# ExponentialFamilyManifolds.jl: Representing exponential families as Riemannian manifolds

Mykola Lukashchuk[1], Dmitry Bagaev[1], Albert Podusenko[2], İsmail Şenöz[2], and Bert de Vries[1]

[1]Department of Electrical Engineering, Eindhoven University of Technology
[2]Lazy Dynamics BV, Eindhoven

## ABSTRACT

`ExponentialFamilyManifolds.jl` implements exponential family natural parameter spaces as Riemannian manifolds, enabling geometric optimization over probability distributions. The package automatically manages parameter constraints, such as ensuring the positive definiteness of precision matrices for normal distributions. By representing exponential family distributions as manifolds that conform to the `ManifoldsBase.jl` interface, it allows users to leverage optimization techniques from `Manopt.jl` for these manifolds. Applications in maximum likelihood estimation and variational inference highlight the package's practical utility.

## Keywords

Exponential Family, Julia, Manifold Optimization, Optimization, Probability Distributions

## 1. Introduction

Probabilistic inference problems can often be formulated as optimization problems, such as maximum likelihood estimation (MLE) or variational inference (VI) [5]. In practice, this often involves optimizing over a space of probability distributions to find one that best fits the data (MLE) or approximates a target distribution (VI). Unfortunately, direct optimization over arbitrary probability distributions is intractable. To address these challenges, we often resort to solutions within parametric families, with the exponential family of distributions being a popular choice for its ability to transform inference into convex optimization problems and its natural emergence from maximum entropy principles [7]. As detailed in [7, Chapter 3], this family not only includes many common distributions like Gaussian, Dirichlet, and Wishart, but also provides a unified mathematical framework for representing graphical models while ensuring distributions maintain maximal uncertainty consistent with observed data. Formally, a distribution is a member of the exponential family if there exists a parameter vector $\eta$ such that its probability density can be written as

$$p(x|\eta) = h(x) \exp\left(\eta^\top T(x) - A(\eta)\right),$$

where $\eta$ is called the natural parameter. Although this parametrization makes optimization more tractable, challenges remain due to the geometric constraints on the parameter space. For example, the natural parameter vector $\eta$ for a multivariate Normal distribution includes a matrix component proportional to the negative precision matrix.

We present `ExponentialFamilyManifolds.jl`, a Julia [4] package that addresses these challenges by implementing the parameter spaces as Riemannian manifolds. By bridging `ExponentialFamily.jl` [8] with the manifold interface from `ManifoldsBase.jl` [2] and concrete manifolds from `Manifolds.jl`, the new package enables geometrically aware optimization techniques to handle parameter constraints automatically using `Manopt.jl` [3].

## 2. A tour through `ExponentialFamilyManifolds.jl`

In this section, we introduce the `NaturalParametersManifold` interface, the core functionality of our package. We then demonstrate how this interface can be applied to optimization tasks such as maximum likelihood estimation and variational inference.

### 2.1 The core interface: `NaturalParametersManifold`

A key feature of the core interface `NaturalParametersManifold` from `ExponentialFamilyManifolds.jl` is that any point on a manifold—used within `Manopt.jl`—can also be viewed as an exponential-family distribution. This dual view means we can invoke typical distribution operations (e.g., `rand` or `logpdf`) on the same object that supports geometric operations like `retract`. In other words, we can both *manipulate* a point as part of a manifold-based optimization routine, and use it to *evaluate* functions that treat a point as a distribution for operations like likelihood and sampling.
The `NaturalParametersManifold` is implemented as a subtype of `ManifoldsBase.AbstractDecoratorManifold`, defining a generic interface to construct and work with the natural-parameter space of an arbitrary exponential-family distribution. This "decorator" approach allows us to extend the functionality of a base manifold while preserving its core operations. For example, the natural parameters of the Beta distribution form the manifold

$$\mathcal{M}_{\text{Beta}} = (-1, \infty) \times (-1, \infty),$$

which can be represented using `ProductManifold` from `ManifoldsBase.jl` together with concrete manifolds from `Manifolds.jl` through coordinate transformations, which we call `partition_point` and `transform_back`; the first one takes a natural parameter and transforms it into the corresponding manifold (in the Beta case, shifting it by 1), and the second one is the inverse operation.
`ExponentialFamilyManifolds.jl` offers an interface to construct `NaturalParametersManifold` objects

```
M = get_natural_manifold(Beta, ())
```

The method takes a distribution type, a dimensions tuple (empty for univariate distributions, or e.g., `(3,)` for 3D multivariate normals), and an optional conditioner. This produces a manifold `M` on which each point corresponds to valid parameters of the Beta distribution. To obtain a standard `ExponentialFamilyDistribution` object from a point `p` on `M`, we use `convert(EFD, M, p)`[1] method. Internally, `convert` applies `transform_back(M, p)` to recover the usual parameter vector before constructing the distribution. An example of this can be seen in Listing 2, where we use `convert` inside an MLE routine.

## 2.2 Using `ExponentialFamilyManifolds.jl` for Optimization

`ExponentialFamilyManifolds.jl` transforms distribution-approximation tasks (e.g., maximum likelihood estimation or variational inference) into manifold-optimization problems, allowing users to focus solely on defining an objective function. Listing 1 demonstrates a generic workflow for performing gradient-based optimization on a manifold `M`: first, compute the gradient of the objective `f` using `ManifoldDiff.jl`, which provides differentials for functions defined on manifolds that implement the `ManifoldsBase.jl` interface [2], and then execute `gradient_descent` from `Manopt.jl`. The `gradient_descent` function iteratively applies the `retract` method to update parameters along the negative gradient direction.

Code 1: Generic manifold-based optimization

```
function optimize(M, f, η_init)
    # compute the Riemannian gradient
    # ... your differentiation backend here
    g(M, η) = ManifoldDiff.gradient(M, f, η,
backend)
    return gradient_descent(M, f, g, η_init)
end
```

To perform MLE and VI, we only need to define their respective objective functions, the negative log-likelihood for MLE, and the Free Energy for VI, and then use the generic `optimize` function to perform the inference [2]. The objective functions for MLE and VI are provided in listings 2 and 3, respectively [1]. Figure 1 shows the results of both approaches. The upper panel demonstrates how MLE and VI produce different approximations to the same target distribution, while the bottom panel compares different families of the approximating distribution.

Code 2: Objective functions for MLE

```
function f_mle(M, η, samples)
    dist = convert(EFD, M, η)
    return -mean(logpdf(dist, s) for s in samples)
end
```

Code 3: Objective functions for VI

```
function f_vi(M, η, t, size)
    q = convert(EFD, M, η)
    samples = rand(MersenneTwister(22), q, size)
    diff(s) = logpdf(t, s) - logpdf(q, s)
    return -mean(diff, samples)
end
```

[1]Note that to make the listings smaller in place of `ExponentialFamilyDistribution`, we write `EFD`.

[2]The complete example is implemented in the code at `https://github.com/ReactiveBayes/ExponentialFamilyManifolds.jl/blob/paper/paper_example/paper_example.jl`.

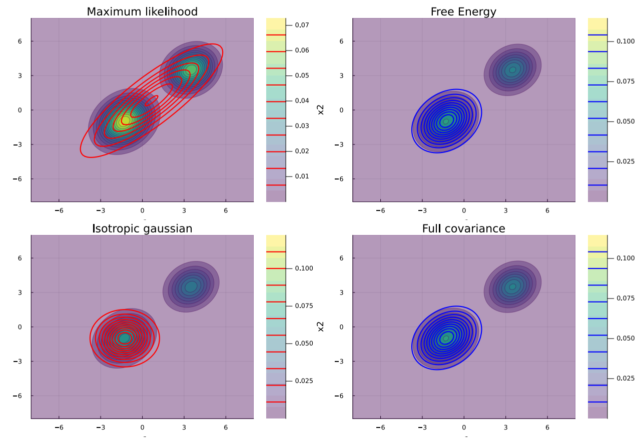

Fig. 1: **Top:** MLE (red) vs. VI (blue) approximations of a target distribution (shown in the background): a mixture of two Gaussian distributions with different means and covariances. While MLE showed covering behavior towards the target density, VI demonstrated a mode-seeking behavior. **Bottom:** Isotropic (red) vs. full-covariance (blue) Gaussian approximations using Free Energy objective for both, showing the trade-off between computational efficiency and approximation quality - Isotropic Gaussian approximations are fast to obtain, but less flexible than full-covariance Gaussians.

The `optimize` (the Listing 1) function demonstrates how gradient descent can be implemented using the `NaturalParametersManifold` interface. Additionally, the approach can be easily adapted to use the natural gradient [1] by incorporating the Fisher information into the gradient computations, as shown in Listing 4.

Code 4: Computing the natural gradient with `ForwardDiff.jl` [6]

```
function natural_gradient(M, p, f, grad)
    q = convert(EFD, M, p)
    F = fisherinformation(q)
    grad = ForwardDiff.gradient(
        x -> f(M, p), p
    )
    return F \ grad
end
```

## 3. Future work

While most of the distributions in this package use product manifold geometry, the Fisher-Rao metric [1] provides a more suitable geometry for exponential families, potentially leading to more robust optimization (see Listing 4). Additionally, extending the package to support mixture families of exponential family distributions could enhance its capabilities since, for example, mixture models can better capture multiple modes in VI.

## 4. References

[1] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998. doi:10.1162/089976698300017746.

[2] Seth D. Axen, Mateusz Baran, Ronny Bergmann, and Krzysztof Rzecki. Manifolds.Jl: An Extensible Julia Framework for Data Analysis on Manifolds. *ACM Transactions on Mathematical Software*, 49(4), 2023. doi:10.1145/3618296.

[3] Ronny Bergmann. Manopt.jl: Optimization on Manifolds in Julia. *Journal of Open Source Software*, 7(70):3866, 2022. doi:10.21105/joss.03866.

[4] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. doi:10.1137/141000671.

[5] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017. doi:10.1080/01621459.2017.1285773.

[6] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-Mode Automatic Differentiation in Julia. *arXiv:1607.07892 [cs]*, 2016.

[7] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1):1–305, 2008. doi:10.1561/2200000001.

[8] İsmail Şenöz, Dmitry Bagaev, and Mykola Lukashchuk. ExponentialFamily.jl. *GitHub repository*, 2023. doi:10.5281/zenodo.15101588.