The Journal of Open Source Software

RxInfer: A Julia package for reactive real-time Bayesian inference

Dmitry Bagaev [1¶], **Albert Podusenko** [1], **and Bert de Vries** [1]

1 Technical University of Eindhoven ¶ Corresponding author

DOI: 10.21105/joss.05161

Software

- Review
- Repository
- Archive

Editor: Jacob Schreiber

Reviewers:

- @dhvalden
- @alstat

Submitted: 03 January 2023
Published: 31 March 2023

Summary

Bayesian inference realizes optimal information processing through a full commitment to reasoning by probability theory. The Bayesian framework is positioned at the core of modern AI technology for applications such as speech and image recognition and generation, medical analysis, robot navigation, and more. The framework describes how a rational agent should update its beliefs when new information is revealed by the agent's environment. Unfortunately, perfect Bayesian reasoning is generally intractable, since calculations of (often) very high-dimensional integrals are required for many models of interest. As a result, a number of numerical algorithms for approximating Bayesian inference have been developed and implemented in probabilistic programming packages. Successful methods include the Laplace approximation (Gelman et al., 2015), variants of Monte Carlo (MC) sampling (Salimans et al., n.d.), Variational Inference (VI) (Blei et al., 2017), Automatic-Differentiation Variational Inference (ADVI) (Kucukelbir et al., 2017), and Black-Box Variational Inference (BBVI) (Bamler & Mandt, 2017).

We present **RxInfer.jl**, which is a Julia (Jeff Bezanson et al., 2012; J. Bezanson et al., 2017) package for real-time variational Bayesian inference based on reactive message passing in a factor graph representation of the model under study (Bagaev & Vries, 2021). **RxInfer.jl** provides access to a powerful model specification language that translates a textual description of a probabilistic model into a corresponding factor graph representation. In addition, **RxInfer.jl** supports hybrid variational inference processes, where different Bayesian inference methods can be combined in different parts of the model, resulting in a straightforward mechanism to trade off accuracy for computational speed. The underlying implementation relies on a reactive programming paradigm and supports by design the processing of infinite asynchronous data streams. In the proposed framework, the inference engine *reacts* to new data and automatically updates relevant posteriors.

Over the past few years, the inference methods in this package have been tested on many advanced probabilistic models, resulting in several publications in highly ranked journals such as Entropy (Podusenko, Kouw, et al., 2021; Şenöz et al., 2021), Frontiers (Podusenko, Erp, Koudahl, et al., 2021), and conferences such as MLSP-2021 (Podusenko, Erp, Bagaev, et al., 2021), EUSIPCO-2022 (Erp & Vries, 2022; Podusenko et al., 2022) and SiPS (Nguyen et al., 2022).

Statement of need

Many important AI applications, including audio processing, self-driving vehicles, weather forecasting, and extended-reality video processing, and others require continually solving an inference task in sophisticated probabilistic models with a large number of latent variables. Often, the inference task in these applications must be performed continually and in real time in response to new observations. Popular MC-based inference methods, such as the No U-Turn Sampler (NUTS) (Hoffman & Gelman, 2011) or Hamiltonian Monte Carlo (HMC)

sampling (Brooks et al., 2011), rely on computationally heavy sampling procedures that do not scale well to probabilistic models with thousands of latent states. Therefore, MC-based inference is practically not suitable for real-time applications. While the alternative variational inference method (VI) promises to scale better to large models than sampling-based inference, VI requires the derivation of gradients of the "variational Free Energy" cost function. For large models, manual derivation of these gradients might not be feasible, while automated "black-box" gradient methods do not scale either because they are not capable of taking advantage of sparsity or conjugate pairs in the model. Therefore, while Bayesian inference is known as the optimal data processing framework, in practice, real-time AI applications rely on much simpler, often ad hoc, data processing algorithms.

Solution proposal

We present **RxInfer.jl**, a package for processing infinite data streams by real-time Bayesian inference in large probabilistic models. **RxInfer.jl** implements variational Bayesian inference as a variational Constrained Bethe Free Energy (CBFE) functional optimization process (Şenöz et al., 2021). The underlying inference engine derives its speed from taking advantage of both statistical independencies and conjugate pairings of variables in the factor graph. Inference proceeds continually by an automated reactive message passing process on the graph, where each message carves away a bit of the variational Free Energy cost function. Very often, closed-form message computation rules are available for specific nodes and node combinations, leading to much faster inference than sampling-based inference methods, and additionally enables hierarchical composition of different models without need for extra derivations. These properties distinguish **RxInfer.jl** from other popular Bayesian inference libraries in Julia, such as **Turing.jl** (Ge et al., 2018), **Stan.jl** (Stan Development Team, 2022; Stan.jl Development Team, 2022), and others, which are not designed to run inference continually in response to new observations in real-time.

Overview of functionality

RxInfer.jl is an open source package, available at https://github.com/biaslab/RxInfer.jl, and enjoys the following features:

- A user-friendly specification of probabilistic models. Through Julia macros, **RxInfer.jl** is capable of automatically transforming a textual description of a probabilistic model to a factor graph representation of that model.
- A hybrid inference engine. The inference engine supports a variety of well-known message passing-based inference methods such as belief propagation, structured and mean-field variational message passing, expectation propagation, expectation maximization, and conjugate-computation variational inference (CVI) (Akbayrak et al., 2022).
- A customized trade-off between accuracy and speed. For each location (node and edge) in the graph, **RxInfer.jl** allows a custom specification of the inference constraints on the variational family of distributions in the CBFE optimization procedure. This enables the use of different Bayesian inference methods at different locations of the graph, leading to an optimized trade-off between accuracy and speed.
- Support for real-time processing of infinite data streams. **RxInfer.jl** is based on a reactive programming paradigm that enables asynchronous data processing as soon as data arrives.
- Support for large static data sets. The package is not limited to real-time processing of data streams and also scales well to batch processing of large data sets and large probabilistic models that can include hundreds of thousands of latent variables (Bagaev, 2021).
- **RxInfer.jl** is extensible. The public API defines a straightforward and user-friendly way to extend the built-in functionality with custom nodes and message update rules.

- A large collection of precomputed analytical inference solutions. Current built-in functionality includes fast inference solutions for linear Gaussian dynamical systems, autoregressive models, hierarchical models, discrete-valued models, mixture models, invertible neural networks (Erp & Vries, 2022), arbitrary nonlinear state transition functions, and conjugate pair primitives.
- The inference procedure is auto-differentiable with external packages, such as **ForwardDiff.jl** (Revels et al., 2016) or **ReverseDiff.jl**.
- The inference engine supports different types of floating-point numbers, such as `Float32`, `Float64`, and `BigFloat`.

A large collection of examples is available at https://biaslab.github.io/RxInfer.jl/stable/examples/overview/.

Example usage

In this section, we show a small example based on Example 3.7 in Sarkka (Särkkä, 2013), where the goal is to track in real-time the state (angle and velocity) of a simple pendulum system. The differential equations for a simple pendulum can be written as a special case of a continuous-time nonlinear dynamic system where the hidden state $x(t)$ is a two-dimensional vector $\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \equiv \begin{bmatrix} \alpha \\ v \end{bmatrix}$ with α and v being the angle and velocity, respectively, and the state transition function $f(x) = \begin{bmatrix} x^{(1)} + x^{(2)}\Delta t \\ x^{(2)} - g \cdot \sin(x^{(1)})\Delta t \end{bmatrix}$. For more detailed derivations we refer interested reader to Särkkä (2013).

We use the **RxInfer**'s `@model` macro to specify the probabilistic model. We use the `@meta` macro to specify an approximation method for the nonlinearity in the model, the `@constraints` macro to define constraints for the variational distributions in the Bethe Free Energy optimization procedure, and the `@autoupdates` macro to specify how to update priors about the current state of the system. Finally, we use the `rxinference` function to execute the inference process, see Figure 1. The inference process runs in real time and takes 162 microseconds on average per observation on a single CPU of a regular office laptop (MacBook Pro 2018, 2.6 GHz Intel Core i7).

```julia
# `g` is the gravitational constant
f(x) = [x[1] + x[2] * Δt, x[2] - g * sin(x[1]) * Δt]

# We use the `@model` macro to define the probabilistic model
@model function pendulum()
    # Define reactive inputs for the `prior`
    # of the current angle state
    prior_mean = datavar(Vector{Float64})
    prior_cov  = datavar(Matrix{Float64})

    previous_state ~ MvNormal(mean = prior_mean, cov = prior_cov)
    # Use `f` as state transition function
    state ~ f(previous_state)

    # Assign a prior for the noise component
    noise_shape = datavar(Float64)
    noise_scale = datavar(Float64)
    noise ~ Gamma(shape = noise_shape, scale = noise_scale)

    # Define reactive input for the `observation`
    observation = datavar(Float64)
```

```
    # We observe only the first component of the state
    observation ~ Normal(mean = dot([1.0, 0.0], state), precision = noise)
end

@constraints function pendulum_constraint()
    # Assume the `state` and the `noise` are independent
    q(state, noise) = q(state)q(noise)
end

@meta function pendulum_meta()
    # Use the `Linearization` approximation method
    # around the nonlinear function `f`
    f() -> Linearization()
end

function pendulum_experiment(observations)

    # The `@autoupdates` structure defines how to update
    # the priors for the next observation
    autoupdates = @autoupdates begin
        prior_mean  = mean(q(state))
        prior_cov   = cov(q(state))
        noise_shape = shape(q(noise))
        noise_scale = scale(q(noise))
    end

    results = rxinference(
        model = pendulum(),
        constraints = pendulum_constraint(),
        meta = pendulum_meta(),
        autoupdates = autoupdates,
        data = (observation = observations,),
        initmarginals = (
            # We assume a relatively good prior for the very first state
            state = MvNormalMeanPrecision([0.5, 0.0], [100.0 0.0; 0.0 100.0]),
            # And we assign a vague prior for the noise component
            noise = Gamma(1.0, 100.0)
        ),
        # We indicate that we want to keep a history of estimated
        # states and the noise component
        historyvars = (state = KeepLast(), noise = KeepLast()),
        keephistory = length(observations),
        # We perform 5 VMP iterations on each observation
        iterations = 5,
        # We start the inference procedure automatically
        autostart = true
    )

    return results
end
```

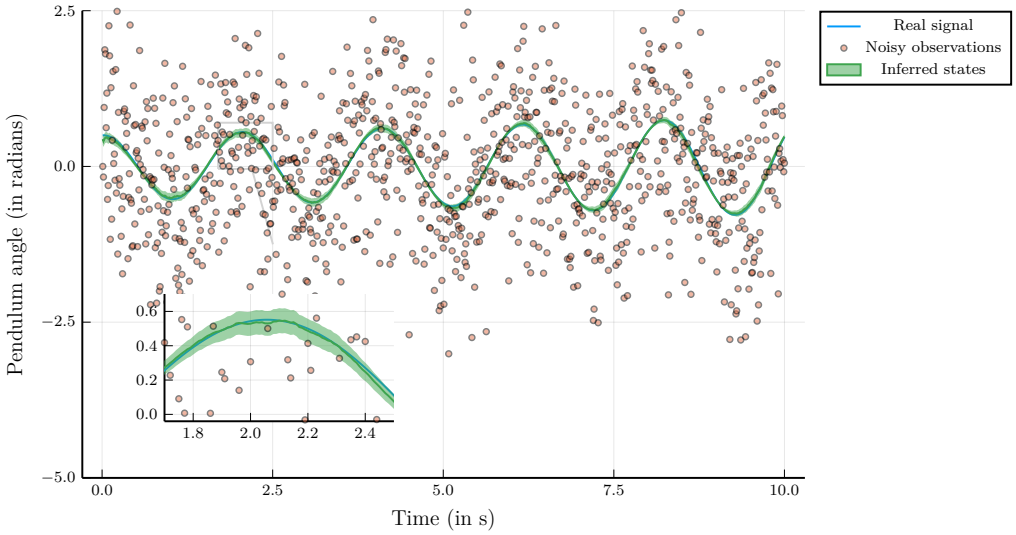

Figure 1: The inference results for the pendulum example. X-axis represents time t (in seconds). Y-axis represents the current angle of the pendulum (in radians) at time t. Real (unobserved) signal is shown in blue line. Observations are shown as orange dots. The inference results are shown as green line with area, which represents posterior uncertainty (one standard deviation). The inference process runs in real time and takes 162 microseconds on average per observation on a single CPU of a regular office laptop (MacBook Pro 2018, 2.6 GHz Intel Core i7).

Acknowledgments

The authors gratefully acknowledge contributions and support from colleagues in the BIASlab in the Department of Electrical Engineering at the University of Eindhoven of Technology.

References

Akbayrak, S., Şenöz, İ., Sarı, A., & de Vries, B. (2022). Probabilistic programming with stochastic variational message passing. *International Journal of Approximate Reasoning*, *148*, 235–252. https://doi.org/10.1016/j.ijar.2022.06.006

Bagaev, D. (2021). *ReactiveMP.jl: A Julia package for automatic Bayesian inference on a factor graph with reactive message passing*. Zenodo. https://doi.org/10.5281/ZENODO.6365000

Bagaev, D., & Vries, B. de. (2021). Reactive Message Passing for Scalable Bayesian Inference. *arXiv:2112.13251 [Cs]*. http://arxiv.org/abs/2112.13251

Bamler, R., & Mandt, S. (2017). Structured Black Box Variational Inference for Latent Time Series Models. *arXiv:1707.01069 [Cs, Stat]*. http://arxiv.org/abs/1707.01069

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Bezanson, Jeff, Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A Fast Dynamic Language for Technical Computing. *arXiv:1209.5145 [Cs]*. https://doi.org/10.48550/arXiv.1209.5145

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877. https://doi.org/10.1080/01621459.2017.1285773

Brooks, S., Gelman, A., Jones, G., & Meng, X.-L. (Eds.). (2011). *Handbook of markov chain monte carlo*. Chapman; Hall/CRC. https://doi.org/10.1201/b10905

Erp, B. van, & Vries, B. de. (2022). Hybrid Inference with Invertible Neural Networks in Factor Graphs. *2022 30th European Signal Processing Conference (EUSIPCO)*, 1397–1401. https://doi.org/10.23919/EUSIPCO55093.2022.9909873

Ge, H., Xu, K., & Ghahramani, Z. (2018). Turing: A language for flexible probabilistic inference. *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, 1682–1690. http://proceedings.mlr.press/v84/ge18b.html

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2015). *Bayesian Data Analysis* (3rd ed.). Chapman; Hall/CRC. https://doi.org/10.1201/b16018

Hoffman, M. D., & Gelman, A. (2011). *The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo*. arXiv. https://doi.org/10.48550/ARXIV.1111.4246

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2017). Automatic Differentiation Variational Inference. *Journal of Machine Learning Research*, *18*(1), 430–474. http://www.jmlr.org/papers/volume18/16-107/16-107.pdf

Nguyen, H. M. H., Erp, B. van, Senoz, I., & Vries, B. de. (2022). Efficient Model Evidence Computation in Tree-structured Factor Graphs. *2022 IEEE Workshop on Signal Processing Systems (SiPS)*, 6. https://doi.org/10.1109/SiPS55645.2022.9919250

Podusenko, A., Erp, B. van, Bagaev, D., Şenöz, İsmail, & Vries, B. de. (2021). Message Passing-Based Inference in the Gamma Mixture Model. *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, 1–6. https://doi.org/10.1109/MLSP52302.2021.9596329

Podusenko, A., Erp, B. van, Bagaev, D., şenöz, İ., & Vries, B. de. (2022). Message Passing-based Inference in Switching Autoregressive Models. *2022 30th European Signal Processing Conference (EUSIPCO)*, 1497–1501. https://doi.org/10.23919/EUSIPCO55093.2022.9909828

Podusenko, A., Erp, B. van, Koudahl, M., & Vries, B. de. (2021). AIDA: An Active Inference-based Design Agent for Audio Processing Algorithms. *arXiv:2112.13366 [Cs, Eess, Stat]*. https://doi.org/10.3389/frsip.2022.842477

Podusenko, A., Kouw, W. M., & Vries, B. de. (2021). Message Passing-Based Inference for Time-Varying Autoregressive Models. *Entropy*, *23*(6), 683. https://doi.org/10.3390/e23060683

Revels, J., Lubin, M., & Papamarkou, T. (2016). Forward-Mode Automatic Differentiation in Julia. *arXiv:1607.07892 [Cs]*. http://arxiv.org/abs/1607.07892

Salimans, T., Kingma, D. P., & Welling, M. (n.d.). Markov Chain Monte Carlo and Variational Inference:Bridging the Gap. *Bridging the Gap*, 9.

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press. ISBN: 978-0-415-55809-9

Şenöz, İ., Laar, T. van de, Bagaev, D., & Vries, B. de. (2021). Variational Message Passing and Local Constraint Manipulation in Factor Graphs. *Entropy*, *23*(7), 807. https://doi.org/10.3390/e23070807

Stan Development Team. (2022). *Stan modeling language users guide and reference manual, version 2.31*. https://mc-stan.org

Stan.jl Development Team. (2022). *Stan modeling language in julia, version 10.3.2*. https://github.com/StanJulia/Stan.jl