juliacon

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` 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 normal distribution covariance matrices, while providing advanced optimization techniques by integrating `ExponentialFamily` with `Manifolds`. Applications for maximum likelihood estimation and variational inference demonstrate 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. This family includes many common distributions, such as Gaussian, Dirichlet, and Wishart distributions. Any member of this family can be written as

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

where η are the natural parameters. Although this parametrization makes optimization more tractable, challenges remain due to the geometric constraints on the parameter space. For example, the covariance matrices of a Normal distribution must be positive definite. We present `ExponentialFamilyManifolds`, a Julia [4] package that addresses these challenges by implementing the parameter spaces as Riemannian manifolds. By bridging `ExponentialFamily` [7] and `Manifolds` [2], the new package enables geometrically aware optimization techniques to handle parameter constraints automatically using `Manopt` [3].

2. A tour through `ExponentialFamilyManifolds`

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

The NaturalParametersManifold defines a generic interface to construct and work with the natural parameters space of an arbitrary distribution from the exponential family. For example, the natural parameters of the Beta distribution form the manifold

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

which, through `Manifolds`, can be represented as follows[1]

```
ProductManifold(
    ShiftedPositiveNumbers(static(-1)),
    ShiftedPositiveNumbers(static(-1))
)
```

`ExponentialFamilyManifolds` provides the following method

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

to construct this manifold. Here, `M` is the manifold corresponding to $\mathcal{M}_{\text{Beta}}$. Once `M` is obtained, one can sample a vector of natural parameters on `M` and convert points from it to an instance of ExponentialFamilyDistribution, the main interface in `ExponentialFamily`. We highlight the retract method in the NaturalParametersManifold interface because gradient-based optimization requires a mechanism to move in a given direction (a tangent vector) without leaving the manifold. For instance, to move from a point η in the direction X can be implemented in the following way

```
ζ = retract(M, η, X)
```

This step enforces constraints automatically (e.g., $\zeta \in \mathcal{M}_{\text{Beta}}$), freeing practitioners from manual checks of validity.

2.2 Using `ExponentialFamilyManifolds` for Optimization

`ExponentialFamilyManifolds` 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 `ForwardDiff` [6], and then execute gradient_descent from `Manopt`. The gradient_descent function iteratively applies the retract method to update parameters along the gradient direction.

[1] `ShiftedPositiveNumbers` is a convenience wrapper around the `PositiveNumbers` manifold implemented in `ExponentialFamilyManifolds` to shift the domain by a constant.

Code 1: Generic manifold-based optimization

```
function optimize(M, f, η_init)
    g(M, η) = ForwardDiff.gradient(
            η -> f(M, η), η
    )
    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 carry on the inference[2]. The objective functions for MLE and VI are provided in listings 2 and 3, respectively (note that to make the listings smaller in place of ExponentialFamilyDistribution, we write EFD). 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
```

The optimize (the Listing 1) function demonstrates how gradient descent can be implemented using the NaturalParametersManifold interface. Additionally, it can be easily adapted to use the natural gradient [1] by incorporating the Fisher information into the gradient computations, as shown in Listing 4. This flexibility allows researchers to perform natural MLE or natural VI by simply reusing the objective functions.

Code 4: Implementing the natural gradient

```
function natural_gradient(M, η, f)
    q = convert(EFD, M, η)
    F = fisherinformation(q)
    grad = ForwardDiff.gradient(
        x -> f(M, x), η
    )
    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 variational inference.

[2] A complete working implementations is available at `https://github.com/ReactiveBayes/ExponentialFamilyManifolds.jl/blob/paper/paper_example/paper_example.jl`

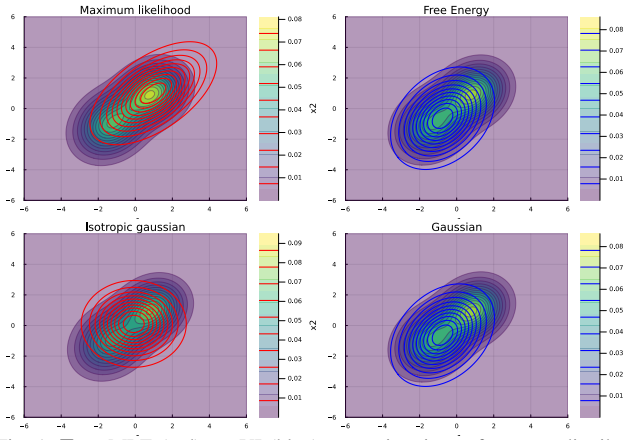

Fig. 1: **Top:** MLE (red) vs. VI (blue) approximations of a target distribution (shown in the background): a mixture of two Gaussian distributions with different means and covariances. While MLE showed covering behavior towards the target density, VI demonstrated a mode-seeking behavior. **Bottom:** 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.

4. References

[1] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, January 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), December 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. Publisher: The Open Journal.

[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, April 2017. doi:10.1080/01621459.2017.1285773. Publisher: ASA Website _eprint: https://doi.org/10.1080/01621459.2017.1285773.

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

[7] İsmail Şenöz, Dmitry Bagaev, and Mykola Lukashchuk. ExponentialFamily.jl, October 2023.