The Journal of Open Source Software

LINFA: a Python library for variational inference with normalizing flow and annealing

Yu Wang[1], **Emma R. Cobian**[1], **Jubilee Lee**[1], **Fang Liu**[1], **Jonathan D. Hauenstein**[1], **and Daniele E. Schiavazzi**[1¶]

1 Department of Applied and Computational Mathematics and Statistics, University of Notre Dame, Notre Dame, IN 46556, USA. ¶ Corresponding author

DOI: 10.21105/joss.06309

Software

- Review
- Repository
- Archive

Editor: Oskar Laverny

Reviewers:

- @robmoss
- @selimfirat

Submitted: 28 November 2023
Published: 05 April 2024

Summary

Variational inference is an increasingly popular method in statistics and machine learning for approximating probability distributions. We developed LINFA (Library for Inference with Normalizing Flow and Annealing), a Python library for variational inference to accommodate computationally expensive models and difficult-to-sample distributions with dependent parameters. We discuss the theoretical background, capabilities, and performance of LINFA in various benchmarks. LINFA is publicly available on GitHub at https://github.com/desResLab/LINFA.

Statement of need

Generating samples from a posterior distribution is a fundamental task in Bayesian inference. The development of sampling-based algorithms from the Markov chain Monte Carlo family (Gelfand & Smith, 1990; Geman & Geman, 1984; Hastings, 1970; Metropolis et al., 1953) has made solving Bayesian inverse problems accessible to a wide audience of both researchers and practitioners. However, the number of samples required by these approaches is typically significant and the convergence of Markov chains to their stationary distribution can be slow especially in high-dimensions. Additionally, satisfactory convergence may not be always easy to quantify, even if a number of metrics have been proposed in the literature over the years. More recent paradigms have been proposed in the context of variational inference (Wainwright et al., 2008), where an optimization problem is formulated to determine the optimal member of a parametric family of distributions that can approximate a target posterior density. In addition, flexible approaches to parametrize variational distributions through a composition of transformations (closely related to the concept of *trasport maps*, see, e.g., Villani & others (2009)) have reached popularity under the name of *normalizing flows* (Dinh et al., 2016; Kingma et al., 2016; Kobyzev et al., 2020; Papamakarios et al., 2021; Rezende & Mohamed, 2015). The combination of variational inference and normalizing flow has received significant recent interest in the context of general algorithms for solving inverse problems (El Moselhy & Marzouk, 2012; Rezende & Mohamed, 2015).

However, cases where the computational cost of evaluating the underlying probability distribution is significant occur quite often in engineering and applied sciences, for example when such evaluation requires the solution of an ordinary or partial differential equation. In such cases, inference can easily become intractable. Additionally, strong and nonlinear dependence between model parameters may results in difficult-to-sample posterior distributions characterized by features at multiple scales or by multiple modes. The LINFA library is specifically designed for cases where the model evaluation is computationally expensive. In such cases, the construction of an adaptively trained surrogate model is key to reducing the computational cost of inference (Wang et al., 2022). In addition, LINFA provides an adaptive annealing

scheduler, where temperature increments are automatically determined based on the available variational approximant of the posterior distribution. Thus, adaptive annealing makes it easier to sample from complicated densities (Cobian et al., 2023).

Capabilities

LINFA is designed as a general inference engine and allows the user to define custom input transformations, computational models, surrogates, and likelihood functions.

1. **User-defined input parameter transformations** - Input transformations may reduce the complexity of inference and surrogate model construction in situations where the ranges of the input variables differ substantially or when the input parameters are bounded. A number of pre-defined univariate transformations are provided, i.e, `identity`, `tanh`, `linear`, and `exp`. These transformations are independently defined for each input variable, using four parameters (a, b, c, d), providing a nonlinear transformation between the *normalized* interval $[a, b]$ and the *physical* interval $[c, d]$. Additional transformations can be defined by implementing the following member functions.
    - `forward` - It evaluates the transformation from the normalized to the physical space. One transformation needs to be defined for each input dimension. For example, the list of lists

      ```
      trsf_info = [['tanh',-7.0,7.0,100.0,1500.0],
                   ['tanh',-7.0,7.0,100.0,1500.0],
                   ['exp',-7.0,7.0,1.0e-5,1.0e-2]]
      ```

      defines a hyperbolic tangent transformation for the first two variables and an exponential transformation for the third.
    - `compute_log_jacob_func` - This is the log Jacobian of the transformation that needs to be included in the computation of the log posterior density to account for the additional change in volume.
2. **User-defined computational models** - LINFA can accommodate any type of models from analytically defined posteriors with the gradient computed through automatic differentiation to legacy computational solvers for which the solution gradient is not available nor easy to compute. New models are created by implementing the methods below.
    - `genDataFile` - This is a pre-processing function used to generate synthetic observations. It computes the model output corresponding to the default parameter values (usually defined as part of the model) and adds noise with a user-specified distribution. Observations will be stored in a file and are typically assigned to `model.data` so they are available for computing the log posterior.
    - `solve_t` - This function solves the model for multiple values of the *physical* input parameters specified in a matrix format (with one sample for each row and one column for each input parameter dimension).
3. **User-defined surrogate models** - For computational models that are too expensive for online inference, LINFA provides functionalities to create, train, and fine-tune a *surrogate model*. The `Surrogate` class implements the following functionalities:
    - A new surrogate model can be created using the `Surrogate` constructor.
    - `limits` (i.e. upper and lower bounds) are stored as a list of lists using the format `[[low_0, high_0], [low_1, high_1], ...]`.
    - A *pre-grid* is defined as an a priori selected point cloud created inside the hyper-rectangle defined by `limits`. The pre-grid can be either of type `'tensor'` (tensor

product grid) where the grid order (number of points in each dimension) is defined through the argument `gridnum`, or of type `'sobol'` (using low-discrepancy quasi-random Sobol' sequences, see Sobol' (1967)), in which case the variable `gridnum` defines the total number of samples.

- Surrogate model Input/Output. The two functions `surrogate_save()` and `surrogate_load()` are provided to save a snapshot of a given surrogate or to read it from a file.
- The `pre_train()` function is provided to perform an initial training of the surrogate model on the pre-grid. In addition, the `update()` function is also available to re-train the model once additional training examples are available.
- The `forward()` function evaluates the surrogate model at multiple input realizations. If a transformation is defined, the surrogate should always be specified in the *normalized domain* with limits defined in terms of the normalized intervals (i.e., $[a, b]$).

4. **User-defined likelihood** - A user-defined likelihood function can be defined by passing the parameters, the model, the surrogate and a coordinate transformation using

   `log_density(x, model, surrogate, transformation),`

   and then assigning it as a member function of the `experiment` class using:

   `exp.model_logdensity = lambda x: log_density(x, model, surr, transf).`

5. **Linear and adaptive annealing schedulers** - LINFA provides two annealing schedulers by default. The first is the `'Linear'` scheduler with constant increments. The second is the `'AdaAnn'` adaptive scheduler (Cobian et al., 2023) with hyperparameters reported in Table 7. For the AdaAnn scheduler, the user can also specify a different number of parameter updates to be performed at the initial temperature t_0, final temperature t_1, and for any temperature $t_0 < t < 1$. Finally, the batch size (number of samples used to evaluate the expectations in the loss function) can also be differentiated for $t = 1$ and $t < 1$.

6. **User-defined hyperparameters** - A complete list of hyperparameters with a description of their functionality can be found in the Appendix.

Related software modules and packages for variational inference

Other Python modules and packages were found to provide an implementation of variational inference with a number of additional features. An incomplete list of these packages is reported below.

- PyMC (Abril-Pla et al., 2023).
- BayesPy (Luttinen, 2016) (with an accompanying paper, BayesPy: Variational Bayesian Inference in Python).
- Pyro (Bingham et al., 2019) (with some examples).
- PyVBMC (Huggins et al., 2023) with accompanying JOSS article.
- Online notebooks (see this example) which implement variational inference from scratch in `pytorch`.

LINFA is based on normalizing flow transformations and therefore can infer non linear parameter dependence. It also provides the ability to adaptively train a surrogate model (NoFAS) which significantly reduces the computational cost of inference for the parameters of expensive computational models. Finally, LINFA provides an adaptive annealing algorithm (AdaAnn) which autonomously selects the appropriate annealing steps based on the current approximation of the posterior distribution.

Numerical benchmarks

We tested LINFA on multiple problems. These include inference on unimodal and multi-modal posterior distributions specified in closed form, ordinary differential models and dynamical systems with gradients directly computed through automatic differentiation in PyTorch, identifiable and non-identifiable physics-based models with fixed and adaptive surrogates, and high-dimensional statistical models. Some of the above tests are included with the library and systematically tested using GitHub Actions. A detailed discussion of these test cases is provided in the Appendix. To run the test type

```
python -m unittest linfa.linfa_test_suite.NAME_example
```

where `NAME` is the name of the test case, either `trivial`, `highdim`, `rc`, `rcr`, `adaann` or `rcr_nofas_adaann`.

Conclusion and Future Work

In this paper, we have introduced the LINFA library for variational inference, briefly discussed the relevant background, its capabilities, and report its performance on a number of test cases. Some interesting directions for future work are mentioned below.

Future versions will support user-defined privacy-preserving synthetic data generation and variational inference through differentially private gradient descent algorithms. This will allow the user to perform inference tasks while preserving a pre-defined privacy budget, as discussed in (Su et al., 2023). LINFA will also be extended to handle multiple models. This will open new possibilities to solve inverse problems combining variational inference and multi-fidelity surrogates (see, e.g., Siahkoohi et al. (2021)). In addition, for inverse problems with significant dependence among the parameters, it is often possible to simplify the inference task by operating on manifolds of reduced dimensionality (Brennan et al., 2020). New modules for dimensionality reduction will be developed and integrated with the LINFA library. Finally, the ELBO loss typically used in variational inference has known limitations, some of which are related to its close connection with the KL divergence. Future versions of LINFA will provide the option to use alternative losses.

Acknowledgements

The authors gratefully acknowledge the support by the NSF Big Data Science & Engineering grant #1918692 and the computational resources provided through the Center for Research Computing at the University of Notre Dame. DES also acknowledges support from NSF CAREER grant #1942662.

Appendix

Background theory

Variational inference with normalizing flow

Consider the problem of estimating (in a Bayesian sense) the parameters $z \in \mathcal{Z}$ of a physics-based or statistical model

$$x = f(z) + \varepsilon,$$

from the observations $x \in \mathcal{X}$ and a known statistical characterization of the error ε. We tackle this problem with variational inference and normalizing flow. A normalizing flow (NF) is a nonlinear transformation $F : \mathbb{R}^d \times \Lambda \to \mathbb{R}^d$ designed to map an easy-to-sample *base*

distribution $q_0(z_0)$ into a close approximation $q_K(z_K)$ of a desired target posterior density $p(z|x)$. This transformation can be determined by composing K bijections

$$z_K = F(z_0) = F_K \circ F_{K-1} \circ \cdots \circ F_k \circ \cdots \circ F_1(z_0),$$

and evaluating the transformed density through the change of variable formula (see Villani & others (2009)).

In the context of variational inference, we seek to determine an *optimal* set of parameters $\lambda \in \Lambda$ so that $q_K(z_K) \approx p(z|x)$. Given observations $x \in \mathcal{X}$, a likelihood function $l_z(x)$ (informed by the distribution of the error ε) and prior $p(z)$, a NF-based approximation $q_K(z)$ of the posterior distribution $p(z|x)$ can be computed by maximizing the lower bound to the log marginal likelihood $\log p(x)$ (the so-called *evidence lower bound* or ELBO), or, equivalently, by minimizing a *free energy bound* (see, e.g., Rezende & Mohamed (2015)).

$$\begin{aligned} \mathcal{F}(x) &= \mathbb{E}_{q_K(z_K)}\left[\log q_K(z_K) - \log p(x, z_K)\right] \\ &= \mathbb{E}_{q_0(z_0)}[\log q_0(z_0)] - \mathbb{E}_{q_0(z_0)}[\log p(x, z_K)] - \mathbb{E}_{q_0(z_0)}\left[\sum_{k=1}^{K} \log \left|\det \frac{\partial z_k}{\partial z_{k-1}}\right|\right]. \end{aligned} \tag{1}$$

For computational convenience, normalizing flow transformations are selected to be easily invertible and their Jacobian determinant can be computed with a cost that grows linearly with the problem dimensionality. Approaches in the literature include RealNVP (Dinh et al., 2016), GLOW (Kingma & Dhariwal, 2018), and autoregressive transformations such as MAF (Papamakarios et al., 2017) and IAF (Kingma et al., 2016). Detailed reviews on a wide range of flow formulations can be found in Kobyzev et al. (2020) and Papamakarios et al. (2021).

MAF and RealNVP

LINFA implements two widely used normalizing flow formulations, MAF (Papamakarios et al., 2017) and RealNVP (Dinh et al., 2016). MAF belongs to the class of *autoregressive* normalizing flows. Given the latent variable $z = (z_1, z_2, \dots, z_d)$, it assumes $p(z_i|z_1, \dots, z_{i-1}) = \phi[(z_i - \mu_i)/e^{\alpha_i}]$, where ϕ is the standard normal distribution, $\mu_i = f_{\mu_i}(z_1, \dots, z_{i-1})$, $\alpha_i = f_{\alpha_i}(z_1, \dots, z_{i-1})$, $i = 1, 2, \dots, d$, and f_{μ_i} and f_{α_i} are masked autoencoder neural networks (MADE, Germain et al. (2015)). In a MADE autoencoder the network connectivities are multiplied by Boolean masks so the input-output relation maintains a lower triangular structure, making the computation of the Jacobian determinant particularly simple. MAF transformations are then composed of multiple MADE layers, possibly interleaved by batch normalization layers (Ioffe & Szegedy, 2015), typically used to add stability during training and increase network accuracy (Papamakarios et al., 2017).

RealNVP is another widely used flow where, at each layer the first d' variables are left unaltered while the remaining $d - d'$ are subject to an affine transformation of the form $\hat{z}_{d'+1:d} = z_{d'+1:d} \odot e^{\alpha} + \mu$, where $\mu = f_\mu(z_{1:d'})$ and $\alpha = f_\alpha(z_{d'+1:d})$ are MADE autoencoders. In this context, MAF could be seen as a generalization of RealNVP by setting $\mu_i = \alpha_i = 0$ for $i \leq d'$ (Papamakarios et al., 2017).

Normalizing flow with adaptive surrogate (NoFAS)

LINFA is designed to accommodate black-box models $f : \mathcal{Z} \to \mathcal{X}$ between the random inputs $z = (z_1, z_2, \cdots, z_d)^T \in \mathcal{Z}$ and the outputs $(x_1, x_2, \cdots, x_m)^T \in \mathcal{X}$, and assumes n observations $x = \{x_i\}_{i=1}^n \subset \mathcal{X}$ to be available. Our goal is to infer z and to quantify its uncertainty given x. We embrace a variational Bayesian paradigm and sample from the posterior distribution $p(z|x) \propto \ell_z(x, f)\, p(z)$, with prior $p(z)$ via normalizing flows.

This requires the evaluation of the gradient of the ELBO (1) with respect to the NF parameters λ, replacing $p(x, z_K)$ with $p(x|z_K)\, p(z) = \ell_{z_K}(x, f)\, p(z)$, and approximating the expectations

with their MC estimates. However, the likelihood function needs to be evaluated at every MC realization, which can be costly if the model $f(z)$ is computationally expensive. In addition, automatic differentiation through a legacy (e.g. physics-based) solver may be an impractical, time-consuming, or require the development of an adjoint solver.

Our solution is to replace the model f with a computationally inexpensive surrogate $\widehat{f} : \mathcal{Z} \times \mathcal{W} \to \mathcal{X}$ parameterized by the weigths $w \in \mathcal{W}$, whose derivatives can be obtained at a relatively low computational cost, but intrinsic bias in the selected surrogate formulation, a limited number of training examples, and locally optimal w can compromise the accuracy of $\widehat{f}$.

To resolve these issues, LINFA implements NoFAS, which updates the surrogate model adaptively by smartly weighting the samples of z from NF thanks to a *memory-aware* loss function. Once a newly updated surrogate is obtained, the likelihood function is updated, leading to a new posterior distribution that will be approximated by VI-NF, producing, in turn, new samples for the next surrogate model update, and so on. Additional details can be found in Wang et al. (2022).

Adaptive Annealing

Annealing is a technique to parametrically smooth a target density to improve sampling efficiency and accuracy during inference. In the discrete case, this is achieved by incrementing an *inverse temperature* t_k and setting $p_k(z, x) = p^{t_k}(z, x),$ for $k = 0, \dots, K$, where $0 < t_0 < \cdots < t_K \leq 1$. The result of exponentiation produces a smooth unimodal distribution for a sufficiently small t_0, recovering the target density as t_k approaches 1. In other words, annealing provides a continuous deformation from an easier to approximate unimodal distribution to a desired target density.

A linear annealing scheduler with fixed temperature increments is often used in practice (see, e.g., Rezende & Mohamed (2015)), where $t_j = t_0 + j(1 - t_0)/K$ for $j = 0, \dots, K$ with constant increments $\epsilon = (1 - t_0)/K$. Intuitively, small temperature changes are desirable to carefully explore the parameter spaces at the beginning of the annealing process, whereas larger changes can be taken as t_k increases, after annealing has helped to capture important features of the target distribution (e.g., locating all the relevant modes).

The AdaAnn scheduler determines the increment ϵ_k that approximately produces a pre-defined change in the KL divergence between two distributions annealed at~t_k and $t_{k+1} = t_k + \epsilon_k$, respectively. Letting the KL divergence equal a constant $\tau^2/2$, where τ is referred to as the *KL tolerance*, the step size ϵ_k becomes

$$\epsilon_k = \tau / \sqrt{\mathbb{V}_{p^{t_k}}[\log p(z, x)]}. \tag{2}$$

The denominator is large when the support of the annealed distribution $p^{t_k}(z, x)$ is wider than the support of the target $p(z, x)$, and progressively reduces with increasing t_k. Further detail on the derivation of the expression for ϵ_k can be found in Cobian et al. (2023).

Numerical benchmarks

Simple two-dimensional map with Gaussian likelihood

A model $f : \mathbb{R}^2 \to \mathbb{R}^2$ is chosen in this experiment having the closed-form expression

$$f(z) = f(z_1, z_2) = (z_1^3/10 + \exp(z_2/3), z_1^3/10 - \exp(z_2/3))^T.$$

Observations x are generated as

$$x = x^* + 0.05\,|x^*| \odot x_0, \tag{3}$$

where $x_0 \sim \mathcal{N}(0, I_2)$ and $\odot$ is the Hadamard product. We set the *true* model parameters at $z^* = (3, 5)^T$, with output $x^* = f(z^*) = (7.99, -2.59)^T$, and simulate 50 sets of observations from (3). The likelihood of z given x is assumed Gaussian, and we adopt a noninformative uniform prior $p(z)$. We allocate a budget of $4 \times 4 = 16$ model solutions to the pre-grid and use the rest to adaptively calibrate $\hat{f}$ using 2 samples every 1000 normalizing flow iterations.

Results in terms of loss profile, variational approximation, and posterior predictive distribution are shown in Figure 1.

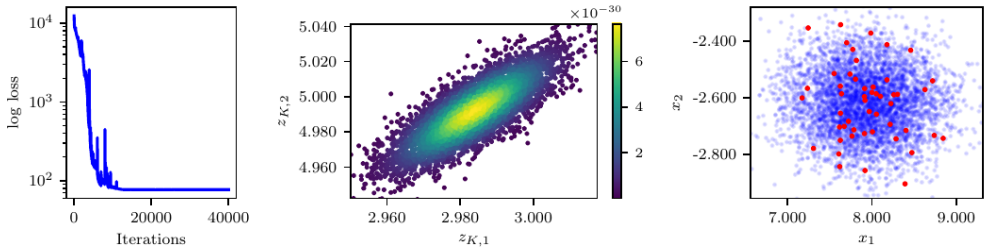


Figure 1: Results from the simple two-dimensional map. Loss profile (left), posterior samples (center), and posterior predictive distribution (right).

High-dimensional example

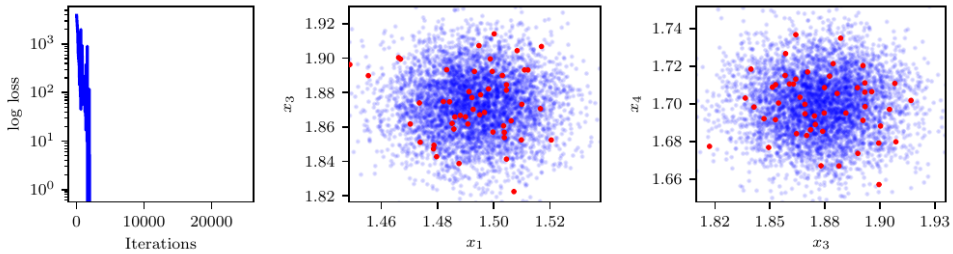
We consider a map $f : \mathbb{R}^5 \to \mathbb{R}^4$ expressed as

$$f(z) = A\, g(e^z),$$

where $g_i(r) = (2 \cdot |2\, a_i - 1| + r_i)/(1 + r_i)$ with $r_i > 0$ for $i = 1, \ldots, 5$ is the *Sobol'* function (Sobol', 2003) and A is a 4×5 matrix. We also set

$$a = (0.084, 0.229, 0.913, 0.152, 0.826)^T \text{ and } A = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

The true parameter vector is $z^* = (2.75, -1.5, 0.25, -2.5, 1.75)^T$. While the Sobol' function is bijective and analytic, f is over-parameterized and non identifiabile. This is also confirmed by the fact that the curve segment $\gamma(t) = g^{-1}(g(z^*) + v\, t) \in Z$ gives the same model solution as $x^* = f(z^*) = f(\gamma(t)) \approx (1.4910, 1.6650, 1.8715, 1.7011)^T$ for $t \in (-0.0153, 0.0686]$, where $v = (1, -1, 1, -1, 1)^T$. This is consistent with the one-dimensional null-space of the matrix A. We also generate synthetic observations from the Gaussian distribution $x = x^* + 0.01 \cdot |x^*| \odot x_0$ with $x_0 \sim \mathcal{N}(0, I_5)$, and results shown in Figure 2.



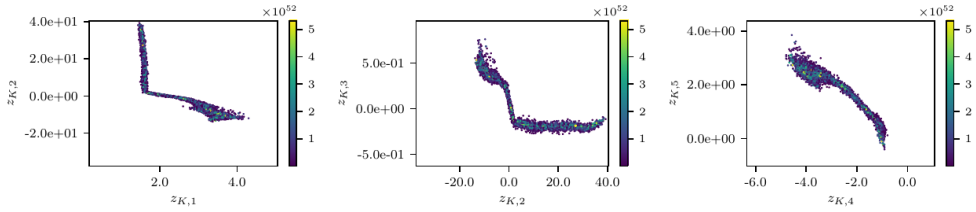


Figure 2: Results from the high-dimensional example. The top row contains the loss profile (left) and samples from the posterior predictive distribution plus the available observations (right). Samples from the posterior distribution are instead shown in the bottom row.

Two-element Windkessel Model

The two-element Windkessel model (often referred to as the *RC* model) is the simplest representation of the human systemic circulation and requires two parameters, i.e., a resistance $R \in [100, 1500]$ Barye· s/ml and a capacitance $C \in [1 \times 10^{-5}, 1 \times 10^{-2}]$ ml/Barye. We provide a periodic time history of the aortic flow (see Wang et al. (2022) for additional details) and use the RC model to predict the time history of the proximal pressure $P_p(t)$, specifically its maximum, minimum, and average values over a typical heart cycle, while assuming the distal resistance $P_d(t)$ as a constant in time, equal to 55 mmHg. In our experiment, we set the true resistance and capacitance as $z^*_{K,1} = R^* = 1000$ Barye· s/ml and $z^*_{K,2} = C^* = 5 \times 10^{-5}$ ml/Barye, and determine $P_p(t)$ from a RK4 numerical solution of the following algebraic-differential system

$$Q_d = \frac{P_p - P_d}{R}, \quad \frac{dP_p}{dt} = \frac{Q_p - Q_d}{C}, \tag{4}$$

where Q_p is the flow entering the RC system and Q_d is the distal flow. Synthetic observations are generated by adding Gaussian noise to the true model solution $x^* = (x^*_1, x^*_2, x^*_3) = (P_{p,\text{min}}, P_{p,\text{max}}, P_{p,\text{avg}}) = (78.28, 101.12, 85.75)$, i.e., x follows a multivariate Gaussian distribution with mean x^* and a diagonal covariance matrix with entries $0.05\, x^*_i$, where $i = 1, 2, 3$ corresponds to the maximum, minimum, and average pressures, respectively. The aim is to quantify the uncertainty in the RC model parameters given 50 repeated pressure measurements. We imposed a non-informative prior on R and C. Results are shown in Figure 3.

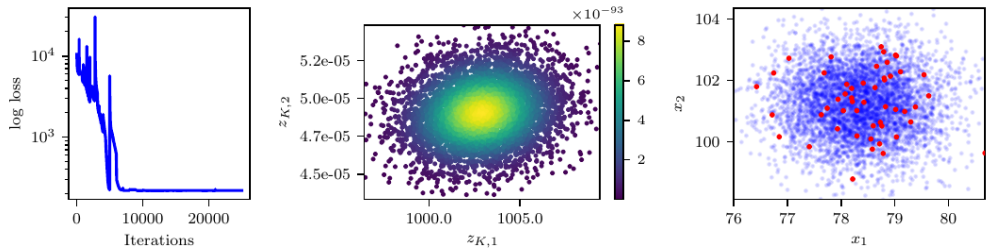


Figure 3: Results from the RC model. Loss profile (left), posterior samples (center) for R and C, and the posterior predictive distribution for $P_{p,\text{min}}$ and $P_{p,\text{max}}$ (right, $P_{p,\text{avg}}$ not shown).

Three-element Wndkessel Circulatory Model (NoFAS + AdaAnn)

The three-parameter Windkessel or *RCR* model is characterized by proximal and distal resistance parameters $R_p, R_d \in [100, 1500]$ Barye·s/ml, and one capacitance parameter $C \in [1 \times 10^{-5}, 1 \times 10^{-2}]$ ml/Barye. This model is not identifiable. The average distal pressure is only

affected by the total system resistance, i.e. the sum $R_p + R_d$, leading to a negative correlation between these two parameters. Thus, an increment in the proximal resistance is compensated by a reduction in the distal resistance (so the average distal pressure remains the same) which, in turn, reduces the friction encountered by the flow exiting the capacitor. An increase in the value of C is finally needed to restore the average, minimum and maximum pressure. This leads to a positive correlation between C and R_d.

The output consists of the maximum, minimum, and average values of the proximal pressure $P_p(t)$, i.e., $(P_{p,\text{min}}, P_{p,\text{max}}, P_{p,\text{avg}})$ over one heart cycle. The true parameters are $z^*_{K,1} = R^*_p = 1000$ Barye·s/ml, $z^*_{K,2} = R^*_d = 1000$ Barye·s/ml, and $C^* = 5 \times 10^{-5}$ ml/Barye. The proximal pressure is computed from the solution of the algebraic-differential system

$$Q_p = \frac{P_p - P_c}{R_p}, \quad Q_d = \frac{P_c - P_d}{R_d}, \quad \frac{d\,P_c}{d\,t} = \frac{Q_p - Q_d}{C},$$

where the distal pressure is set to $P_d = 55$ mmHg. Synthetic observations are generated from $N(\mu, \Sigma)$, where $\mu = (f_1(z^*), f_2(z^*), f_3(z^*))^T = (P_{p,\text{min}}, P_{p,\text{max}}, P_{p,\text{ave}})^T = (100.96, 148.02, 116.50)^T$ and Σ is a diagonal matrix with entries $(5.05, 7.40, 5.83)^T$. The budgeted number of true model solutions is 216; the fixed surrogate model is evaluated on a $6 \times 6 \times 6 = 216$ pre-grid while the adaptive surrogate is evaluated with a pre-grid of size $4 \times 4 \times 4 = 64$ and the other 152 evaluations are adaptively selected.

This example also demonstrates how NoFAS can be combined with annealing for improved convergence. The results in Figure 4 are generated using the AdaAnn adaptive annealing scheduler with intial inverse temperature $t_0 = 0.05$, KL tolerance $\tau = 0.01$ and a batch size of 100 samples. The number of parameter updates is set to 500, 5000 and 5 for t_0, t_1 and $t_0 < t < t_1$, respectively and 1000 Monte Carlo realizations are used to evaluate the denominator in equation (2). The posterior samples capture well the nonlinear correlation among the parameters and generate a fairly accurate posterior predictive distribution that overlaps with the observations. Additional details can be found in Wang et al. (2022) and Cobian et al. (2023).

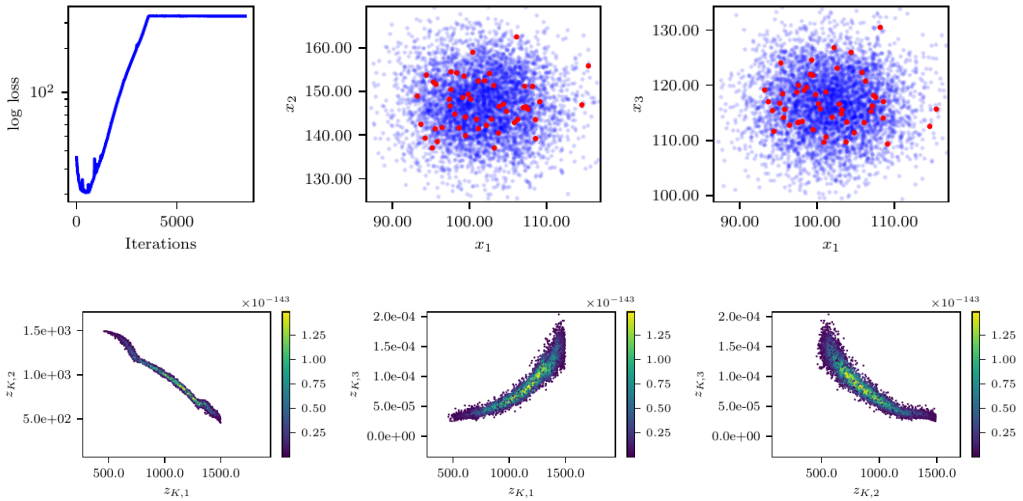


Figure 4: Results from the RCR model. The top row contains the loss profile (left) and samples from the posterior predictive distribution plus the available observations (right). Samples from the posterior distribution are instead shown in the bottom row.

True Value	Mode 1	
	Post. Mean	Post. SD
$\beta_1 = 10$	10.0285	0.1000
$\beta_2 = \pm\sqrt{20}$	4.2187	0.1719
$\beta_3 = 0.5$	0.4854	0.0004
$\beta_4 = 10$	10.0987	0.0491
$\beta_5 = 5$	5.0182	0.1142
$\beta_6 = 0$	0.1113	0.0785
$\beta_7 = 0$	0.0707	0.0043
$\beta_8 = 0$	-0.1315	0.1008
$\beta_9 = 0$	0.0976	0.0387
$\beta_{10} = 0$	0.1192	0.0463

Table 1: Posterior mean and standard deviation for positive mode in the modified Friedman test case.

Friedman 1 model (AdaAnn)

We consider a modified version of the Friedman 1 dataset (Friedman, 1991) to examine the performance of our adaptive annealing scheduler in a high-dimensional context. According to the original model in Friedman (1991), the data are generated as

$$y_i = \mu_i(\beta) + \epsilon_i, \text{ where } \mu_i(\beta) = \beta_1 \sin(\pi x_{i,1} x_{i,2}) + \beta_2 (x_{i,3} - \beta_3)^2 + \sum_{j=4}^{10} \beta_j x_{i,j}, \quad (5)$$

where $\epsilon_i \sim \mathcal{N}(0, 1)$. We made a slight modification to the model in (5) as

$$\mu_i(\beta) = \beta_1 \sin(\pi x_{i,1} x_{i,2}) + \beta_2^2 (x_{i,3} - \beta_3)^2 + \sum_{j=4}^{10} \beta_j x_{i,j}, \quad (6)$$

and set the true parameter combination to $\beta = (\beta_1, \dots, \beta_{10}) = (10, \pm\sqrt{20}, 0.5, 10, 5, 0, 0, 0, 0, 0)$. Note that both (5) and (6) contain linear, nonlinear, and interaction terms of the input variables X_1 to X_{10}, five of which (X_6 to X_{10}) are irrelevant to Y. Each X is drawn independently from $\mathcal{U}(0, 1)$. We used R package `tgp` (Gramacy, 2007) to generate a Friedman1 dataset with a sample size of n=1000. We impose a non-informative uniform prior $p(\beta)$ and, unlike the original modal, we now expect a bimodal posterior distribution of β. Results in terms of marginal statistics and their convergence for the mode with positive $z_{K,2}$ are illustrated in Table 1 and Figure 5.

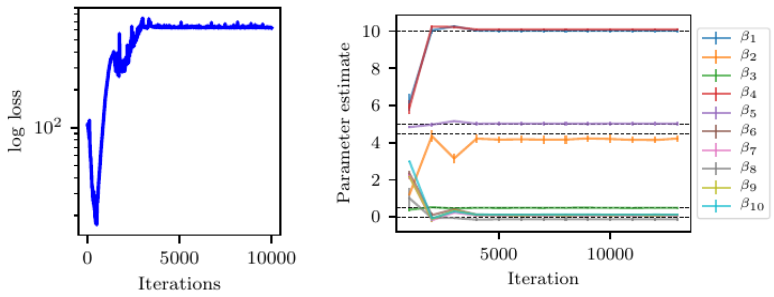


Figure 5: Loss profile (left) and posterior marginal statistics (right) for positive mode in the modified Friedman test case.

Hyperparameters in LINFA

This section contains the list of all hyperparameters in the library, their default values, and a description of the functionalities they control. General hyperparameters are listed in Table 6,

those related to the optimization process in Table 5, and to the output folder and files in Table 2. Hyperparameters for the proposed NoFAS and AdaAnn approaches are listed in Table 3 and Table 7, respectively. Finally, a hyperparameter used to select the hardware device is described in Table 4.

Table 2: Output parameters

Option	Type	Description
output_dir	string	Name of output folder where results files are saved.
log_file	string	Name of the log file which stores the iteration number, annealing temperature, and value of the loss function at each iteration.
seed	int	Seed for the random number generator.

Table 3: Surrogate model parameters (NoFAS)

Option	Type	Description
n_sample	int	Batch size used when saving results to the disk (i.e., once every `save_interval` iterations).
calibrate_interval	int	Number of NF iteration between successive updates of the surrogate model (default *1000*).
budget	int	Maximum allowable number of true model evaluations.
surr_pre_it	int	Number of pre-training iterations for surrogate model (default *40000*).
surr_upd_it	int	Number of iterations for the surrogate model update (default *6000*).
surr_folder	string	Folder where the surrogate model is stored (default `'./'`).
use_new_surr	bool	Start by pre-training a new surrogate and ignore existing surrogates (default *True*).
store_surr_interval	int	Save interval for surrogate model (*None* for no save, default *None*).

Table 4: Device parameters

Option	Type	Description
no_cuda	bool	Do not use GPU acceleration.

Table 5: Optimizer and learning rate parameters

Option	Type	Description
optimizer	string	Type of SGD optimizer (default *'Adam'*).
lr	float	Learning rate (default *0.003*).
lr_decay	float	Learning rate decay (default *0.9999*).
lr_scheduler	string	Type of learning rate scheduler (*'StepLR'* or *'ExponentialLR'*).
lr_step	int	Number of steps before learning rate reduction for the step scheduler.
log_interval	int	Number of iterations between successive loss printouts (default *10*).

Table 6: General parameters

Option	Type	Description
name	str	Name of the experiment.
flow_type	str	type of normalizing flow (*'maf'*,*'realnvp'*).
n_blocks	int	Number of normalizing flow layers (default *5*).
hidden_size	int	Number of neurons in MADE and RealNVP hidden layers (default *100*).
n_hidden	int	Number of hidden layers in MADE (default 1).
activation_fn	str	Activation function for MADE network used by MAF (default *'relu'*).
input_order	str	Input order for MADE mask creation (*'sequential'* or *'random'*, default *'sequential'*).
batch_norm_order	bool	Adds batchnorm layer after each MAF or RealNVP layer (default *True*).
save_interval	int	How often to save results from the normalizing flow iterations. Saved results include posterior samples, loss profile, samples from the posterior predictive distribution, observations, and marginal statistics.
input_size	int	Input dimensionality (default *2*).
batch_size	int	Number of samples from the basic distribution generated at each iteration (default *100*).
true_data_num	int	Number of additional true model evaluations at each surrogate model update (default *2*).
n_iter	int	Total number of NF iterations (default *25001*).

Table 7: Parameters for the adaptive annealing scheduler (AdaAnn)

Option	Type	Description
annealing	bool	Flag to activate the annealing scheduler. If this is *False*, the target posterior distribution is left unchanged during the iterations.
scheduler	string	Type of annealing scheduler (*'AdaAnn'* or *'fixed'*, default *'AdaAnn'*).
tol	float	KL tolerance. It is kept constant during inference and used in the numerator of equation (2).
t0	float	Initial inverse temperature.
N	int	Number of batch samples during annealing.
N_1	int	Number of batch samples at $t = 1$.
T_0	int	Number of initial parameter updates at t_0.
T	int	Number of parameter updates after each temperature update. During such updates the temperature is kept fixed.
T_1	int	Number of parameter updates at $t = 1$
M	int	Number of Monte Carlo samples used to evaluate the denominator in equation (2).

References

Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., & others. (2023). PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, *9*, e1516. https://doi.org/10.7717/peerj-cs.1516

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., & Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, *20*(28), 1–6.

Brennan, M., Bigoni, D., Zahm, O., Spantini, A., & Marzouk, Y. (2020). Greedy inference with structure-exploiting lazy maps. *Advances in Neural Information Processing Systems*, *33*, 8330–8342.

Cobian, E. R., Hauenstein, J. D., Liu, F., & Schiavazzi, D. E. (2023). AdaAnn: Adaptive annealing scheduler for probability density approximation. *International Journal for Uncertainty Quantification*, *13*. https://doi.org/10.1615/Int.J.UncertaintyQuantification.2022043110

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real NVP. *arXiv Preprint arXiv:1605.08803*.

El Moselhy, T. A., & Marzouk, Y. M. (2012). Bayesian inference with optimal maps. *Journal of Computational Physics*, *231*(23), 7815–7850. https://doi.org/10.1016/j.jcp.2012.07.022

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, *19*(1), 1–67. https://doi.org/10.1214/aos/1176347963

Gelfand, A. E., & Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, *85*(410), 398–409. https://doi.org/10.1080/01621459.1990.10476213

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*, 721–741. https://doi.org/10.1109/TPAMI.1984.4767596

Germain, M., Gregor, K., Murray, I., & Larochelle, H. (2015). MADE: Masked autoencoder for distribution estimation. *International Conference on Machine Learning*, 881–889.

Gramacy, R. B. (2007). Tgp: An R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models. *Journal of Statistical Software*, *19*, 1–46. https://doi.org/10.18637/jss.v019.i09

Hastings, W. K. (1970). *Monte Carlo sampling methods using Markov chains and their applications*. https://doi.org/10.1093/biomet/57.1.97

Huggins, B., Li, C., Tobaben, M., Aarnos, M. J., & Acerbi, L. (2023). PyVBMC: Efficient Bayesian inference in Python. *Journal of Open Source Software*, *8*(86), 5428. https://doi.org/10.21105/joss.05428

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, 448–456.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, *31*.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, *29*, 4743–4751.

Kobyzev, I., Prince, S. J., & Brubaker, M. A. (2020). Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(11), 3964–3979. https://doi.org/10.1109/TPAMI.2020.2992934

Luttinen, J. (2016). Bayespy: Variational Bayesian inference in Python. *The Journal of Machine Learning Research*, *17*(1), 1419–1424.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., & Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *The Journal of Machine Learning Research*, *22*(1), 2617–2680.

Papamakarios, G., Pavlakou, T., & Murray, I. (2017). Masked autoregressive flow for density estimation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/6c1da886822c67822bcf3679d04369fa-Paper.pdf

Rezende, D., & Mohamed, S. (2015). Variational inference with normalizing flows. *International Conference on Machine Learning*, 1530–1538.

Siahkoohi, A., Rizzuti, G., Louboutin, M., Witte, P., & Herrmann, F. (2021). Preconditioned training of normalizing flows for variational inference in inverse problems. *Third Symposium on Advances in Approximate Bayesian Inference*. https://openreview.net/forum?id=P9m1sMaNQ8T

Sobol', I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, *7*(4), 784–802. https://doi.org/10.1016/0041-5553(67)90144-9

Sobol', I. M. (2003). Theorems and examples on high dimensional model representation. *Reliability Engineering and System Safety*, *79*(2), 187–193. https://doi.org/10.1016/S0951-8320(02)00229-6

Su, B., Wang, Y., Schiavazzi, D. E., & Liu, F. (2023). Differentially private normalizing flows for density estimation, data synthesis, and variational inference with application to electronic health records. *arXiv Preprint arXiv:2302.05787*.

Villani, C., & others. (2009). *Optimal transport: Old and new* (Vol. 338). Springer. https://doi.org/10.1007/978-3-540-71050-9

Wainwright, M. J., Jordan, M. I., & others. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, *1*(1–2), 1–305. https://doi.org/10.1561/2200000001

Wang, Y., Liu, F., & Schiavazzi, D. E. (2022). Variational inference with NoFAS: Normalizing flow with adaptive surrogate for computationally expensive models. *Journal of Computational Physics*, *467*, 111454. https://doi.org/10.1016/j.jcp.2022.111454