Modern Integration Methods in Machine Learning

Draft

Marc Peter Deisenroth

m.deisenroth@ucl.ac.uk

University College London

March 13, 2025

Foreword

This document was created for the COMP0168 module at University College London. This document is based on the NeurIPS tutorial by Ong and Deisenroth (2020).[1]

[1] https://mml-book.github.io/slopes-expectations.html

London, February 2025
Marc Deisenroth

Contents

1 Motivation

This tutorial gives a brief overview of integration methods that are commonly used in machine learning, providing some level of detail. Throughout, we focus on integration for computing expected values of the form

$$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] := \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}, \tag{1}$$

where f is a (possibly nonlinear) function and $\boldsymbol{x}$ is a random variable. Expected values of the form (1) play a central role in machine learning. Here are some examples:

- Statistical quantities. For example, mean, variance, and moments of random variables x can be expressed as expected values (in terms of (1)):

$$\mathbb{E}_x[x] = \int xp(x)dx =: \mu \tag{2}$$

$$\mathbb{V}_x[x] = \int (x-\mu)^2 p(x)dx = \mathbb{E}_x[(x-\mu)^2] \tag{3}$$

$$M_k[x] = \int x^k p(x)dx = \mathbb{E}_x[x^k] \tag{4}$$

- Marginal likelihoods. Marginal likelihoods, which are important quantities for model selection or model training, can also be expressed as expected values. Consider a supervised learning problem with training inputs $\mathcal{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, corresponding training targets $\mathcal{Y} = \{y_1, \ldots, y_N\}$, and model parameters $\boldsymbol{\theta}$ with a corresponding prior $p(\boldsymbol{\theta})$. Then, the marginal likelihood is given by

$$p(\mathcal{Y}|\mathcal{X}) = \int \underbrace{p(\mathcal{Y}|\mathcal{X},\boldsymbol{\theta})}_{\textit{likelihood}} \underbrace{p(\boldsymbol{\theta})}_{\textit{prior}} d\boldsymbol{\theta} \tag{5}$$

$$= \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta})}[\mathcal{Y}|\mathcal{X},\boldsymbol{\theta}]. \tag{6}$$

  Writing the marginal likelihood (the predictive distribution of the training targets given the training inputs) in terms of an expected value as in (6) makes it clear that the marginal likelihood is nothing but an expected likelihood, where the expectation is taken with respect to the parameter prior $p(\boldsymbol{\theta})$.

- Predictions in a Bayesian model. Considering the same supervised setting as above, we have a parameter posterior $p(\boldsymbol{\theta}|\mathcal{X},\mathcal{Y})$. To make predictions at a test input $\boldsymbol{x}_*$ we compute the predictive distribution of the corresponding target as

$$p(y_*|\boldsymbol{x}_*,\mathcal{X},\mathcal{Y}) = \int p(y_*|\boldsymbol{x}_*,\boldsymbol{\theta}) \underbrace{p(\boldsymbol{\theta}|\mathcal{X},\mathcal{Y})}_{\textit{posterior}} d\boldsymbol{\theta} \tag{7}$$

$$= \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta}|\mathcal{X},\mathcal{Y})}[y_*|\boldsymbol{x}_*,\boldsymbol{\theta}], \tag{8}$$

  which we can again interpret as an expected value, where we take the expectation with respect to the parameter posterior $p(\boldsymbol{\theta}|\mathcal{X},\mathcal{Y})$.

The marginal likelihood (6) and the Bayesian prediction (8) resemble each other. The main difference is that for the marginal likelihood computation, we compute an expectation (of training targets) with respect to the parameter prior, whereas the predictive distribution in (8) is obtained by an expectation (of a test target) with respect to the parameter posterior.

- Bayesian experimental design and Bayesian decision theory. In Bayesian experimental design [Lindley and Smith, 1972, Chaloner and Verdinelli, 1995] and Bayesian decision theory [Lindley, 1961], we are interested in computing expected utilities of the form

$$\int U(\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta} \sim p}[U(\boldsymbol{x}, \boldsymbol{\theta})] \tag{9}$$

  in order to find optimal designs/decisions $\boldsymbol{x}$.

The fundamental computational problem to be solved in all examples above is to determine the integral in (1). In nearly all interesting cases, this integral cannot be computed analytically, and approximations are required. The remainder of this chapter looks at a range of different ways to solve (1).

2 *Numerical integration (quadrature)*

We consider solving integrals of the form

$$\int_a^b f(x) w(x) dx. \tag{10}$$

where $w(x) \geq 0$ is a weight function. We start by considering the case of $w(x) = 1$, i.e., we would solve the standard, unweighted definite integral.

In numerical integration, we normally approximate f, given a set of nodes x_n and corresponding function values $f(x_n)$, see Figure 1, using an interpolating function that is easy to integrate, e.g., a low-degree polynomial.

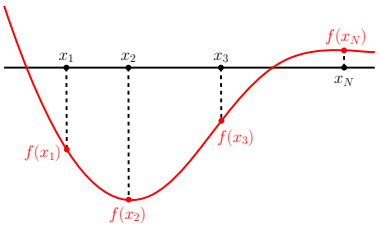


Figure 1: In numerical integration, we compute the expectation as a linear combination of function values $f(x_n)$ evaluated at nodes x_n.

Newton–Cotes approaches use low-degree polynomials and equidistant nodes x_n to solve the integration problem. For example, if we approximate f locally using a constant, we obtain the midpoint rule. The *trapezoidal rule* approximates f using piecewise linear functions, and with a quadratic approximation, we obtain the *Simpson rule*. *Gaussian quadrature* approximates f by a set of orthogonal polynomials, and the nodes x_n are the roots of these polynomials. *Bayesian quadrature* approximates f by a Gaussian process, and the nodes x_n are defined by the user.

2.1 *Newton–Cotes*

Newton–Cotes approaches use low-degree polynomials to approximate f and equidistant nodes x_n to solve the integration problem in the interval $[a, b]$; see Figure 2. In the following, we will discuss

the trapezoidal rule and Simpson's rule as important examples of Newton–Cotes in more detail.

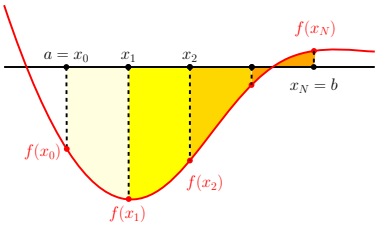


Figure 2: Newton–Cotes quadrature approximates f by a low-degree polynomial at equidistant nodes x_n. We compute the integrals for each subinterval analytically and sum them up to obtain the desired result.

Trapezoidal rule

The trapezoidal rule partitions the interval $[a, b]$ into N segments. Nodes x_n are chosen equidistantly with $|x_n - x_{n-1}| =: h$. With a locally linear approximation of f, we compute the area between two nodes as

$$\int_{x_n}^{x_{n+1}} f(x)dx \approx h\frac{1}{2}\big(f(x_n) + f(x_{n+1})\big), \tag{11}$$

which is the area of a trapezoid with corners $(x_n, x_{n+1}, f(x_{n+1}), f(x_n))$, see Figure 3. Then, we obtain the value for the integral in (10) as

$$\int_a^b f(x)dx \approx \tfrac{1}{2}h\big(f_0 + 2f_1 + \cdots + 2f_{N-1} + f_N\big), \tag{12}$$

where we used the shorthand notation $f_n := f(x_n)$.

The trapezoidal rule is straightforward, easy to implement, and fairly robust. It is a good choice, if the integrand is non-smooth, i.e., it exhibits discontinuities in its first derivative. The trapezoidal rule has an approximation error that shrinks in $\mathcal{O}(1/N^2)$, where N is the number of partitions.

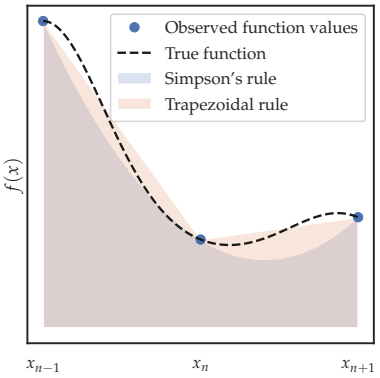


Figure 3: Locally linear (trapezoidal rule) and quadratic (Simpson's rule) approximation of the underlying function f, evaluated at nodes x_n.

Simpson's rule

Simpson's rule approximates the integrand locally with quadratic functions that connect triplets (f_n, f_{n+1}, f_{n+2}) at neighboring nodes x_n, x_{n+1}, x_{n+2}; see Figure 3. The corresponding local integral can then be approximated as

$$\int_{x_n}^{x_{n+2}} f(x)dx \approx \frac{h}{3}(f_n + 4f_{n+1} + f_{n+2}) \tag{13}$$

so that we obtain the approximation

$$\int_a^b f(x)dx \approx \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \cdots + 2f_{N-2} + 4f_{N-1} + f_N) \tag{14}$$

of the integral in (10).

Simpson's rule yields a more accurate approximation than the trapezoidal rule if the integrand f is smooth. Simpson's rule has an approximation error that shrinks in $\mathcal{O}(1/N^4)$, i.e., it converges significantly faster than the trapezoidal rule.

Example

Consider solving the integral

$$\int_0^1 \exp(-x^2 - \sin(3x)^2)dx \tag{15}$$

using the trapezoidal and Simpson rules. Figure 4 shows that Simpson's rule approximates the value of the integral in (15) well, and the error declines significantly faster than with the trapezoidal rule (as a function of the number of nodes).

Simpson's rule is the Newton–Cotes rule most often used in practice because it retains algorithmic simplicity while offering an adequate degree of approximation.

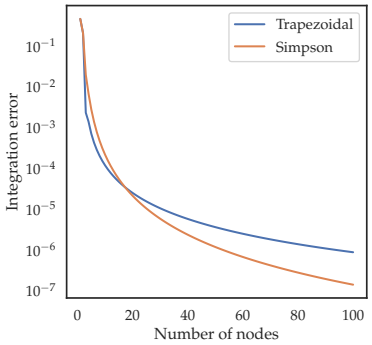


Figure 4: Comparison of trapezoidal and Simpson's rule for approximating integral (15).

2.2 *Gaussian quadrature*

Gaussian quadrature (named after Carl Friedrich Gauß) considers computing integrals of the form

$$\int_a^b f(x)w(x)dx, \quad w(x) \geq 0. \tag{16}$$

Gaussian quadrature approximates the integral by a weighted sum of function values $f(x_n)$ evaluated at nodes x_n (which are no longer need to be equidistant), i.e.,

$$\int_a^b f(x)w(x)dx \approx \sum_{n=1}^{N} w_n f(x_n), \tag{17}$$

where the nodes x_n and weights w_n, $n = 1, \ldots, N$, are chosen so that the approximation error in (17) is minimized.[2] Then, Gaussian quadrature yields an exact result when the integrated f is a polynomial of degree up to $2N - 1$.

The central idea of Gaussian quadrature is to choose the nodes x_n as the roots of a family of orthogonal polynomials. These roots can be found in a look-up table. The corresponding optimal weights w_n are also known for commonly used orthogonal polynomials. Depending on the integration bounds a, b and the choice of the weight function $w(x)$, different families of orthogonal polynomials are used. Table 1 gives an overview of the most commonly used Gaussian quadrature rules and their corresponding families of orthogonal polynomials.

[2] The weights w_n and the weight function $w(x)$ are two different quantities and $w_n \neq w(x_n)$.

$[a, b]$	$w(x)$	Orthogonal polynomial
$[-1, 1]$	1	Legendre polynomials
$[-1, 1]$	$(1 - x^2)^{-\frac{1}{2}}$	Chebychev polynomials
$[0, \infty]$	$\exp(-x)$	Laguerre polynomials
$[-\infty, \infty]$	$\exp(-x^2)$	Hermite polynomials

Table 1: Overview of Gaussian quadrature rules for commonly encountered problems [Stoer and Bulirsch, 2002]. Depending on the integration bounds a, b and the weight function $w(x)$, different orthogonal polynomials are used to achieve optimal results.

Example: Gauß–Hermite quadrature An important quadrature rule is the Gauß–Hermite quadrature. Here, we are interested in solving

$$\int f(x)\exp(-x^2)dx, \tag{18}$$

which we can also interpret as an expectation with respect to a Gaussian, i.e.,

$$\begin{aligned}\int f(x)\underbrace{\exp(-x^2)}_{w(x)}dx &= \int f(x)\frac{\sqrt{2\pi}}{\exp(-\frac{x^2}{2})}\mathcal{N}(x|0,1)dx \\ &= \sqrt{2\pi}\mathbb{E}_{x\sim\mathcal{N}(0,1)}\left[\frac{f(x)}{\exp(-\frac{x^2}{2})}\right],\end{aligned}$$

which is of high practical relevance in machine learning and many engineering disciplines. With a change of variables, we can compute expected values of function values under a general Gaussian distribution as

$$\mathbb{E}_{x\sim\mathcal{N}(\mu,\sigma^2)}[f(x)] \approx \frac{1}{\sqrt{\pi}}\sum_{n=1}^{N} w_n f(\sqrt{2}\sigma x_n + \mu). \tag{19}$$

In Gauß–Hermite quadrature, the nodes x_n are the roots of the physicists' version of the Hermite polynomial

$$H_N(x) := (-1)^n \exp\left(\frac{x^2}{2}\right)\frac{d^n}{dx^n}\exp(-x^2), \tag{20}$$

and the weights are given by

$$w_n := \frac{2^{N-1}N!\sqrt{\pi}}{N^2 H_{N-1}^2(x_n)}. \tag{21}$$

Discussion Comparing Gaussian quadrature with Newton–Cotes approaches, we see that Gaussian quadrature solves integration problems with higher accuracy at the same number of function evaluations $f(x_n)$, $n = 1,\dots,N$. In practice, a good choice of N is unknown in advance, so that one tries out different values of N. While Newton–Cotes approaches can re-use computations from smaller values of N, this does not hold for Gaussian quadrature, so that some of the computational advantages vanish. There are ways to mitigate this [Kronrod, 1965]. For a more detailed discussion, we refer to Stoer and Bulirsch [2002]. Overall, Gaussian quadrature is the method of choice for numerical integration in low dimensions. Therefore, Gaussian quadrature is often a key component in software packages to accurately solve low-dimensional integrals. Important application areas of Gaussian quadrature include computing probabilities for rectangular bivariate/trivariate Gaussian and t-distributions [Genz, 2004], marginalization of a few hyper-parameters in latent-variable models [Rue et al., 2009] or making predictions with a Gaussian process classifier [Matthews et al., 2017].

However, Gaussian quadrature also has limitations. It does not scale to more than three-dimensional inputs and it cannot deal with noisy function evaluations $f(x_n)$. To address these issues, there are alternative integration methods, such as Bayesian quadrature and Monte Carlo estimation.

2.3 *Bayesian quadrature*

In all quadrature schemes we discussed so far, we assumed that evaluating the function f at nodes $\boldsymbol{x}_n$ is cheap. That is typically the case when functions are analytic or simple lookup tables. However, we may encounter situations where evaluating a function is costly. For example, if we were to calculate the risk or expected utility of a new drug, evaluating f may require testing of a drug, which may take weeks or months. In these cases, we are interested in computing expectations (such as risks and expected utilities) with a small number of function evaluations.

Bayesian quadrature [O'Hagan, 1991] addresses this issue by formulating quadrature as a statistical inference problem:[3] We are interested in finding something out about the integral value

$$Z := \int f(\boldsymbol{x}_*) p(\boldsymbol{x}_*) d\boldsymbol{x}_* = \mathbb{E}_{\boldsymbol{x}_* \sim p}[f(\boldsymbol{x}_*)] \tag{22}$$

using previously observed data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, where $y_n = f(x_n) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. Bayesian quadrature places a Gaussian process (GP) prior on f and uses Bayes' theorem to determine a posterior distribution on f, which induces a posterior distribution on the value Z of the integral [Diaconis, 1988, O'Hagan, 1991, Rasmussen and Ghahramani, 2003].[4]

[3] The research area of *probabilistic numerics* very much exploits the close relationship between statistical inference and quadrature [Hennig et al., 2015, Briol et al., 2015].

[4] A brief introduction to Gaussian processes is given in Appendix A.

Mean and variance of the integral value The GP on f in (22) induces a distribution $p(Z) = \mathcal{N}(Z|\mu_Z, \sigma_Z^2)$ on Z itself.[5] The expected value of Z in (22) is

$$\mu_Z = \mathbb{E}_f[Z] = \mathbb{E}_{f,\boldsymbol{x}_*}[f(\boldsymbol{x}_*)] = \mathbb{E}_{\boldsymbol{x}_* \sim p}\big[\mathbb{E}_{f \sim GP}[f(\boldsymbol{x}_*)|\boldsymbol{x}_*]\big] \tag{23}$$

$$= \mathbb{E}_{\boldsymbol{x}_* \sim p}[\mu_{\text{post}}(\boldsymbol{x}_*)], \tag{24}$$

[5] The integral in (22) is a linear projection onto the direction defined by $p(x)$. Therefore, the posterior is also Gaussian [Rasmussen and Ghahramani, 2003].

where we exploited the law of iterated expectations. With a GP prior $f \sim GP(0, k)$

$$\mu_{\text{post}}(\cdot) = k(\cdot, \boldsymbol{X})(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1} \boldsymbol{y} \tag{25}$$

is the GP's posterior mean function, where $\boldsymbol{X}$ and $\boldsymbol{y}$ are the training inputs and targets, respectively. Furthermore, k is the GP's kernel and $\boldsymbol{K}$ the corresponding kernel matrix. Then, we obtain the expected value of the integral Z as

$$\mu_Z = \int k(\boldsymbol{x}_*, \boldsymbol{X}) p(\boldsymbol{x}_*) d\boldsymbol{x}_* (\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1} \boldsymbol{y} = \boldsymbol{z}^\top (\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \tag{26}$$

where

$$z_n = \int k(\boldsymbol{x}_*, \boldsymbol{x}_n) p(\boldsymbol{x}_*) d\boldsymbol{x}_* = \mathbb{E}_{\boldsymbol{x}_* \sim p}[k(\boldsymbol{x}_*, \boldsymbol{x}_n)] \tag{27}$$

is a *kernel expectation*, which computes the expected covariance between $f(\boldsymbol{x}_*)$ and $f(\boldsymbol{x}_n)$.

The variance of the integral Z is given by

$$\sigma_Z^2 = \mathbb{V}_f[Z] = \iint k_{\text{post}}(\boldsymbol{x}_*, \boldsymbol{x}'_*) p(\boldsymbol{x}_*) p(\boldsymbol{x}'_*) d\boldsymbol{x}_* d\boldsymbol{x}'_*, \tag{28}$$

where $k_{\text{post}}(\cdot,\cdot)$ is the posterior covariance function

$$k_{\text{post}}(\cdot,\cdot') = k(\cdot,\cdot') - k(\cdot,\boldsymbol{X})(\boldsymbol{K}+\sigma_\varepsilon^2\boldsymbol{I})^{-1}k(\boldsymbol{X},\cdot'). \tag{29}$$

We now obtain the variance of Z (by using (29) in (28)) as

$$\begin{aligned}\sigma_Z^2 &= \iint k(\boldsymbol{x}_*,\boldsymbol{x}_*')p(\boldsymbol{x}_*)p(\boldsymbol{x}_*')d\boldsymbol{x}_*d\boldsymbol{x}_*' \\ &\quad - \underbrace{\int k(\boldsymbol{x}_*,\boldsymbol{X})p(\boldsymbol{x}_*)d\boldsymbol{x}_*}_{=\boldsymbol{z}^\top}(\boldsymbol{K}+\sigma_\varepsilon^2\boldsymbol{I})^{-1}\underbrace{\int k(\boldsymbol{X},\boldsymbol{x}_*')p(\boldsymbol{x}_*')d\boldsymbol{x}_*'}_{=\boldsymbol{z}'} \qquad (30)\\ &= \mathbb{E}_{\boldsymbol{x}_*,\boldsymbol{x}_*'}[k(\boldsymbol{x}_*,\boldsymbol{x}_*')] - \boldsymbol{z}^\top(\boldsymbol{K}+\sigma_\varepsilon^2\boldsymbol{I})^{-1}\boldsymbol{z}', \qquad (31)\end{aligned}$$

where the kernel expectations $\boldsymbol{z},\boldsymbol{z}'$ are defined in (27).

Kernel expectations Computing the kernel expectations z_n in (27) and (31) still requires solving an integration problem, but arguably this integration problem is easier to solve than the original one. Kernel expectations can be computed analytically in a few cases, e.g., when p is a Gaussian and the kernel $k(\cdot,\cdot)$ is a polynomial or an RBF kernel [Rasmussen and Ghahramani, 2003, Girard et al., 2003, Deisenroth et al., 2015]. If p is non-Gaussian (but we still use an RBF or polynomial kernel), we could use an importance-sampling scheme

$$Z = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \int \frac{f(\boldsymbol{x})p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x}, \tag{32}$$

where the GP now models $\frac{f(\boldsymbol{x})p(\boldsymbol{x})}{q(\boldsymbol{x})}$, where q is Gaussian and p arbitrary. If k is a kernel for which we cannot compute the kernel expectations (27) and (31) analytically, we may be able to resort to numerical integration (e.g., using the techniques discussed earlier) or Monte Carlo integration. Kernel expectations appear not only in

	Input distribution p	
Kernel k	Gaussian	non-Gaussian
Polynomial/RBF	analytical	analytical via importance-sampling trick
otherwise	Monte Carlo or numerical integration	Monte Carlo or numerical integration

Table 2: Overview of how kernel expectations can be computed.

Bayesian quadrature but also in kernel MMD [Gretton et al., 2012], time-series analysis with Gaussian processes [Girard et al., 2003, Ko and Fox, 2009, Deisenroth et al., 2012, Deisenroth and Mohamed, 2012, Eleftheriadis et al., 2017], deep Gaussian processes [Damianou and Lawrence, 2013, Salimbeni and Deisenroth, 2017, Salimbeni et al., 2019] or model-based reinforcement learning with GPs [Deisenroth and Rasmussen, 2011, Deisenroth et al., 2015, Cutler and How, 2015].

While the use of Gaussian process priors may seem unwieldy, they also allow numerical integration to applied to situations where observations of f are noisy, evaluating f is expensive, we wish to exploit correlations between function values, or we know something about the structure of f, which can be exploited when choosing the GP's mean and covariance functions.

2.4 *Summary*

Numerical integration methods are good for solving low-dimensional integration problems quickly and accurately. The central approximation was

$$\int_a^b f(\boldsymbol{x})d\boldsymbol{x} \approx \sum_{n=1}^{N} w_n f(\boldsymbol{x}_n). \tag{33}$$

Newton–Cotes methods use equidistant nodes x_n and low-degree polynomial approximations of f. In Gaussian quadrature, the nodes x_n are the roots of interpolating orthogonal polynomials. Bayesian quadrature formulates integration as a statistical inference problem and uses a global approximation of f by means of a GP. These methods can be applied to low and moderate-dimensional integration problems. For higher-dimensional settings (maybe up to 100–1000 dimensions), we may want to resort to Monte Carlo integration (Section 3) to get an exact solution in the limit. Table 3 gives an overview of when various integration techniques can be applied.

Dimensionality	Method
≤ 3	Gaussian quadrature
≤ 10	Bayesian quadrature
≤ 1000	Monte Carlo integration

Table 3: Overview of integration techniques based on the dimensionality of the input dimension.

However, Monte Carlo estimation can be fairly slow, especially when we have to use MCMC methods to draw samples from high-dimensional complex distributions. In this case, we may want to consider approximation techniques that will not give us the exact solution to our integral problem, but will give us an approximate solution relatively fast. Algorithms that fall into this category make approximating assumptions on p and include variational inference or the Laplace approximation.

3 *Monte Carlo integration*

Monte Carlo methods are computational techniques that make use of random numbers. We consider two typical problems:

1. Generate samples $\boldsymbol{x}^{(s)}$ from a given probability distribution $p(\boldsymbol{x})$, e.g., for simulation (generative models) or particle representations of distributions.

2. Computing expectations of functions under that distribution, i.e., we need to solve integrals of the form

$$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}. \tag{34}$$

In the context of this tutorial, we will exclusively focus on the second problem, i.e., we will avoid the (more complicated) problem of generating samples from a probability distribution. However, we refer to Iain Murray's tutorial [Murray, 2015] for an excellent overview.

The key idea behind Monte Carlo integration is to use random numbers to approximate an integral. Specifcally, Monte Carlo methods compute expectations by statistical sampling, so that

$$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)}), \quad \boldsymbol{x}^{(s)} \sim p(\boldsymbol{x}). \tag{35}$$

The Monte Carlo estimator (35) is unbiased and asymptotically consistent, i.e.,

$$\lim_{S\to\infty} \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)}) = \mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] + \epsilon, \tag{36}$$

where the error ϵ is Gaussian distributed and its variance shrinks in $\mathcal{O}(1/S)$, independent of the dimensionality.

4 *Normalizing flows*

Normalizing flows [Rezende and Mohamed, 2015] provide a great way to build complex distributions from simple distributions via a flow of successive (invertible) transformations. A great overview of flow-based models is given by Weng [2018] and Papamakarios et al. [2019].

Flow-based models are special versions of latent-variable models. Here, the distribution $p(\boldsymbol{x})$ of the data is obtained by marginalizing out a latent variable $\boldsymbol{z}$ according to

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}. \tag{37}$$

This latent-variable model consists of a prior distribution $p(\boldsymbol{z})$ on the latent variable (code) $\boldsymbol{z}$ and a "prescription" $p(\boldsymbol{x}|\boldsymbol{z})$ of how to generate data $\boldsymbol{x}$ for a given realization $\boldsymbol{z}$ of the latent variable. Instead of modeling a complicated data distribution $p(\boldsymbol{x})$ directly, which is often hard, we can parametrize the prior $p(\boldsymbol{z})$ and the generator $p(\boldsymbol{x}|\boldsymbol{z})$ as an indirect way to get $p(\boldsymbol{x})$.

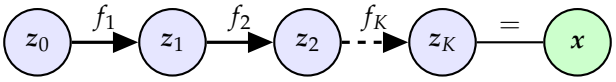


Figure 5: Normalizing flow. The normalzing flow defines a chain of invertible transformations f_k of a simple base distribution $p_0(\boldsymbol{z}_0)$ into a complex data distribution $p(\boldsymbol{x})$.

A normalizing flow is a specific way to define $p(\boldsymbol{z})$ and, more importantly, $p(\boldsymbol{x}|\boldsymbol{z})$. Focusing on $p(\boldsymbol{x}|\boldsymbol{z})$, a normalizing flow defines a chain of K transformations

$$\boldsymbol{x} = \boldsymbol{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\boldsymbol{z}_0) \tag{38}$$

with $f_k : \mathbb{R}^D \to \mathbb{R}^D$ invertible and $\boldsymbol{z}_0 \sim p_0$, where p_0 is called a *base distribution*. The path $(\boldsymbol{z}_0, \ldots, \boldsymbol{z}_K)$ of random variable $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1})$, $k = 1, \ldots, K$, is called a *flow*, and the path $(p_0, \ldots, p_K)$ of the corresponding distributions is called a *normalizing flow*. Figure 5 illustrates a normalizing flow.

Under mild assumptions, a normalizing-flow model can express any distribution, even if the base distribution p_0 is simple [Papamakarios et al., 2019].

By repeated application of the change-of-variables-trick[6], we obtain the marginal distribution at the "end" of the flow as

$$p(\boldsymbol{x}) = p(\boldsymbol{z}_K) = p(\boldsymbol{z}_0) \prod_{k=1}^{K} \left| \det \frac{df_k^{-1}(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right| = \frac{p(\boldsymbol{z}_0)}{\prod_{k=1}^{K} \left| \det \frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right|} \tag{39}$$

and the entropy can be efficiently computed as

$$\log p(\boldsymbol{x}) = \log p(\boldsymbol{z}_K) = \log p(\boldsymbol{z}_0) - \sum_{k=1}^{K} \log \left| \det \left(\frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right) \right|. \tag{40}$$

Note that we do not need to invert f_k explicitly, since we are only interested in the determinant

$$\det \left(\frac{df_k^{-1}(\boldsymbol{z}_k)}{d\boldsymbol{z}_k} \right) = \frac{1}{\det \left(\frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right)} \tag{41}$$

for which we only need the forward transformation f_k.

[6] See Appendix B for a brief introduction.

4.1 *Example*

We consider an example where $p_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. We successively apply planar flows

$$\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b). \tag{42}$$

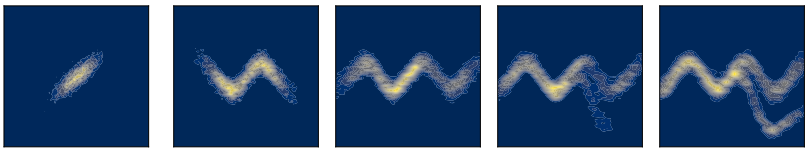


Figure 6: Illustration of a normalizing flow with 1, 2, 3, 7, and 12 planar flows; generated using a PyMC3 tutorial [Salvatier et al., 2016].

Figure 6 shows the transformed distributions after application of 1, 2, 3, 7, and 12 applications of the planar flow (42). The more often the planar flow is applied the more complex distributions we can model.

4.2 *Computing expectations*

Expectations with respect to p_K can be computed without explicitly knowing p_K (law of the unconscious statistician; LOTUS). Assume we want to compute an expected loss $\mathbb{E}_{p_K}[l(\boldsymbol{x})]$, then we can get this as

$$\mathbb{E}_{p_X}[l(\boldsymbol{x})] = \mathbb{E}_{p_K}[l(\boldsymbol{z}_K)] = \mathbb{E}_{p_0}[l(f_K \circ f_{K-1} \circ \cdots \circ f_1(\boldsymbol{z}_0))]. \tag{43}$$

We can obtain this expectation as a Monte Carlo estimate. First, we generate a sample $\boldsymbol{z}_0^{(s)} \sim p_0(\boldsymbol{z}_0)$ from the base distribution and then push that sample through $f_1, \ldots, f_K$ (ancestral sampling), which yields a valid sample $\boldsymbol{x}^{(s)} \sim p_X(\boldsymbol{x})$ from our target distribution. Using these samples, we can use Monte Carlo integration (see Section 3) to compute the expected value in (43).

4.3 *Computational aspects*

The computational challenge in normalizing flows lies in the computation of the (log-)determinant terms of the Jacobians, e.g., (40). These determinants can be computed efficiently, i.e., in linear time, if the Jacobian is diagonal or a triangular matrix, in which case the determinant is the product of the diagonal elements. To construct Jacobians that are triangular, we can define functions f_k in a particular way.

Looking at the Jacobian $df_k/dz_{k-1} = dz_k/dz_{k-1}$, $z_{k-1} \in \mathbb{R}^D$, we get

$$\begin{bmatrix} \frac{\partial z_k^{(1)}}{\partial z_{k-1}^{(1)}} & \frac{\partial z_k^{(1)}}{\partial z_{k-1}^{(2)}} & \cdots & \frac{\partial z_k^{(1)}}{\partial z_{k-1}^{(D)}} \\ \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(1)}} & \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(2)}} & \cdots & \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(D)}} \\ \vdots & & \ddots & \vdots \\ \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(1)}} & \cdots & \cdots & \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(D)}} \end{bmatrix} \in \mathbb{R}^{D \times D}. \tag{44}$$

To make this Jacobian (collection of partial derivatives) a triangular matrix, we require the partial derivatives $\frac{\partial z_k^{(d)}}{\partial z_{k-1}^{(>d)}}$ in the upper-triangular matrix (in red) to vanish.

One way to achieve this is to use an autoregressive model (applied to the dimensions of z_k) to sequentially build up z_k. In this *autoregressive flow*, each dimension d of z_k can be written as

$$z_k^{(d)} = f_k(z_{k-1}^{(1)}, \ldots, z_{k-1}^{(d)}) = f_k(z_{k-1}^{(\leq d)}). \tag{45}$$

A special example of an autoregressive flow uses the transformation

$$z_k^{(d)} = \tau(z_{k-1}^{(d)}; c_d(z_{k-1}^{(<d)})), \tag{46}$$

where τ is a *transformer* and c a *conditioner*. The conditioner parametrizes the transformer, but does itself not need to be invertible [Papamakarios et al., 2019].

Examples of autoregressive-flow models include NICE [Dinh et al., 2014], Real NVP [Dinh et al., 2017], Masked Autoregressive Flow [Papamakarios et al., 2017], Inverse Autoregressive Flow [Kingma et al., 2016], Neural Autoregressive Flows, spline flows, Block Neural Autoregressive Flows, Glow [Kingma and Dhariwal, 2018].

4.4 *Applications*

We can think of the re-parametrization trick used in variational autoencoders (VAEs) [Rezende et al., 2014, Kingma and Welling, 2014] as a special case of a normalizing flow, so that the key ideas of normalizing flows are used to perform (variational) inference in deep generative model. In a VAE, $p(z)$ is a complex posterior over latent variables z, and f transforms a simple input distribution (for example, a standard normal distribution) over x into a complex approximate posterior $q(z)$. Other application areas of normalizing flows

include graph neural networks [Liu et al., 2019], parallel WaveNet [Oord et al., 2018], but also neural ODEs [E, 2017, Chen et al., 2018]. Normalizing flows have also been generalized to flows on manifolds [Gemici et al., 2016, Rezende et al., 2020, Mathieu and Nickel, 2020].

4.5 Summary

Normalizing flows provide a constructive way to generate rich distributions. The key idea is to transform a simple distribution using a flow of successive (invertible) transformations. Key ingredient is the change-of-variables trick. From a practical (computational) perspective, Jacobians can be computed efficiently, if the transformations are defined appropriately. Normalizing flows can be used as a generator and inference mechanism.

5 Inference in time series models

As an application of integration, we consider the problem of inference in time series. We focus on discrete-time settings.

Assume a distribution $p(\boldsymbol{x}_0)$ of the initial state $\boldsymbol{x}_0$ and a Markovian state evolution

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + \boldsymbol{\epsilon}, \qquad \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}) \tag{47}$$

where f is a transition function and $p(\boldsymbol{x}_0)$ the distribution of the initial sate.[7]

[7] We make the simplifying assumption of additive Gaussian noise in (47) as this greatly simplifies some of the problems we have to solve.

We are often interested in computing an expected utility

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})], \tag{48}$$

where the expectation is taken with respect to state trajectories $\boldsymbol{\tau} := (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$. A trajectory is a sequence of states from the initial state at time 0 to a final state at time T. Examples include:

- Reinforcement learning and optimal control, where U is a long-term cost/reward function [Sutton and Barto, 1998, Bertsekas, 2005]
- Logistics, when we forecast demand and associated costs
- Weather/climate forecasts, which can be used for assessing the risk level of flooding.

The main challenge in all these scenarios is to make long-term predictions, which requires uncertainty propagation. Uncertainty could enter through uncertain initial states, noise in the system or, if we learn f, uncertainty about some model parameters.

The problem we consider is to determine a (predictive) distribution of the state $\boldsymbol{x}_t$ for $t = 1, \ldots, T$. There are multiple ways to look at this problem:

1. Iterative prediction (similar to Kalman filtering). In this *distributional perspective*, we compute marginal distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$ iteratively. In the context of this tutorial, we consider Gaussian

marginals that are parametrized by a mean $\boldsymbol{\mu}_t$ and a covariance matrix $\boldsymbol{\Sigma}_t$.

2. Trajectory sampling (done in RL quite often). In this *pathwise perspective*, we generate multiple trajectories $\boldsymbol{\tau}_i := (\boldsymbol{x}_1^{(i)}, \dots, \boldsymbol{x}_T^{(i)})$ from which we can extract the marginals $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$.

In the following, we will discuss two approaches for making long-term predictions: Deterministic and stochastic approximate inference. In deterministic approximate inference, we iteratively compute the state marginal distributions $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$, compute expected utilities $\mathbb{E}_{\boldsymbol{x}_t}[u(\boldsymbol{x}_t)]$ at every time step, and sum them up to obtain

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}[u(\boldsymbol{x}_t)] = \sum_{t=0}^{T} \int u(\boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t. \tag{49}$$

In stochastic approximate inference, we generate sample trajectories $\boldsymbol{\tau}^{(s)}$ and use Monte Carlo integration to determine the expected utility (48) via

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] \approx \frac{1}{S} \sum_{s=1}^{S} U(\boldsymbol{\tau}^{(s)}). \tag{50}$$

5.1 Deterministic approximate inference

Predicting the (marginal) distributions $p(\boldsymbol{x}_1), \dots, p(\boldsymbol{x}_T)$ of the state trajectory iteratively, requires us to repeatedly solve the following integral:

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t \tag{51}$$

$$= \int \mathcal{N}\big(\boldsymbol{x}_{t+1}\big|f(\boldsymbol{x}_t), \boldsymbol{Q}\big) p(\boldsymbol{x}_t) d\boldsymbol{x}_t. \tag{52}$$

Commonly, the marginals $p(\boldsymbol{x}_t)$ are approximated by Gaussians, i.e., $p(\boldsymbol{x}_t) \approx \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$. Classical approaches for determining an approximate Gaussian distribution include linearization[8] (e.g., extended Kalman filter), unscented transformation[9] (e.g., unscented Kalman filter), and moment matching[10] (e.g., assumed density filter).

[8] approximate f

[9] approximate $p(\boldsymbol{x}_t)$.

[10] Determine correct mean and variance of predictive distribution.

Linearization

Linearization [Smith et al., 1962, Ohab and Stubberud, 1965] exploits that a linear/affine transformation of a Gaussian distribution remains Gaussian. With a Gaussian distribution $p(\boldsymbol{x}_t) = \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$, an affine transformation of $\boldsymbol{x}_t$ via

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{b} \tag{53}$$

results in a Gaussian distribution $p(\boldsymbol{x}_{t+1}) = \mathcal{N}\big(\boldsymbol{A}\boldsymbol{\mu}_t + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}_t\boldsymbol{A}^\top\big)$.

In the context of the time-series model (47), function f is a nonlinear transformation of $\boldsymbol{x}_t$. To locally approximate f using a linear

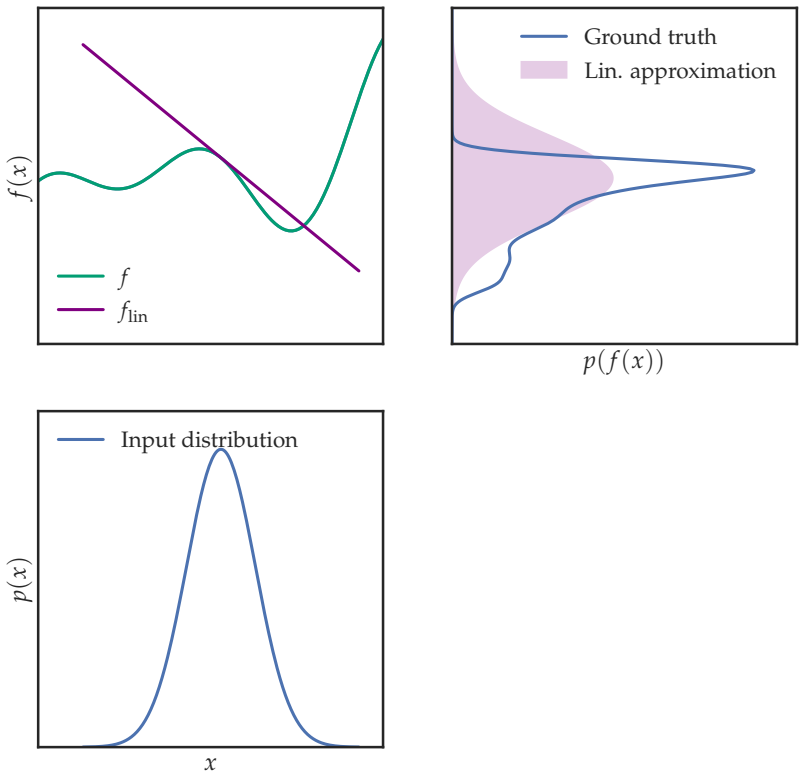


Figure 7: Illustration of linearization. Prediction by locally linearizing a nonlinear function and then analytically pushing the Gaussian through the linearized function.

function f_{lin} (thereby turning the nonlinear transformation into a linear one), we can use a first-order Taylor-series expansion around $\boldsymbol{\mu}_t$ to obtain

$$f_{\text{lin}}(\boldsymbol{x}) = f(\boldsymbol{\mu}_t) + \boldsymbol{J}(\boldsymbol{\mu}_t)(\boldsymbol{x} - \boldsymbol{\mu}_t), \tag{54}$$

where

$$\boldsymbol{J}(\boldsymbol{\mu}_t) := \left.\frac{df(\boldsymbol{x})}{d\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{\mu}_t} \tag{55}$$

is the Jacobian of f evaluated at $\boldsymbol{x} = \boldsymbol{\mu}_t$. Approximating f by f_{lin} allows us to compute an approximate Gaussian distribution of $\boldsymbol{x}_{t+1}$ as

$$p(\boldsymbol{x}_{t+1}) \approx \mathcal{N}\left(\boldsymbol{x}_{t+1} | \boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}\right) \tag{56}$$

$$\boldsymbol{\mu}_{t+1} = \mathbb{E}_{\boldsymbol{x}}[f_{\text{lin}}(\boldsymbol{x})] = f(\boldsymbol{\mu}_t) \tag{57}$$

$$\boldsymbol{\Sigma}_{t+1} = \mathbb{V}_{\boldsymbol{x}}[f_{\text{lin}}(\boldsymbol{x})] = \boldsymbol{J}(\boldsymbol{\mu}_t)\boldsymbol{\Sigma}_t\boldsymbol{J}(\boldsymbol{\mu}_t)^\top + \boldsymbol{Q}, \tag{58}$$

where the additional $\boldsymbol{Q}$ in the expression of the predictive covariance is due to the additive noise term in (47).

Computing a Gaussian approximate predictive distribution via linearization is conceptually straightforward, but it requires a differentiable transition function f. Moreover, in practice, the linearization approach tends to underestimate the true covariance, which results in overconfident predictions. This can lead to problems in downstream applications that rely on reasonable uncertainty estimates.

Computing the predictive distribution scales in $\mathcal{O}(D^3)$, where D is the dimensionality of the state. Linearization, as a way to approximate (52) is widely used in engineering. For example, the extended Kalman filter (EKF) exploits linearization to compute a posterior distribution on unobserved states. The EKF is at the core of GPS and was used in multiple Apollo missions.

Unscented transformation

An alternative approach to linearization to approximate the integral in (52) is the unscented transformation [Julier et al., 1995]. The key idea behind the unscented transformation [Julier et al., 1995] is that instead of approximating function f (which linearization does), we can approximate $p(\boldsymbol{x}_t)$. More specifically, the unscented transformation represents $p(\boldsymbol{x}_t)$ using a small set of $2D+1$ *sigma points*, which we can think of as deterministically chosen particles. We then evaluate the original function f at those sigma points and compute a (weighted) Monte Carlo estimate of the predictive mean and covariance of the mapped sigma points.[11] Figure 8 illustrates the idea of the unscented transformation.

[11] The expression "Monte Carlo estimate" is somewhat misleading in this context since neither the sigma points nor the weights are chosen at random.

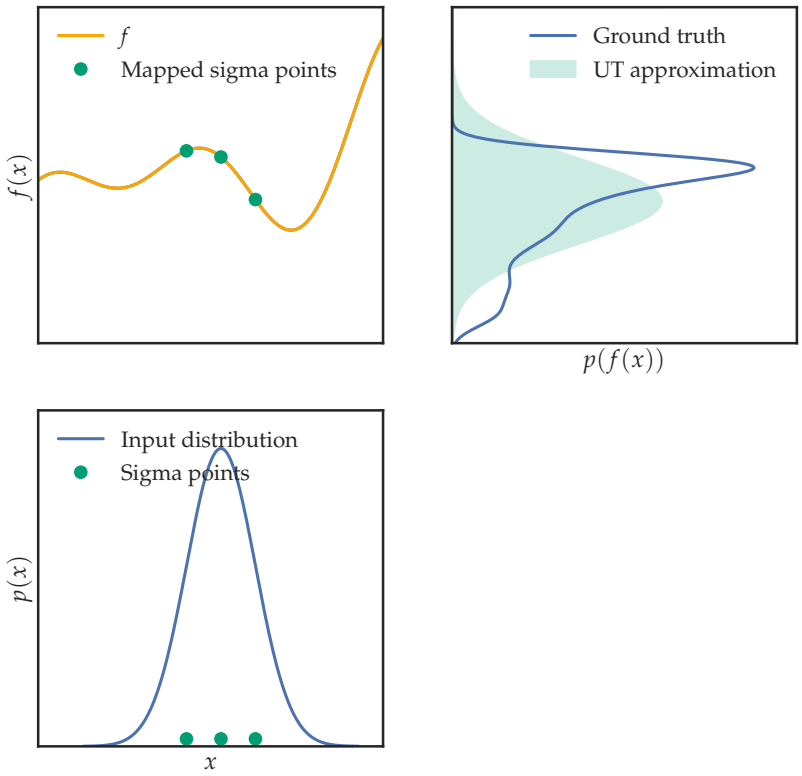


Figure 8: Illustration of unscented transformation. Prediction by approximating the input distribution by means of sigma points.

Sigma points are deterministically chosen as

$$\mathcal{X}_t = \{\boldsymbol{\mu}_t + \alpha(\sqrt{\boldsymbol{\Sigma}_t})_i, i = 1, \ldots, D\}, \tag{59}$$

where $\sqrt{\boldsymbol{\Sigma}_t}$ is a square-root of the covariance matrix (the Cholesky factor would be one option), and α spreads the sigma points symmetrically around the mean. The sigma points satisfy some nice properties, such that their mean and variance match $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ of the input distribution.

The predictive mean and covariance are then given by

$$\boldsymbol{\mu}_{t+1} \approx \sum_{d=1}^{2D+1} w_d^{\mu} f(\mathcal{X}_t^{(d)}) \tag{60}$$

$$\boldsymbol{\Sigma}_{t+1} \approx \sum_{d=1}^{2D+1} w_d^{\Sigma} (f(\mathcal{X}_t^{(d)}) - \boldsymbol{\mu}_{t+1})(f(\mathcal{X}_t^{(d)}) - \boldsymbol{\mu}_{t+1})^{\top}, \tag{61}$$

respectively, where D is the dimensionality of $\boldsymbol{x}_t$, $\mathcal{X}_t^{(d)}$ are the sigma points and w_d^{μ} and w_d^{Σ} are weights for the mean and covariance, respectively. For further details on the unscented transformation, we refer to [Julier and Uhlmann, 2004, Thrun et al., 2005].

The unscented transformation does not require an explicit calculation of the Jacobian. Furthermore, it achieves a slightly higher accuracy than linearization for the covariance estimate. The UT approximations are accurate to the third order for Gaussian inputs for all nonlinearities. For non-Gaussian inputs, approximations are accurate to at least the second order [Julier and Uhlmann, 1997]. Note that the unscented transformation is not a Monte Carlo method since the unscented transformation does not use any random numbers; sigma points are deterministically determined.

Moment matching

In moment matching, we choose a distribution that is easy for us to work with and project the true predictive distribution onto this distribution family. The moments of the approximate distribution (which lies in the distribution family of our choice) are found by minimizing the KL divergence between the true posterior and the distribution of our choice. In the end, this comes down to moment matching.

To compute the mean and covariance of $\boldsymbol{x}_{t+1}$, both linearization and the unscented transformation implicitly approximate the joint distribution $p(\boldsymbol{x}_t, \boldsymbol{x}_{t+1})$ using a Gaussian distribution [Deisenroth et al., 2012]. However, the moments of this approximate Gaussian distribution are not exact. It is sometimes possible to compute the moments of the joint distribution analytically, which then yields a better approximation in a KL sense. This means that the approximating (Gaussian) distribution is the best (unimodal) approximation to the true (unknown) distribution.

Assuming $p(\boldsymbol{x}_t)$ is Gaussian, moment matching computes the exact mean and covariance of the predictive distribution

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t \tag{62}$$

and approximates $p(\boldsymbol{x}_{t+1})$ by means of a Gaussian $\mathcal{N}(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1})$, where the mean and the covariance correspond to the mean and

the covariance of $p(\boldsymbol{x}_{t+1})$. Note that this is not true for linearization or the unscented transformation. This Gaussian approximation can then be taken as the new input distribution, so that this approximation scheme, when iterated, yields approximate Gaussian state distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$.

Given that $p(\boldsymbol{x}_t)$ is Gaussian, not all hope is lost to compute the mean and covariance of $p(\boldsymbol{x}_{t+1})$ in (62). For example, if the transition function f is a polynomial, a radial-basis function network (with Gaussian basis functions), or a Fourier series, these quantities can be computed analytically. Gradshteyn and Ryzhik [2007] provide a great overview of analytical solutions for these (and more) settings. It is also possible to use Monte Carlo integration to compute the moments as it is easy to sample from a Gaussian $p(\boldsymbol{x}_t)$.

Moment matching is used in the context of assumed density filtering [Brigo et al., 1999], expectation propagation (a message passing algorithm) [Minka, 2001, Csató et al., 2002], reinforcement learning and robotics [Deisenroth et al., 2015] and for inference in Bayesian neural networks [Hernández-Lobato and Adams, 2015, Ghosh et al., 2016].

5.2 *Stochastic approximate inference*

The expectation (48) requires averaging over state trajectories $\boldsymbol{\tau} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$. In stochastic approximate inference, we would sample trajectories $\boldsymbol{\tau}^{(s)}$ and compute a Monte Carlo estimate

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] \approx \frac{1}{S} \sum_{s=1}^{S} U(\boldsymbol{\tau}^{(s)}). \tag{63}$$

To sample a trajectory, we start with a sample $\boldsymbol{x}_0^{(s)} \sim p(\boldsymbol{x}_0)$. Subsequently, we will need to sample

$$x_t^{(s)} \sim p(\boldsymbol{x}_t | \boldsymbol{x}_0^{(s)}, \ldots, \boldsymbol{x}_{t-1}^{(s)}) = p(\boldsymbol{x}_t | \boldsymbol{x}_0^{(s)}), \tag{64}$$

where the latter equality holds for Markovian systems. Figure 9 illustrates a distribution over trajectories represented by a collection/ ensemble oft these sampled trajectories. We do not require parametric assumptions on the distribution of trajectories; however, we will have to store all samples, which may result in memory issues. Sequential Monte Carlo and particle filtering [Doucet et al., 2000, Thrun et al., 2005] rely on this kind of inference.

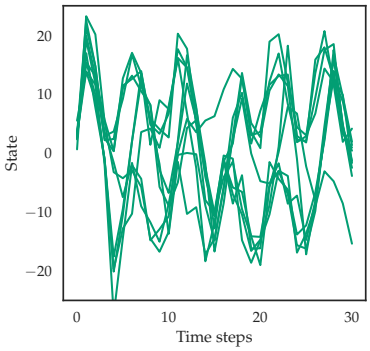


Figure 9: Time evolutions of a state. Trajectories are generated according to (64). The distribution over trajectories is represented by the collection/ ensemble of samples.

5.3 *Discussion*

Table 4 gives a brief overview of properties of deterministic and stochastic inference techniques. Deterministic approximations typically have a parametric representation of the marginals, whereas in the stochastic case, we extract the marginals from particles/samples. Deterministic approaches introduce bias, whereas stochastic ones do not. Typically, deterministic approaches are fast (they only require a single "sweep" of computations) while stochastic approaches could

	Deterministic	Stochastic
Marginal representation	Parametric	Particles
Bias	Yes	No
Time correlation	No	Yes
Speed	Fast	(Slow)
Parallelization		Easy
Memory consumption	Low	(High)
Gradients	Deterministic	Stochastic

Table 4: Properties of deterministic and stochastic approximations.

be considered slower. That said, typically, each single computation in the stochastic approach is fairly cheap and these computations can be easily parallelized. Therefore, the wall-clock time of stochastic approaches may not be too bad. Memory consumption in deterministic approaches is low compared to stochastic approaches, because one only has to store the moments of the marginals, whereas in the stochastic case one needs to keep all particles around. Looking at gradients, deterministic approximate inference gives rise to deterministic gradients, while stochastic approximate inference will invariably yield only stochastic gradients. How to efficiently deal with stochastic gradients is described in detail by Mohamed et al. [2020].

5.4 *Example: Time-series inference with Gaussian processes*

In the following, we consider the case of time-series inference where $f \sim GP$ is distributed according to a Gaussian process, i.e.,

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + \boldsymbol{\epsilon}, \qquad \boldsymbol{x}_0 \sim p(\boldsymbol{x}_0), \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}), \quad f \sim GP(m, k). \tag{65}$$

In the following, we will outline approaches to deterministic and stochastic approximate inference in these systems. Figure 10 illustrates the difference between both approaches: In sampling-based (stochastic) approaches, the GP distribution is represented by a set of sampled trajectories (green), while deterministic approaches use parametric (Gaussian) distributions to represent marginals of the GP.

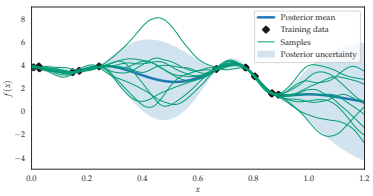


Figure 10: Samples from a GP posterior (green). The marginal Gaussian posteriors are indicated by the mean (black) and the shaded region representing the corresponding uncertainty.

Deterministic approximate inference Here, we aim to extract marginal distributions

$$p(\boldsymbol{x}_{t+1}) = \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) p(\boldsymbol{x}_t) d\boldsymbol{x}_t \tag{66}$$

$$= \iint p(f(\boldsymbol{x}_t)|f, \boldsymbol{x}_t) p(f) df\, p(\boldsymbol{x}_t) d\boldsymbol{x}_t \tag{67}$$

by means of deterministic approximations, i.e., without the use of randomness. Similarly to non-GP systems, deterministic inference can be performed by means of linearization [Ko and Fox, 2009], unscented transformation [Ko and Fox, 2009] or moment matching [Quiñonero-Candela et al., 2003, Deisenroth et al., 2009]. In all cases, the uncertainty of the GP must be taken into account when predicting the mean and covariance matrix of $\boldsymbol{x}_{t+1}$.

For example, to do moment matching with GPs, we aim to compute the mean and variance of $\boldsymbol{x}_{t+1}$. If we assume $\boldsymbol{x}_t \sim p$ and

$f \sim GP(m, k)$, then

$$\boldsymbol{\mu}_{t+1} := \mathbb{E}_{f\sim GP, \boldsymbol{x}_t \sim p}[f(\boldsymbol{x}_t)] = \mathbb{E}_{\boldsymbol{x}_t \sim p}\Big[\mathbb{E}_{f\sim GP}[f(\boldsymbol{x}_t)]\Big] = \mathbb{E}_{\boldsymbol{x}_t \sim p}[m(\boldsymbol{x}_t)], \tag{68}$$

where we exploited the law of iterated expectations and where we use m for the (posterior) mean function of the GP. Computing this expectation will require computing kernel expectations as discussed in (27) in Section 2.3.

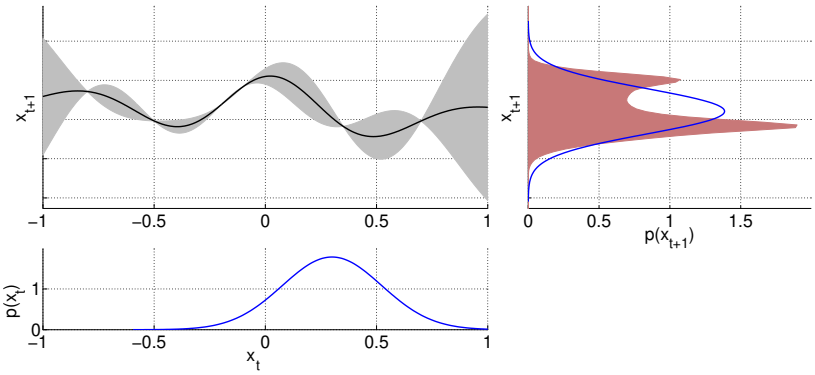


Figure 11: Moment matching with Gaussian processes. An input distribution $p(\boldsymbol{x}_t)$ is pushed through the Gaussian process. The exact predictive distribution (red) cannot be computed analytically, but the mean and the variance can be determined via kernel expectations. Then, the true predictive distribution can be approximated by a Gaussian that has the correct mean and variance.

The predictive variance is obtained as

$$\begin{aligned}\boldsymbol{\Sigma}_{t+1} &:= \mathbb{V}_{f\sim GP, \boldsymbol{x}_t \sim p}[f(\boldsymbol{x})] && (69)\\ &= \mathbb{V}_{\boldsymbol{x}_t \sim p}\Big[\mathbb{E}_{f\sim GP}[f(\boldsymbol{x}_t)]\Big] + \mathbb{E}_{\boldsymbol{x}_t \sim p}\Big[\mathbb{V}_{f\sim GP}[f(\boldsymbol{x}_t)|\boldsymbol{x}_t]\Big], && (70)\end{aligned}$$

where we used the law of total variance. Again, we will need to compute kernel expectations to compute the predictive variance. Figure 11 illustrates moment matching with Gaussian processes. More details on moment matching with GPs in dynamical systems can be found in [Deisenroth et al., 2012, 2015].

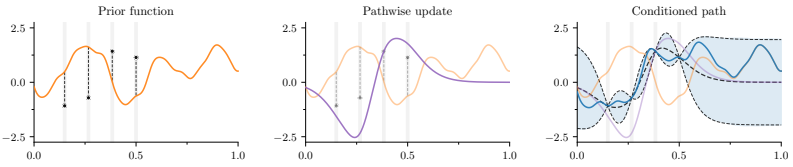
Stochastic approximate inference Instead of iteratively computing marginal distributions of future states, we can also follow the pathwise perspective and sample trajectories from (posterior) Gaussian processes. To ensure we get consistent trajectories from a GP posterior, we need to account for samples $\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_t^{(i)}$ when sampling $\boldsymbol{x}_{t+1}^{(i)}$. This can be done by augmenting the training dataset with these samples[12]. The issue with this approach to trajectory sampling is that it scales cubically in the length of the trajectory, when implemented naively, as it would require sampling from a T-dimensional multivariate Gaussian distribution.

[12] We will need to delete these samples from the training set once the full trajectory has been generated.

Wilson et al. [2020a] propose a more efficient way to sample trajectories from GP posteriors. The key insight is to exploit Matheron's rule, which allows us to write a posterior sample as a sum of a sample from the prior prior and a data-dependent update term:

$$\underbrace{f^{(s)}(\cdot)}_{\substack{\text{sample}\\ \text{from prior}}} + \underbrace{k(\cdot, \boldsymbol{X})(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\substack{\text{sample}\\ \text{from posterior}}}. \tag{71}$$

The data-dependent update term depends on error/residual between the prior sample and the training data $\boldsymbol{y}$. The update can be thought of as a mapping from prior to posterior. Figure 12 illustrates the decomposition of the GP posterior into a sum of the prior and a data-dependent update term.



(a) Sample from the prior (orange) ignores the data (dots).

(b) Data-dependent update (purple) of the prior sample.

(c) Sample from the posterior (blue) is obtained as the sum of the prior sample and the data-dependent update.

Figure 12: Efficiently sampling from posterior GPs. Matheron's rule allows us to express a sample from the posterior GP as the sum of a sample from the GP prior and a (deterministic) data-dependent update term. Figure from [Wilson et al., 2020a].

To increase computational efficiency, Wilson et al. [2020a] use different representations of the GP prior and the data-dependent update term. By separately representing the prior using Fourier basis functions (assuming a stationary kernel) and the update term using canonical basis functions $k(\cdot, \boldsymbol{x}_j)$, an efficient approximator of the posterior GP is obtained.[13] The data-dependent update term depends on error/residual between the prior sample and the training data $\boldsymbol{y}$. The update can be thought of as a mapping from prior to posterior. Different representations for prior and update terms can be used, e.g., random Fourier features (RFF) for the prior and finite basis-function representation for update. Then,

- Sampling from RFF prior scales linearly in the number T of test inputs.
- The update term can be computed linearly in the number T of test inputs.

Overall, functions can be sampled efficiently, i.e., linearly in the number of test/query inputs, which is a significant speedup from the original cubic scaling with a naive implementation.

Applications of this efficient sampling strategy include deep convolutional GP auto-encoders [Wilson et al., 2020b], Bayesian optimization with Thompson sampling [Wilson et al., 2020a], sampling from GPs on manifolds [Borovitskiy et al., 2020], and model-based reinforcement learning [Wilson, 2023]. It is also integrated into BOTorch, a software toolbox for Bayesian optimization [Balandat et al., 2020].

[13] The Fourier basis is well-suited for representing the prior [Rahimi and Recht, 2008], and the canonical basis is well-suited for representing the data [Burt et al., 2019].

A Gaussian processes

A GP is a rich Bayesian model that implements a distribution over functions.[14] A GP assumes that any finite collection of function values $\{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\}$ are jointly Gaussian distributed. A GP is fully defined by a mean function $m(\cdot)$ and a kernel $k(\cdot, \cdot)$, and we write $f \sim GP(m, k)$ [O'Hagan, 1978, Rasmussen and Williams, 2006]. An example of a kernel is the RBF kernel given by

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2l^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right), \tag{72}$$

where the kernel hyper-parameters are the signal variance σ_f^2 and the length-scale l. Throughout this chapter, we assume a zero prior mean function $m(\cdot) \equiv 0$. Assume N noisy function observations $y_n = f(x_n) + \varepsilon$ at input locations $\boldsymbol{x}_n$, where $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$. Instead of simply returning an approximate function value $f(\boldsymbol{x}_*)$ at a query point $\boldsymbol{x}_*$, a GP returns a posterior predictive distribution $p(f(\boldsymbol{x}_*)|\boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(f(\boldsymbol{x}_*)|\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$, where

$$\mu(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1}\boldsymbol{y} \tag{73}$$

$$\sigma^2(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma_\varepsilon^2 \boldsymbol{I})^{-1}k(\boldsymbol{X}, \boldsymbol{x}_*). \tag{74}$$

Here we define $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$, $\boldsymbol{y} = [y_1, \ldots, y_N]^\top$ as the training data and $\boldsymbol{K}$ as the kernel matrix with $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i, j = 1, \ldots, N$.

[14] Gaussian processes originated in the geostatistics and mining community under the term 'Kriging', coined after Danie Krige, whose MSc thesis [Krige, 1951] at the University of the Witwatersrand (South Africa) laid the foundations already in 1951.

B Change of variables

Integration by substitution (change of variables) plays an important role in modern machine learning methods, such as *normalizing flows* [Rezende and Mohamed, 2015] and *Neural ODEs* [E, 2017, Chen et al., 2018]. In the following, we will shed some light on the change-of-variables trick, and how it can be used in machine learning.

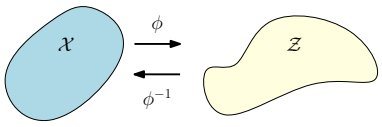


Figure 13: An invertible mapping ϕ transforms random variables $X \in \mathcal{X}$ into random variables $Z \in \mathcal{Z}$.

Consider two random variables $X \in \mathcal{X}$ and $Z \in \mathcal{Z}$ and an invertible function $\phi : \mathcal{X} \to \mathcal{Z}$, so that $\mathcal{Z} = \phi(\mathcal{X})$ (and $\mathcal{X} = \phi^{-1}(\mathcal{Z})$). For given p_X, we are interested in finding p_Z. The key idea behind this *change of variables* is to transform x into z via ϕ while keeping track of the change in distribution.

The key idea is to transform a random variable X into a random variable Z using an invertible transformation ϕ, while keeping track of the change in distribution. Then, the distribution p_X induces a distribution p_Z via ϕ, while p_Z induces a distribution p_X via ϕ^{-1}. This is also illustrated in Figure 13.

By the definition of probability density functions, it holds that

$$\int_{\mathcal{X}} p_X(x)dx = 1 = \int_{\mathcal{Z}} p_Z(z)dz \tag{75}$$

with $p_X \geq 0$ and $p_Z \geq 0$.

By transforming x into z via ϕ, the determinant of the Jacobian (derivative) $d\phi/dx$ corresponds to the scaling factor by which volumes change from dx to dz [Deisenroth et al., 2020]; see Figure 14 for

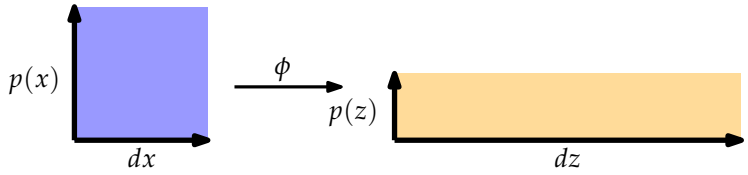


Figure 14: The determinant $|\det(dz/dx)|$ of the Jacobian tells us how much the domain dx is stretched to dz.

an illustration. The probability contained in a differential area must be preserved, i.e.,

$$\text{prob}(z \in dz \subset \mathcal{Z}) = |p_Z(z)dz| = |p_X(x)dx| = \text{prob}(x \in dx \subset \mathcal{X}). \tag{76}$$

Figure 15 illustrates this property. From (76) it follows that

$$|dz| = |dx| \left|\frac{dz}{dx}\right| = |dx| \left|\frac{d\phi(x)}{dx}\right| \tag{77}$$

$$|dx| = |dz| \left|\frac{dx}{dz}\right| = |dz| \left|\frac{d\phi^{-1}(z)}{dz}\right|. \tag{78}$$

Therefore,

$$\int_a^b p_X(x)dx \overset{(77)}{=} \int_a^b p_Z(z) \left|\frac{dz}{dx}\right| dx \tag{79}$$

$$\overset{z=\phi(x)}{=} \int_a^b p_Z(\phi(x)) \left|\frac{d\phi(x)}{dx}\right| dx = \int_{\phi(a)}^{\phi(b)} p_Z(z)dz. \tag{80}$$

Inspecting (80) closely, we performed a change of (integration) variables according to $z = \phi(x)$ and obtain a special case of the general *integration-by-substitution rule*

$$\int_a^b f(\psi(x))\psi'(x)dx = \int_{\psi(a)}^{\psi(b)} f(z)dz \tag{81}$$

with $f = p_Z$ and $\psi = \phi$. Similarly, we get

$$\int_\alpha^\beta p_Z(z)dz \overset{(78)}{=} \int_\alpha^\beta p_X(x) \left|\frac{dx}{dz}\right| dz \tag{82}$$

$$\overset{x=\phi^{-1}(z)}{=} \int_\alpha^\beta p_X(\phi^{-1}(z)) \left|\frac{d\phi^{-1}(z)}{dz}\right| dz = \int_{\phi^{-1}(\alpha)}^{\phi^{-1}(\beta)} p_X(x)dx. \tag{83}$$

Again, this equation can be directly matched with the integration-by-substitution rule in (81) with $\psi = \phi^{-1}$ and $f = p_X$.

From here, we immediately obtain the target distribution

$$p_Z(z) = p_X(\phi^{-1}(z)) \left|\frac{d\phi^{-1}(z)}{dz}\right|. \tag{84}$$

For the multivariate case, we get

$$p_Z(z) = p_X(\phi^{-1}(z)) \left|\det\left(\frac{d\phi^{-1}(z)}{dz}\right)\right| = p_X(x) \left|\det\left(\frac{d\phi}{dx}\right)\right|^{-1} \tag{85}$$

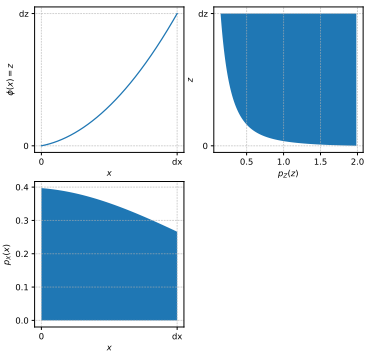


Figure 15: The change-of-variables trick preserves the area under the curve when p_X is transformed into p_Z via an invertible function ϕ.

where we exploited that $\det(\boldsymbol{J}^{-1}) = \det(\boldsymbol{J})^{-1}$ for the inverse Jacobian $\boldsymbol{J}^{-1} = \frac{d\phi^{-1}(z)}{dz}$. Note that the Jacobian $\frac{d\phi^{-1}(z)}{dz}$ exists since ϕ is invertible and ϕ^{-1} exists.

This latter trick allows us to do a forward computation of the desired density via ϕ, and it does not require an explicit inverse ϕ^{-1} to compute the Jacobian determinant.

C *Importance sampling*

Importance sampling is one way to address the problem of drawing samples from a distribution p into drawing samples from a proposal distribution q from which we can easily draw samples.

$$E_{x\sim p}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \tag{86}$$

$$= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} \tag{87}$$

$$= \int f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \tag{88}$$

$$= \mathbb{E}_{x\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] \tag{89}$$

This means, we turned the expectation w.r.t. p into an expectation w.r.t. q. If we choose q in a way that we can easily sample from it, we can approximate this last approximation by Monte Carlo:

$$\mathbb{E}_{x\sim q}\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)})\frac{p(\boldsymbol{x}^{(s)})}{q(\boldsymbol{x}^{(s)})} \tag{90}$$

$$= \frac{1}{S}\sum_{s=1}^{S} w_s f(\boldsymbol{x}^{(s)}), \tag{91}$$

where the w_s are called *importance weights*. We note that (91) closely resembles (17), so that we can interpret importance sampling as a specific instance of numerical integration, where the degree of freedom of determining the importance weights w_s lies in choosing the proposal distribution q.

The importance-sampling estimator is unbiased if $q > 0$, where $p > 0$ and if we can evaluate p. If we do not have enough samples, importance sampling breaks down in the sense that it puts nearly all weights on a single sample. This degeneracy is a well-studied problem, e.g., in the particle filtering (sequential Monte Carlo) literature [Thrun et al., 2005]. Especially in high dimensions, many draws from the proposal density q are required, so that importance sampling only really works in low/moderate-dimensional problems.

References

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew G. Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems*, 2020.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 3rd edition, 2005.

Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc P. Deisenroth. Matern Gaussian Processes on Riemannian Manifolds. In *Advances in Neural Information Processing Systems*, 2020.

Damiano Brigo, Bernard Hanzon, and François Le Gland. Approximate Nonlinear Filtering by Projection on Exponential Manifolds of Densities. *Bernoulli*, 5(3):495–534, 1999.

François-Xavier Briol, Chris Oates, Mark Girolami, and Michael A. Osborne. Frank–Wolfe Bayesian Quadrature: Probabilistic Integration with Theoretical Guarantees. In *Advances in Neural Information Processing Systems*, 2015.

David Burt, Carl Edward Rasmussen, and Mark Van Der Wilk. Rates of Convergence for Sparse Variational Gaussian Process Regression. In *Proceedings of the International Conference on Machine Learning*, 2019.

Kathryn Chaloner and Isabella Verdinelli. Bayesian Experimental Design: A Review. *Statistical Science*, 10:273–304, 1995.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, 2018.

Lehel Csató, Manfred Opper, and Ole Winther. TAP Gibbs Free Energy, Belief Propagation and Sparsity. In *Advances in Neural Information Processing Systems*, 2002.

Mark Cutler and Jonathan P. How. Efficient Reinforcement Learning for Robots using Informative Simulated Priors. In *Proceedings of the International Conference on Robotics and Automation*, 2015.

Andreas Damianou and Neil D. Lawrence. Deep Gaussian Processes. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2013.

Marc P. Deisenroth and Shakir Mohamed. Expectation Propagation in Gaussian Process Dynamical Systems. In *Advances in Neural Information Processing Systems*, 2012.

Marc P. Deisenroth and Carl E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the International Conference on Machine Learning*, 2011.

Marc P. Deisenroth, Carl E. Rasmussen, and Jan Peters. Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7–9):1508–1524, 2009.

Marc P. Deisenroth, Ryan Turner, Marco Huber, Uwe D. Hanebeck, and Carl E. Rasmussen. Robust Filtering and Smoothing with Gaussian Processes. *IEEE Transactions on Automatic Control*, 57(7): 1865–1871, 2012.

Marc P. Deisenroth, Dieter Fox, and Carl E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2): 408–423, 2015.

Marc P. Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. URL `https://mml-book.com`.

Persi Diaconis. Bayesian Numerical Analysis. *Statistical Decision Theory and Related Topics IV*, 1:163–175, 1988.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation. *arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density Estimation Using Real NVP. In *Proceedings of the International Conference on Learning Representation*, 2017.

Arnaud Doucet, Simon J. Godsill, and Christophe Andrieu. On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10:197–208, 2000.

Weinan E. A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 3 2017.

Stefanos Eleftheriadis, Thomas F. W. Nicholson, Marc P. Deisenroth, and James Hensman. Identification of Gaussian Process State Space Models. In *Advances in Neural Information Processing Systems*, 2017.

Mevlana C. Gemici, Danilo J. Rezende, and Shakir Mohamed. Normalizing Flows on Riemannian Manifolds. *arXiv:1611.02304*, 2016.

Alan Genz. Numerical Computation of Rectangular Bivariate and Trivariate Normal and t Probabilities. *Statistics and Computing*, 14: 251–260, 2004.

Soumya Ghosh, Francesco M. Delle Fave, and Jonathan Yedidia. Assumed Density Filtering Methods for Learning Bayesian Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

Agathe Girard, Carl E. Rasmussen, Joaquin Quiñonero Candela, and Roderick Murray-Smith. Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In *Advances in Neural Information Processing Systems*, 2003.

I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, 7th edition, 2007.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

Philipp Hennig, Michael A. Osborne, and Mark Girolami. Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471:20150142, 2015.

José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *Proceedings of the International Conference Machine Learning*, 2015.

Simon J. Julier and Jeffrey K. Uhlmann. A New Extension of the Kalman Filter to Nonlinear Systems. In *Proceedings of AeroSense: Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.

Simon J. Julier and Jeffrey K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, 92(3):401–422, 2004.

Simon J. Julier, Jeffrey K. Uhlmann, and Hugh F. Durrant-Whyte. A New Method for the Nonlinear Transformation of Means and Covariances in Filters and Estimators. In *Proceedings of the American Control Conference*, 1995.

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1×1 Convolutions. In *Advances in Neural Information Processing Systems*, 2018.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

Durk P. Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems*, 2016.

Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian Filtering using Gaussian Process Prediction and Observation Models. *Autonomous Robots*, 27(1):75–90, 2009.

Danie G. Krige. A Statistical Approach to Some Mine Valuations and Allied Problems at the Witwatersrand. Master's thesis, University of the Witwatersrand, 1951.

Aleksandr S. Kronrod. *Nodes and Weights of Quadrature Formulas: Sixteen-place Tables*. Consultants Bureau, 1965.

Dennis V. Lindley. The use of prior probability distributions in statistical inference and decisions. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, 1961.

Dennis V. Lindley and Adrian F. M. Smith. Bayes Estimates for the Linear Model. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(1):1–18, 1972.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph Normalizing Flows. In *Advances in Neural Information Processing Systems*, 2019.

Emile Mathieu and Maximilian Nickel. Riemannian Continuous Normalizing Flows. In *Advances in Neural Information Processing Systems*, 2020.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian Process Library using TensorFlow. *Journal of Machine Learning Research*, 18 (40):1–6, 2017.

Thomas P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *Journal of Machine Learning Research*, 21:1–62, 2020.

Iain Murray. Monte Carlo Inference Methods. *NeurIPS Tutorial*, 2015.

R. F. Ohab and A. R. Stubberud. A Technique for Estimating the State of a Nonlinear System. *IEEE Transactions on Automatic Control*, 10: 150–155, 1965.

Anthony O'Hagan. Curve Fitting and Optimal Design for Prediction. *Journal of the Royal Statistical Society, Series B*, 40(1):1–42, 1978.

Anthony O'Hagan. Bayes-Hermite Quadrature. *Journal of Statistical Planning and Inference*, 29:245–260, 1991.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *Proceedings of the International Conference on Machine Learning*, 2018.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*, 2017.

George Papamakarios, Eric Nalisnick, Danilo J. Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762*, 2019.

Joaquin Quiñonero-Candela, Agathe Girard, Jan Larsen, and Carl E. Rasmussen. Propagation of Uncertainty in Bayesian Kernel Models—Application to Multiple-Step Ahead Forecasting. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2003.

Ali Rahimi and Ben Recht. Random features for large scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008.

Carl E. Rasmussen and Zoubin Ghahramani. Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems*, 2003.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Danilo J. Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the International Conference on Machine Learning*, 2015.

Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Variational Inference in Deep Latent Gaussian Models. In *Proceedings of the International Conference on Machine Learning*, 2014.

Danilo J. Rezende, George Papamakarios, Sébastien Racanière, Michael S. Albergo, Gurtej Kanwar, Phiala E. Shanahan, and Kyle Cranmer. Normalizing Flows on Tori and Spheres. In *Proceedings of the International Conference on Machine Learning*, 2020.

Håvard Rue, Sara Martino, and Nicolas Chopin. Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(2):319–392, 2009.

Hugh Salimbeni and Marc P. Deisenroth. Doubly Stochastic Variational Inference for Deep Gaussian Processes. In *Advances in Neural Information Processing Systems*, 2017.

Hugh Salimbeni, Vincent Dutordoir, James Hensman, and Marc P. Deisenroth. Deep Gaussian Processes with Importance-Weighted Variational Inference. In *Proceedings of the International Conference on Machine Learning*, 2019.

John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic Programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016.

Gerald L. Smith, Stanley F. Schmidt, and Leonard A. McGee. Application of Statistical Filter Theory to the Optimal Estimation of

Position and Velocity on Board a Circumlunar Vehicle. Technical report, NASA, 1962.

Josef Stoer and Roland Bulirsch. *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer-Verlag, 3rd edition, 2002.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

Lilian Weng. Flow-based Deep Generative Models. *lilianweng.github.io/lil-log*, 2018. URL http://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html.

James T. Wilson. *Decision-Making with Gaussian Processes: Sampling Strategies and Monte Carlo Methods*. PhD thesis, Imperial College London, 2023.

James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc P. Deisenroth. Efficiently Sampling Functions from Gaussian Process Posteriors. In *Proceedings of the International Conference on Machine Learning*, 2020a.

James T. Wilson, Viacheslav Borovitskiy, Alexander Terenin, Peter Mostowsky, and Marc P. Deisenroth. Pathwise Conditioning of Gaussian Processes. *Journal of Machine Learning Research*, 22:1–4, 2020b.