to the mean parameter of the normal distribution and $\frac{1}{\eta}$ is similar to the standard deviation.

There are several definitions of the density function of $\mathcal{M}$. In this package, the density is normalized by the uniform distribution without loss of generality. This is also the case in Mardia and Jupp (2009) and Hornik and Grün (2013).

Let $\mathbf{z} \sim \mathcal{M}(\eta, \boldsymbol{\mu})$. Then,

$$f_p(\mathbf{z}|\eta, \boldsymbol{\mu}) = C_p(\eta) e^{\eta \mathbf{z}' \boldsymbol{\mu}},$$

where $C_p(x) = \left(\frac{x}{2}\right)^{\frac{p}{2}-1} \frac{1}{\Gamma\left(\frac{p}{2}\right) I_{\frac{p}{2}-1}(x)}$ is the normalization constant and $I_{.}(.)$ the Bessel function of the first kind defined by:

$$I_\alpha(x) = \sum_{m=0}^{\infty} \frac{\left(\frac{x}{2}\right)^{2m+\alpha}}{m!\Gamma(m+\alpha+1)}.$$

The normalization with respect to the uniform distribution simplifies some results. For example, $C_p(0) = 1$.

Simulation from von Mises Fisher distribution

The following algorithm provides a rejection sampling scheme for drawing a sample from the $\mathcal{M}$ with mean directional parameter $\boldsymbol{\mu} = (0, ..., 0, 1)$ and concentration (intensity) parameter $\eta \geq 0$ (see Section 2.1 in Hornik and Grün, 2014).

- Step 1. Calculate b using * Step 1. Calculate b using

$$b = \frac{p-1}{2\eta + \sqrt{2\eta^2 + (p-1)^2}}.$$

Let $x_0 = (1-b)/(1+b)$ and $c = \eta x_0 + (p-1)\log\left(1-x_0^2\right)$.

- Step 2. Generate $Z \sim Beta((p-1)/2, (p-1)/2)$ and $U \sim Unif([0,1])$ and calculate

$$W = \frac{1-(1+b)Z}{1-(1-b)Z}.$$

- Step 3. If

$$\eta W + (p-1)\log(1-x_0 W) - c < \log(U),$$

go to step 2.

- Step 4. Generate a uniform $(d-1)$-dimensional unit vector V and return

$$X = \left(\sqrt{1-W^2}V', W\right)'$$

The uniform $(d-1)$-dimensional unit vector V can be generated by simulating $d-1$ independent standard normal random variables and normalizing them. To get samples from $\mathcal{M}$ with arbitrary mean direction parameter $\boldsymbol{\mu}$, X is multiplied from the left with a matrix where the first $d-1$ columns consist of unitary basis vectors of the subspace orthogonal to $\boldsymbol{\mu}$ and the last column is equal to $\boldsymbol{\mu}$.

Package Installation

The code source of the package is written in C++ with the package **Rcpp** (Eddelbuettel et al., 2020). **vMF** is currently available on GitHub. Its installation requires to prior install **devtools** package (Wickham et al., 2020). Moreover, Windows users should install Rtools compatible with their R version.

vMF package can be installed from this GitHub repos using the `install_github` function of **devtools**. All the dependencies will also be installed automatically.

```
library(devtools)
install_github("ahoundetoungan/vMF")
```

The option `build_vignettes = TRUE` can be added if one desires to install the vignettes.

Comparison of vMF and movMF

In this section, I compare **vMF** and **movMF**. First of all, I compare samples drawn from both packages to make sure that they are identical.

```
library(vMF)
library(movMF)

n      <- 5 # Number of draws
set.seed(123)
xvMF   <- rvMF(n,c(1,0,0))
set.seed(123)
xmovMF <- rmovMF(n,c(1,0,0))
all.equal(c(xvMF), c(xmovMF))
#> [1] "Mean relative difference: 0.9835279"
xvMF
```

```
#>              [,1]       [,2]        [,3]
#> [1,] -0.01290181  0.9189938  0.39406082
#> [2,]  0.89595883 -0.4399727  0.06067791
#> [3,]  0.45972070  0.7545718 -0.46827153
#> [4,]  0.33965060  0.3410720  0.87653145
#> [5,]  0.67069299  0.7051006  0.23022595
xmovMF
#>             [,1]       [,2]        [,3]
#> [1,]  0.4959510  0.13536134  0.85773532
#> [2,]  0.3337858 -0.71803316 -0.61074989
#> [3,] -0.5661812  0.70756922 -0.42282928
#> [4,]  0.9021605  0.39518250 -0.17302358
#> [5,]  0.9952369 -0.08643024 -0.04509155
```

The outputs are surprisingly different although the random number generators are set fixed. By checking the source code of **movMF** to understand the issue, I notice that both packages do not code their sampling function as the same way. In **movMF**, the function is more general and is built for drawing samples from mixture of $\mathcal{M}$. The sampling function first generates n (the sample size) random numbers from the uniform distribution before running the algorithm described above for the simulation. The uniform random numbers are used further in the function when it is mixture of $\mathcal{M}$. However, they are not used when the distribution is not mixed. Therefore, the outputs are different as the random numbers generator is called n times before running the simulation algorithm in **movMF**. In that case, even by the random number generators are set fixed, they will not return the same numbers. To get the same outputs, I also need to simulate n numbers from uniform distribution after setting the random number generator and before calling `rvMF`.

```
n      <- 30
set.seed(123)
ddpcr::quiet(runif(n))
xvMF   <- rvMF(n,c(1,0,0))
set.seed(123)
xmovMF <- rmovMF(n,c(1,0,0))
all.equal(c(xvMF), c(xmovMF))
#> [1] TRUE
xvMF[1:5,]
#>             [,1]       [,2]        [,3]
#> [1,]  0.9686901 -0.1926995 -0.15654514
#> [2,]  0.8074952  0.5897337 -0.01286950
#> [3,]  0.7669230 -0.6370259  0.07763455
#> [4,] -0.1575664  0.3537674  0.92196607
#> [5,]  0.2602641  0.9038644 -0.33954642
xmovMF[1:5,]
#>             [,1]       [,2]        [,3]
#> [1,]  0.9686901 -0.1926995 -0.15654514
#> [2,]  0.8074952  0.5897337 -0.01286950
#> [3,]  0.7669230 -0.6370259  0.07763455
#> [4,] -0.1575664  0.3537674  0.92196607
#> [5,]  0.2602641  0.9038644 -0.33954642
```

In doing so, the outputs are identical.

I now compare the performance of both packages, especially the running times required to draw samples. I use the **rbenchmark** package (Kusnierczyk, 2012) and compare the performance for several sample sizes.

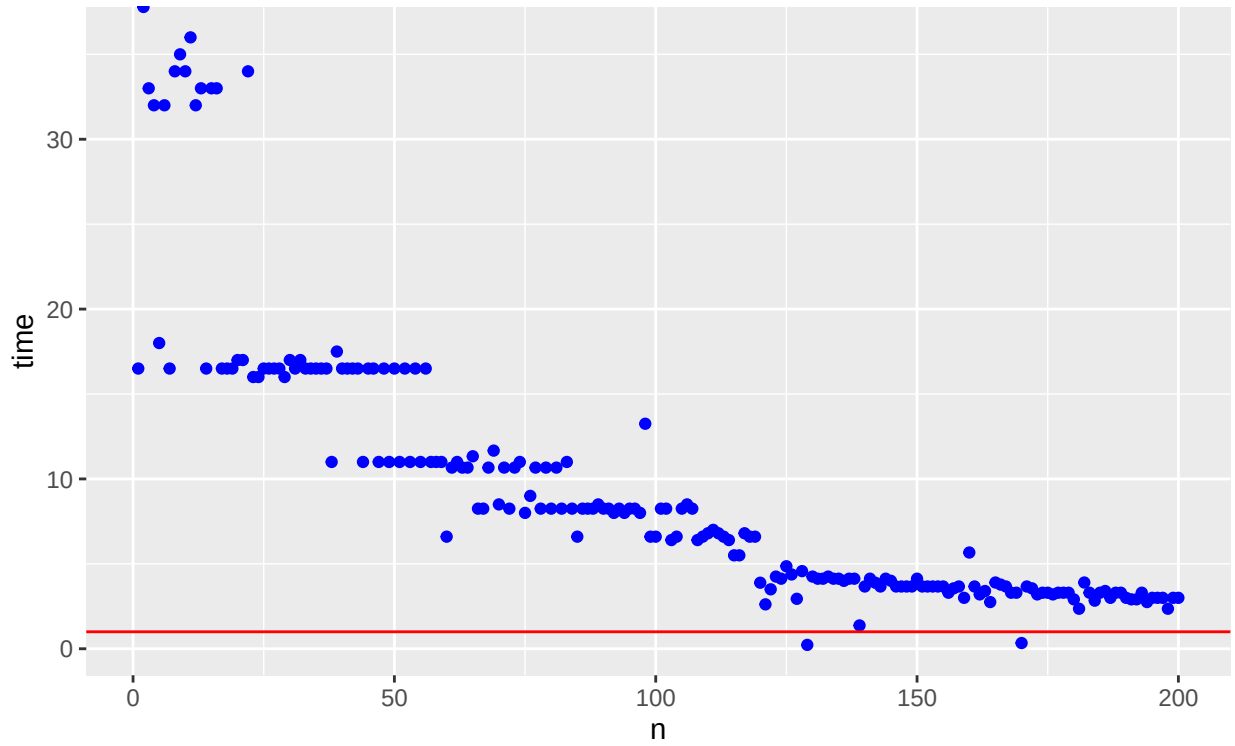
```
library(rbenchmark)
```

```
fcompare <- function(n) {
  benchmark("vMF" = rvMF(n,c(1,0,0)), "movMF" = rmovMF(1,c(1,0,0)))
}

fcompare(1)
#>    test replications elapsed relative user.self sys.self user.child sys.child
#> 2 movMF          100   0.035       35     0.035        0          0         0
#> 1   vMF          100   0.001        1     0.001        0          0         0
fcompare(10)
#>    test replications elapsed relative user.self sys.self user.child sys.child
#> 2 movMF          100   0.034       34     0.034        0          0         0
#> 1   vMF          100   0.001        1     0.002        0          0         0
fcompare(100)
#>    test replications elapsed relative user.self sys.self user.child sys.child
#> 2 movMF          100   0.033     8.25     0.033        0          0         0
#> 1   vMF          100   0.004     1.00     0.004        0          0         0
```

vMF performs over **movMF**. The relative running time of **movMF** decreases when the sample size increases. This is also confirmed by the following outputs. The performance of **vMF** is much better when only few simulations are performed. When the sample is too large, the two package require approximately the same time.

```
out  <- unlist(lapply(1:200, function(x) fcompare(x)$elapsed[1]/fcompare(x)$elapsed[2]))
summary(out)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.2276  3.6667  8.0000     Inf 12.0625     Inf
library(ggplot2)
ggplot(data = data.frame(n = 1:200, time = out), aes(x = n, y = time)) +
  geom_point(col = "blue") + geom_hline(yintercept = 1, col = 2)
```



Some papers use simulations from the von Mises Fisher distribution in Markov Chain Monte Carlo (MCMC).

In these frameworks, only one draw is performed at each iteration of the MCMC. This is for example the case in Boucher and Houndetoungan (2020), Breza et al. (2020), McCormick and Zheng (2015) and many others. I would suggest using **vMF** package in such contexts.

In the following code, I consider the process $(\mathbf{z}_t)_{t\in\mathbb{N}}$ that follows a random walk of von Mises Fisher distribution. The first variable of the process, $\mathbf{z}_0$ is uniformly drawn from an hypersphere of dimension 4 and $\mathbf{z}_t \sim \mathcal{M}(1, \mathbf{z}_{t-1})\ \forall t > 0$. Simulating this process has the same complexity as using von Mises Fisher draws in MCMC. I use both package functions to simulate the first 1000 values of $(\mathbf{z}_t)_{t\in\mathbb{N}}$.

```
set.seed(123)
P                  <- 4
initial            <- rmovMF(1, rep(0, P))
# Fonction based on vMF to simulate theta
SamplevMF          <- function(n) {
  output           <- matrix(0, n + 1, P)
  output[1, ]      <- initial
  for (i in 1:n) {
    output[i + 1,] <- rvMF(1, output[i,])
  }
  return(output)
}

# Fonction based on movMF to simulate theta
SamplemovMF        <-function(n){
  output           <- matrix(0, n + 1, P)
  output[1, ]      <- initial
  for (i in 1:n) {
    output[i + 1,] <- rmovMF(1, output[i,])
  }
  return(output)
}
benchmark("vMF" = SamplevMF(1000), "movMF" = SamplemovMF(1000))
#>    test replications elapsed relative user.self sys.self user.child sys.child
#> 2 movMF          100  33.948   51.988    33.930    0.004          0         0
#> 1   vMF          100   0.653    1.000     0.624    0.028          0         0
```

The comparison of the running times shows that **vMF** performs better.

References

Boucher, V. and Houndetoungan, E. A. (2020). Estimating peer effects using partial network data.

Breza, E., Chandrasekhar, A. G., McCormick, T. H., and Pan, M. (2020). Using aggregated relational data to feasibly identify network structure without network data. *American Economic Review*, 110(8):2454–84.

Eddelbuettel, D., Francois, R., Allaire, J., Ushey, K., Kou, Q., Russell, N., Bates, D., and Chambers, J. (2020). *Rcpp: Seamless R and C++ Integration*. R package version 1.0.5.

Hornik, K. and Grün, B. (2013). On conjugate families and jeffreys priors for von mises–fisher distributions. *Journal of statistical planning and inference*, 143(5):992–999.

Hornik, K. and Grün, B. (2014). movmf: an r package for fitting mixtures of von mises-fisher distributions. *Journal of Statistical Software*, 58(10):1–31.

Hornik, K. and Grün, B. (2018). *movMF: Mixtures of von Mises-Fisher Distributions*. R package version 0.2-4.

Kusnierczyk, W. (2012). *rbenchmark: Benchmarking routine for R*. R package version 1.0.0.

Mardia, K. V. and Jupp, P. E. (2009). *Directional statistics*, volume 494. John Wiley & Sons.

McCormick, T. H. and Zheng, T. (2015). Latent surface models for networks using aggregated relational data. *Journal of the American Statistical Association*, 110(512):1684–1695.

Wickham, H., Hester, J., and Chang, W. (2020). *devtools: Tools to Make Developing R Packages Easier*. R package version 2.2.2.