Integration

Cheng Soon Ong

Data61, CSIRO

chengsoon.ong@anu.edu.au

@ChengSoonOng

Marc Peter Deisenroth

University College London

m.deisenroth@ucl.ac.uk

@mpd37

December 2020

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

- Evidence (marginal likelihood)

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

- Evidence (marginal likelihood)

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

- Relative entropy (KL divergence)

$$\mathsf{KL}(p||q) = \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} p(\boldsymbol{x}) d\boldsymbol{x}$$

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

- Evidence (marginal likelihood)

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

- Relative entropy (KL divergence)

$$\mathsf{KL}(p||q) = \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} p(\boldsymbol{x}) d\boldsymbol{x}$$

- Prediction in time-series models

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

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

- Evidence (marginal likelihood)

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

- Relative entropy (KL divergence)

$$\mathsf{KL}(p||q) = \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} p(\boldsymbol{x}) d\boldsymbol{x}$$

- Prediction in time-series models

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

- Experimental design

$$\mathbb{E}_{\boldsymbol{\theta}}[U(\boldsymbol{x})] = \int U(\boldsymbol{x}; \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

Integration problems

- Moment computation

$$M_k(x) = \int x^k p(x) dx$$

- Evidence (marginal likelihood)

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

- Relative entropy (KL divergence)

$$\mathsf{KL}(p||q) = \int \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} p(\boldsymbol{x}) d\boldsymbol{x}$$

- Prediction in time-series models

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

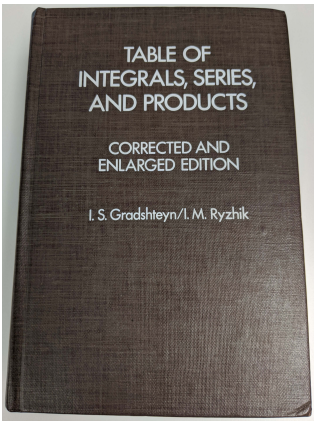
- Experimental design

$$\mathbb{E}_{\boldsymbol{\theta}}[U(\boldsymbol{x})] = \int U(\boldsymbol{x}; \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

- Planning

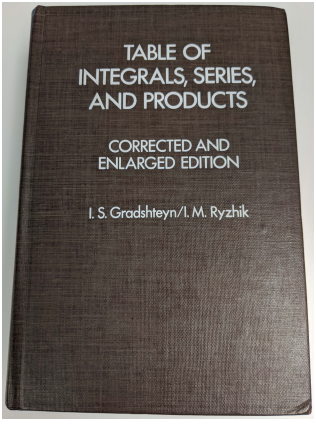
$$J^{\pi}(\boldsymbol{x}(0)) = \int_0^T r(\boldsymbol{x}(t), \boldsymbol{u}(t)) | \boldsymbol{x}(0) dt$$

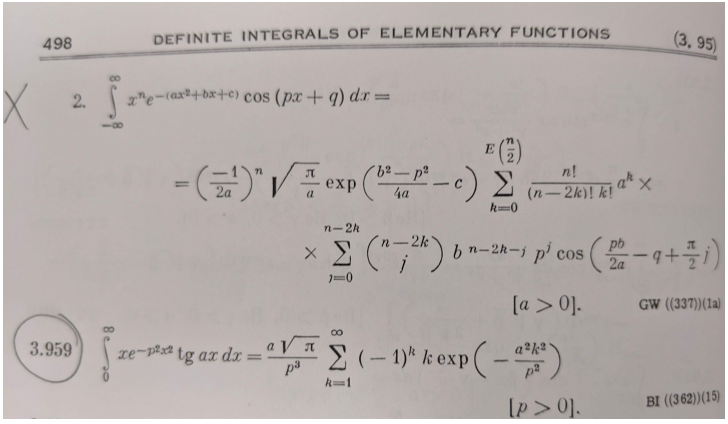
Exact integration



- Compute integrals analytically, if possible (Gradshteyn & Ryzhik, 2007)

Exact integration





- Compute integrals analytically, if possible (Gradshteyn & Ryzhik, 2007)

Approximate integration

Topic	Useful reference	Video
Numerical integration	Stoer & Bulirsch (2002)	Chapter 1
Bayesian quadrature	Rasmussen & Ghahramani (2003), Gunter et al. (2014)	Chapter 1
Monte-Carlo integration	MacKay (2003), Murray (2015)	Chapter 2
Normalizing flows	Weng (2018), Papamakarios et al. (2019), Kobyzev et al. (2020)	Chapter 3
Inference in time series	Julier & Uhlmann (2004), Särkkä (2013)	Chapter 4

References

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

Gunter, T., Osborne, M. A., Garnett, R., Hennig, P., and Roberts, S. J. (2014). Sampling for Inference in Probabilistic Models with Fast Bayesian Quadrature. In *Advances in Neural Information Processing Systems*.

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

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Murray, I. (2015). Monte Carlo Inference Methods. *NeurIPS Tutorial*.

References (cont.)

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762*.

Rasmussen, C. E. and Ghahramani, Z. (2003). Bayesian Monte Carlo. In *Advances in Neural Information Processing Systems*.

Särkkä, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A Look at Gaussian Process Regression Through Kalman Filtering,. *IEEE Signal Processing Magazine*, 30(4):51–61.

Stoer, J. and Bulirsch, R. (2002). *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer-Verlag, 3rd edition.

Weng, L. (2018). Flow-based Deep Generative Models. `lilianweng.github.io/lil-log`.

Monte-Carlo Estimation

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Setting: Computing expectations

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

Setting: Computing expectations

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

Moments of random variables $M_k(x) = \int x^k p(x)dx$

Setting: Computing expectations

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

Moments of random variables $\quad M_k(x) = \int x^k p(x)dx = \mathbb{E}_{x\sim p(x)}[x^k]$

Setting: Computing expectations

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

Moments of random variables

$$M_k(x) = \int x^k p(x)dx = \mathbb{E}_{x\sim p(x)}[x^k]$$

Marginal likelihood

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$

Setting: Computing expectations

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

Moments of random variables

$$M_k(x) = \int x^k p(x)dx = \mathbb{E}_{x\sim p(x)}[x^k]$$

Marginal likelihood
"Average likelihood"

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta})}[p(\boldsymbol{X}|\boldsymbol{\theta})]$$

Setting: Computing expectations

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

Moments of random variables

$$M_k(x) = \int x^k p(x)dx = \mathbb{E}_{x\sim p(x)}[x^k]$$

Marginal likelihood
"Average likelihood"

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta})}[p(\boldsymbol{X}|\boldsymbol{\theta})]$$

Predictions in a Bayesian model

$$p(\boldsymbol{x}_*|\boldsymbol{X}) = \int p(\boldsymbol{x}_*|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X})d\boldsymbol{\theta}$$

Setting: Computing expectations

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

Moments of random variables

$$M_k(x) = \int x^k p(x)dx = \mathbb{E}_{x\sim p(x)}[x^k]$$

Marginal likelihood
"Average likelihood"

$$p(\boldsymbol{X}) = \int p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta})}[p(\boldsymbol{X}|\boldsymbol{\theta})]$$

Predictions in a Bayesian model

$$p(\boldsymbol{x}_*|\boldsymbol{X}) = \int p(\boldsymbol{x}_*|\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{X})d\boldsymbol{\theta}$$

"Average predictive distribution"

$$= \mathbb{E}_{\boldsymbol{\theta}\sim p(\boldsymbol{\theta}|\boldsymbol{X})}[p(\boldsymbol{x}_*|\boldsymbol{\theta})]$$

Key idea

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

Key idea

Make use of random numbers to approximate the expectation.

How it works

Key idea

Make use of random numbers to approximate an expectation.

- Compute expectations via statistical sampling:

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

How it works

Key idea

Make use of random numbers to approximate an expectation.

- Compute expectations via statistical sampling:

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

- Example: Making predictions in a supervised setting (e.g., Bayesian logistic regression with training set $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ at test input $\boldsymbol{x}_*$)

$$p(y_*|\boldsymbol{x}_*, \mathcal{D}) = \int p(y_*|\boldsymbol{\theta}, \boldsymbol{x}_*) \underbrace{p(\boldsymbol{\theta}|\mathcal{D})}_{\substack{\text{parameter} \\ \text{posterior}}} d\boldsymbol{\theta}$$

How it works

Key idea

Make use of random numbers to approximate an expectation.

- Compute expectations via statistical sampling:

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

- Example: Making predictions in a supervised setting (e.g., Bayesian logistic regression with training set $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ at test input $\boldsymbol{x}_*$)

$$p(y_*|\boldsymbol{x}_*, \mathcal{D}) = \int p(y_*|\boldsymbol{\theta}, \boldsymbol{x}_*) \underbrace{p(\boldsymbol{\theta}|\mathcal{D})}_{\substack{\text{parameter} \\ \text{posterior}}} d\boldsymbol{\theta} \approx \frac{1}{S} \sum_{s=1}^{S} p(y_*|\boldsymbol{\theta}^{(s)}, \boldsymbol{x}_*), \quad \boldsymbol{\theta}^{(s)} \sim p(\boldsymbol{\theta}|\mathcal{D})$$

Properties of Monte Carlo estimation

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

- Estimator is unbiased and asymptotically consistent, i.e.,

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

- Error ϵ is normal (Gaussian) and its variance shrinks $\propto 1/S$, independent of the dimensionality

Monte Carlo estimation

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

- How do we get these samples?

Monte Carlo estimation

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

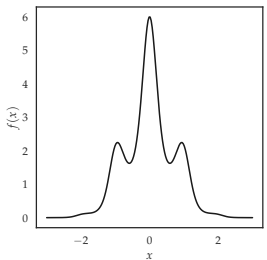
- How do we get these samples?
- Sampling from simple distributions
  - ⏩ Use libraries if the distribution has a "name"

Monte Carlo estimation

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

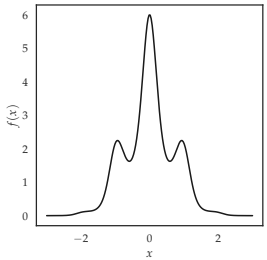
- How do we get these samples?
- Sampling from simple distributions
  ⏩ Use libraries if the distribution has a "name"
- Sampling from complicated distributions
  - Rejection sampling (does not scale to high dimensions)
  - Importance sampling (does not scale to high dimensions)
  - Markov chain Monte Carlo (MCMC) ⏩ Iain Murray's NeurIPS-2015 tutorial

Example



$$Z = \mathbb{E}_x[f(x)] = \int f(x)p(x)dx = \int_{-3}^{3} 6\exp\Big(-x^2 - \sin(3x)^2\Big)\mathcal{U}[-3,3]dx$$

Example

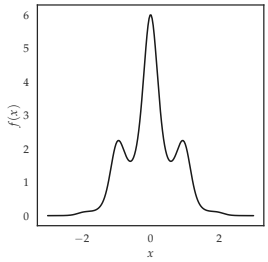


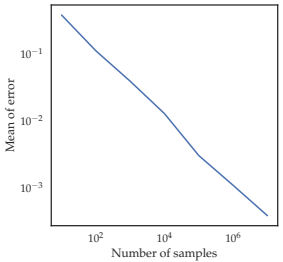
$$Z = \mathbb{E}_x[f(x)] = \int f(x)p(x)dx = \int_{-3}^{3} 6\,\mathsf{exp}\Big(-x^2 - \mathsf{sin}(3x)^2\Big)\mathcal{U}[-3,3]dx$$

- Monte-Carlo estimator

$$\mathbb{E}_{x\sim\mathcal{U}}[f(x)] \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim \mathcal{U}[-3,3]$$

Example



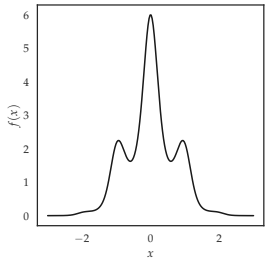


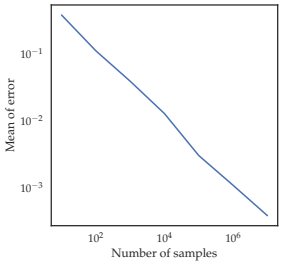
$$Z = \mathbb{E}_x[f(x)] = \int f(x)p(x)dx = \int_{-3}^{3} 6\,\mathsf{exp}\Big(-x^2 - \mathsf{sin}(3x)^2\Big)\mathcal{U}[-3,3]dx$$

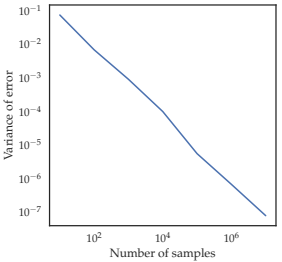
- Monte-Carlo estimator

$$\mathbb{E}_{x\sim\mathcal{U}}[f(x)] \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim \mathcal{U}[-3,3]$$

Example







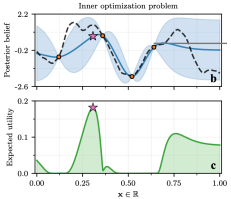
$$Z = \mathbb{E}_x[f(x)] = \int f(x)p(x)dx = \int_{-3}^{3} 6\,\mathsf{exp}\Big(-x^2 - \mathsf{sin}(3x)^2\Big)\mathcal{U}[-3,3]dx$$

- Monte-Carlo estimator

$$\mathbb{E}_{x\sim\mathcal{U}}[f(x)] \approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim \mathcal{U}[-3,3]$$

Some application areas

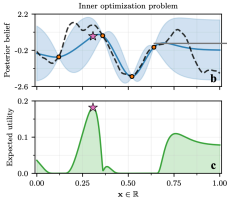
- Empirical risk minimization (Vapnik, 1991)
- Reinforcement learning (e.g., Sutton & Barto, 1998)
- Bayesian optimization
  (e.g., Snoek et al., 2012; Wilson et al., 2018)
- Variational deep learning
  (e.g., Rezende et al., 2014; Kingma & Welling, 2014)
- Probabilistic programming
  ⏩ Frank Wood's NeurIPS-2015 tutorial



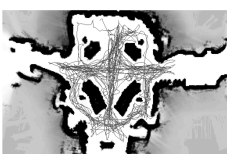
From Wilson et al. (2018)

Some application areas

- Empirical risk minimization (Vapnik, 1991)
- Reinforcement learning (e.g., Sutton & Barto, 1998)
- Bayesian optimization
  (e.g., Snoek et al., 2012; Wilson et al., 2018)
- Variational deep learning
  (e.g., Rezende et al., 2014; Kingma & Welling, 2014)
- Probabilistic programming
  ⏩ Frank Wood's NeurIPS-2015 tutorial
- High-energy physics (e.g., Buckley et al., 2011)
- Robotics (e.g., Dellaert et al., 1999)



From Wilson et al. (2018)



From Dellaert et al. (1999)

Considerations

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

- Require many samples to get a good estimate of the value of the integral
- Design efficient samplers (computationally efficient, low variance)
- Function needs to be cheap to evaluate
- Good for learning, if we are just interested in an unbiased estimator
- Estimator does not take the locations of the samples into account
  ⏩ Could be problematic in small-sample regimes (O'Hagan, 1987)

Summary: Monte Carlo estimation

- Random numbers to compute expectations
- Estimator has nice properties
  (e.g., unbiased, asymptotically consistent)
- Scales to high dimensions
- General approach and straightforward
- Widely applicable
- Generating samples is the key challenge (not covered here)

References

Buckley, A., Butterworth, J., Gieseke, S., Grellscheid, D., Höche, S., Hoeth, H., Krauss, F., Lönnblad, L., Nurse, E., Richardson, P., et al. (2011). General-Purpose Event Generators for LHC Physics. *Physics Reports*, 504(5):145–233.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. In *Proceedings of International Conference on Robotics and Automation*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *Proceedings of the International Conference on Learning Representations*.

Murray, I. (2015). Monte Carlo Inference Methods. *NeurIPS Tutorial*.

O'Hagan, A. (1987). Monte Carlo is Fundamentally Unsound. *The Statistician*, 36(2/3):247–249.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Variational Inference in Deep Latent Gaussian Models. In *Proceedings of the International Conference on Machine Learning*.

References (cont.)

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Vapnik, V. (1991). Principles of Risk Minimization for Learning Theory . In *Advances in Neural Information Processing Systems*.

Wilson, J. T., Hutter, F., and Deisenroth, M. P. (2018). Maximizing Acquisition Functions for Bayesian Optimization. In *Advances in Neural Information Processing Systems*.

Wood, F. (2015). Probabilistic Programming. *NeurIPS Tutorial*.

Change of Variables and Normalizing Flows

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Normalizing flows for density estimation

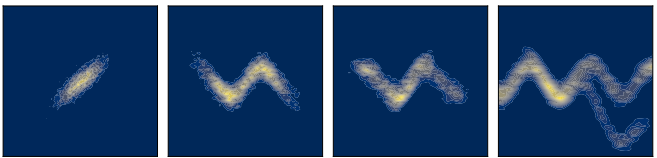


Figure: Generated with PyMC3 (Salvatier et al., 2016)

Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

Build complex distributions from simple distributions via a flow of successive (invertible) transformations

Normalizing flows for density estimation

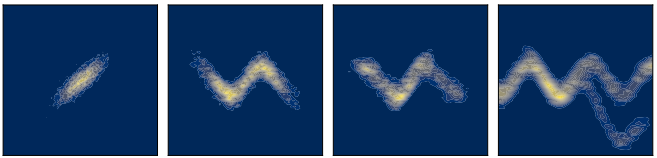


Figure: Generated with PyMC3 (Salvatier et al., 2016)

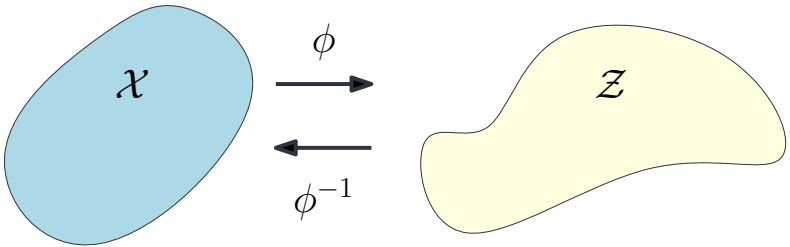
Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

Build complex distributions from simple distributions via a flow of successive (invertible) transformations

Key ingredient: **Change-of-variables trick**

Change of Variables

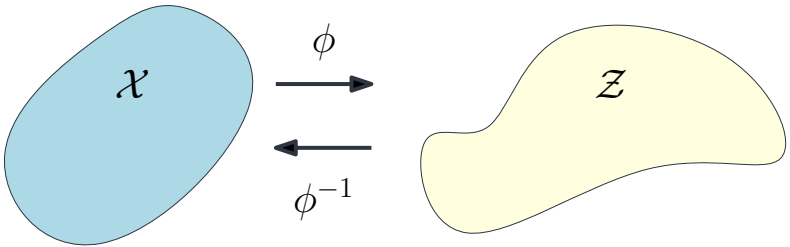
Change of variables: Key idea



Key idea

Transform random variable X into random variable Z using an invertible transformation ϕ, while keeping track of the change in distribution

Change of variables: Key idea

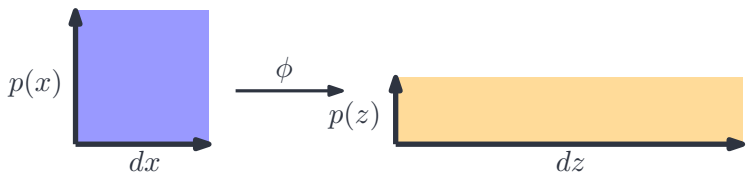


Key idea

Transform random variable X into random variable Z using an invertible transformation ϕ, while keeping track of the change in distribution

- Distribution p_X induces distribution p_Z via transformation ϕ
- Distribution p_Z induces distribution p_X via transformation ϕ^{-1}

Jacobian determinant



- Determinant of Jacobian

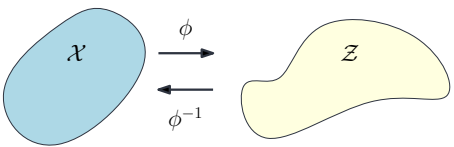
$$\left|\det\left(\frac{dz}{dx}\right)\right| = \left|\det\left(\frac{d\phi(x)}{dx}\right)\right|$$

tells us how much the domain dx is stretched to dz

How it works

- Constraint: volume preservation

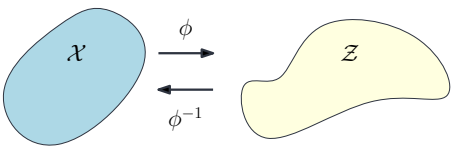
$$\int_{\mathcal{X}} p_X(\boldsymbol{x}) d\boldsymbol{x} = 1 = \int_{\mathcal{Z}} p_Z(\boldsymbol{z}) d\boldsymbol{z}$$



How it works

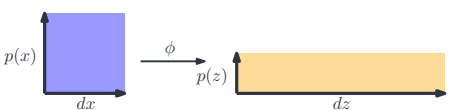
- Constraint: volume preservation

$$\int_{\mathcal{X}} p_X(\boldsymbol{x}) d\boldsymbol{x} = 1 = \int_{\mathcal{Z}} p_Z(\boldsymbol{z}) d\boldsymbol{z}$$

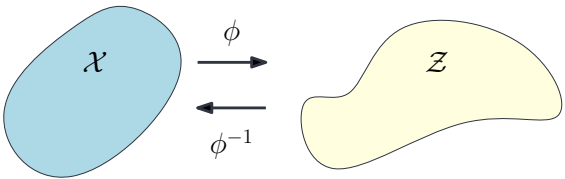


- Volume preservation: Rescale p_Z by the inverse of Jacobian determinant

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



Considerations



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

- Express target distribution p_Z in terms of known distribution p_X and the Jacobian determinant of an invertible mapping ϕ
- No need to invert ϕ explicitly
- Generate expressive distributions p_Z by simple p_X and flexible transformation ϕ

Applications

- Numerical integration (turn indefinite integrals into definite ones)
- Neural ODEs (E 2017, Chen et al., 2018)
- Learning in implicit generative models (e.g., GANs) and likelihood-free inference (e.g., ABC)
  (e.g., Mohamed & Lakshminarayanan, 2016; Sisson et al., 2007)
- **Normalizing flows** (Rezende & Mohamed, 2015)

Normalizing Flows

Normalizing flows for density estimation

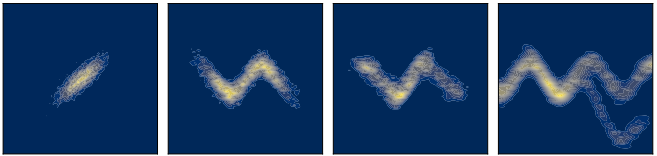
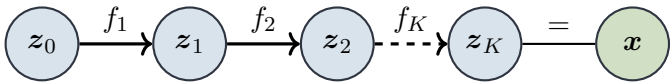


Figure: Generated with PyMC3 (Salvatier et al., 2016)

Key idea
(Tabak & Turner, 2013; Rippel & Adams, 2013; Rezende & Mohamed, 2015)

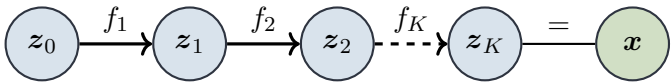
Build complex distributions from simple distributions via a flow of successive (invertible) transformations

How it works



- Random variable $z_0 \sim p_0$
- Simple base distribution p_0, e.g. $p_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

How it works



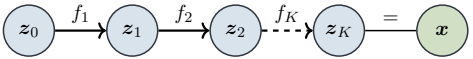
- Random variable $z_0 \sim p_0$
- Simple base distribution p_0, e.g. $p_0 = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
- Successive transformation of z_k via invertible transformations f_k:

$$z_k = f_k(z_{k-1})$$

- Observed data $x = z_K$ at the end of the chain

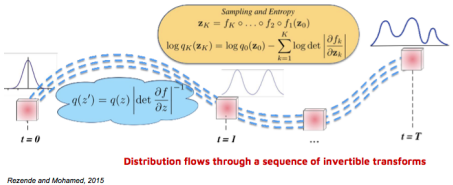
$$x = z_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(z_0)$$

Marginal distribution



- Repeated application of **change-of-variables** trick

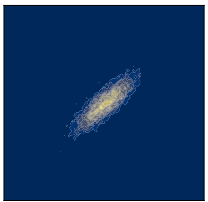
$$p(\boldsymbol{x}) = p(\boldsymbol{z}_K) = p(\boldsymbol{z}_0) \prod_{k=1}^{K} \left| \det \frac{df_k(\boldsymbol{z}_{k-1})}{d\boldsymbol{z}_{k-1}} \right|^{-1}$$



- Entropy

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

Illustration with PyMC3 (Salvatier et al., 2016)

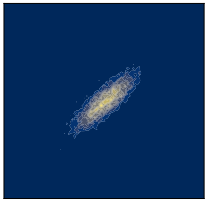


1 planar flow

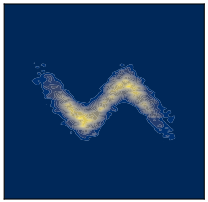
Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)



1 planar flow

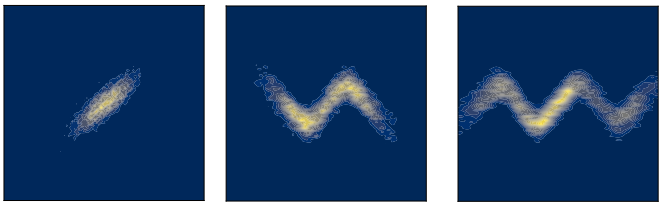


2 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)

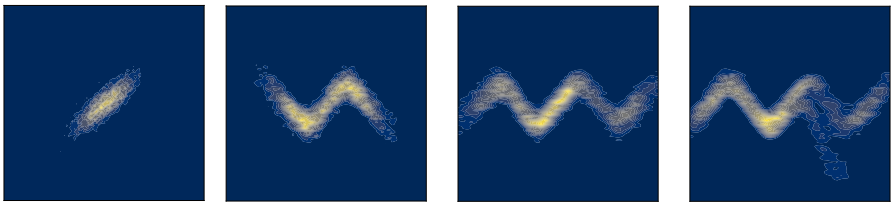


1 planar flow 2 planar flows 3 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)

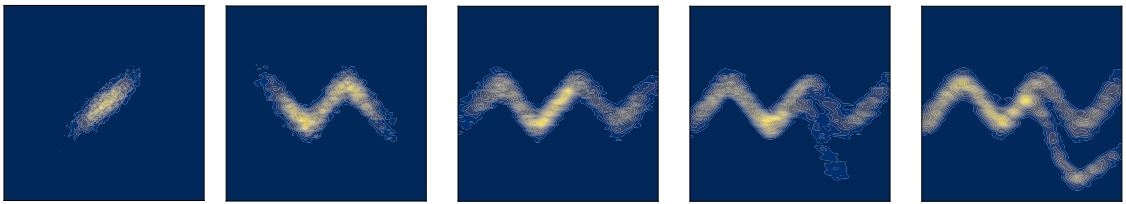


1 planar flow | 2 planar flows | 3 planar flows | 7 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Illustration with PyMC3 (Salvatier et al., 2016)



1 planar flow | 2 planar flows | 3 planar flows | 7 planar flows | 12 planar flows

Figure: Generated using a PyMC3 tutorial (Salvatier et al., 2016)

- Repeated application of a planar flow $\boldsymbol{z}_k = f_k(\boldsymbol{z}_{k-1}) = \boldsymbol{z}_{k-1} + \boldsymbol{u}\sigma(\boldsymbol{w}^\top \boldsymbol{z}_{k-1} + b)$

Computing expectations

$$\mathbb{E}_{p_X}[l(\boldsymbol{x})] = \mathbb{E}_{p_K}[l(\boldsymbol{z}_K)] = \mathbb{E}_{p_0}[l(f_K \circ \cdots \circ f_1(\boldsymbol{z}_0))]$$

Computing expectations

$$\mathbb{E}_{p_X}[l(\boldsymbol{x})] = \mathbb{E}_{p_K}[l(\boldsymbol{z}_K)] = \mathbb{E}_{p_0}[l(f_K \circ \cdots \circ f_1(\boldsymbol{z}_0))]$$

- Expectations w.r.t. p_K can be computed without explicitly knowing p_K or p_X
  - Sample $\boldsymbol{z}_0^{(s)} \sim p_0$
  - Push sample forward through sequence of deterministic transformations
    ⏩ Valid sample $\boldsymbol{x}^{(s)} \sim p_X(\boldsymbol{x})$
- Monte-Carlo estimation to get expected value

Computational considerations

- Compute log-determinant of Jacobian
- Cheap (linear) if Jacobian is (block-)diagonal or triangular

Computational considerations

- Compute log-determinant of Jacobian
- Cheap (linear) if Jacobian is (block-)diagonal or triangular
- Require partial derivatives

$$\frac{\partial z_k^{(d)}}{\partial z_{k-1}^{(>d)}} = 0 \quad \blacktriangleright\!\blacktriangleright\!\blacktriangleright \quad \frac{d\boldsymbol{z}_k}{d\boldsymbol{z}_{k-1}} = \begin{bmatrix} \frac{\partial z_k^{(1)}}{\partial z_{k-1}^{(1)}} & \mathbf{0} & \dots & \mathbf{0} \\ \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(1)}} & \frac{\partial z_k^{(2)}}{\partial z_{k-1}^{(2)}} & \dots & \vdots \\ \vdots & & \ddots & \mathbf{0} \\ \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(1)}} & \dots & \dots & \frac{\partial z_k^{(D)}}{\partial z_{k-1}^{(D)}} \end{bmatrix} \in \mathbb{R}^{D \times D}.$$

Autoregressive flows

- High-level idea:

$$z_k^{(d)} = \phi(z_{k-1}^{(\leq d)})$$

Autoregressive flows

- High-level idea:

$$z_k^{(d)} = \phi(z_{k-1}^{(\leq d)})$$

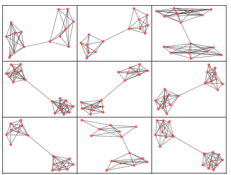
- NICE (Dinh et al., 2014)
- Inverse autoregressive flow (Kingma et al., 2016)
- Real NVP (Dinh et al., 2017)
- Masked autoregressive flow (Papamakarios et al., 2017)
- Glow (Kingma & Dhariwal, 2018)
- (Block) neural autoregressive flows, spline flows, ... (e.g., Huang et al., 2018; de Cao et al., 2019; Durkan et al., 2019)

Application areas

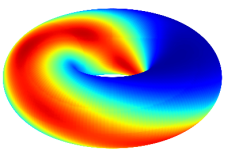
- Variational inference in deep generative models (e.g., Rezende & Mohamed, 2015)
- Graph neural networks (Liu et al., 2019)
- Parallel WaveNet (van den Oord et al., 2018)

Application areas

- Variational inference in deep generative models (e.g., Rezende & Mohamed, 2015)
- Graph neural networks (Liu et al., 2019)
- Parallel WaveNet (van den Oord et al., 2018)
- Continuous flows
  - Neural ODEs (e.g, E, 2017; Chen et al., 2018)
  - Flows on manifolds (e.g., Gemici et al., 2016; Rezende et al., 2020; Mathieu & Nickel, 2020)

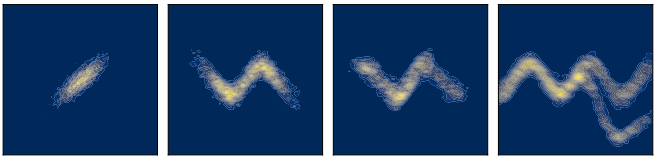


From Liu et al. (2019)



From Rezende et al. (2020)

Summary



- Normalizing flows provide a constructive way to generate rich distributions
- Key idea: Transform a simple distribution using a flow of successive (invertible) transformations
- Key ingredient: Change-of-variables trick
- Jacobians can be computed efficiently, if the transformations are defined appropriately
- Can be used as a generator and inference mechanism

References

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*.

De Cao, N., Aziz, W., and Titov, I. (2019). Block Neural Autoregressive Flow. In *Proceedings of the Conferene on Uncertainty in Artificial Intelligence*.

Dinh, L., Krueger, D., and Bengio, Y. (2014). NICE: Non-linear Independent Components Estimation. *arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density Estimation Using Real NVP. In *Proceedings of the International Conference on Learning Representation*.

Durkan, C., Bekasov, A., Murray, I., and Papamakarios, G. (2019). Neural Spline Flows. In *Advances in Neural Information Processing Systems*.

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

References (cont.)

Gemici, M. C., Rezende, D. J., and Mohamed, S. (2016). Normalizing Flows on Riemannian Manifolds. *arXiv:1611.02304*.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. (2018). Neural Autoregressive Flows. In *Proceedings of the International Conference on Machine Learning*.

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

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Procesing Systems*.

Kobyzev, I., Prince, S., and Brubaker, M. (2020). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. (2019). Graph Normalizing Flows. In *Advances in Neural Information Processing Systems*.

References (cont.)

Mathieu, E. and Nickel, M. (2020). Riemannian Continuous Normalizing Flows. *arXiv:2006.10605*.

Mohamed, S. and Lakshminarayanan, B. (2016). Learning in Implicit Generative Models. *arXiv:1610.03483*.

Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. (2018). Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *Proceedings of the International Conference on Machine Learning*.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2019). Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762*.

Papamakarios, G., Pavlakou, T., and Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*.

Rezende, D. J. and Mohamed, S. (2015). Variational Inference with Normalizing Flows. In *Proceedings of the International Conference on Machine Learning*.

References (cont.)

Rezende, D. J., Papamakarios, G., Racanière, S., Albergo, M. S., Kanwar, G., Shanahan, P. E., and Cranmer, K. (2020). Normalizing Flows on Tori and Spheres. In *Proceedings of the International Conference on Machine Learning*.

Rippel, O. and Adams, R. P. (2013). High-Dimensional Probability Estimation with Deep Density Models. *arXiv:1302.5125*.

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic Programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55.

Sisson, S. A., Fan, Y., and Tanaka, M. M. (2007). Sequential Monte Carlo Without Likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 104(6):1760–1765.

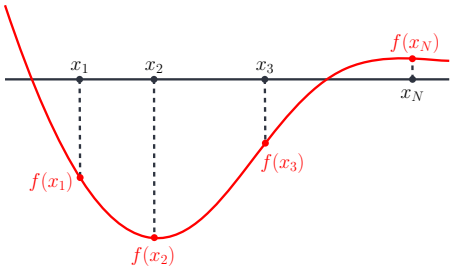
Tabak, E. G. and Turner, C. V. (2013). A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164.

Numerical Integration

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Setting

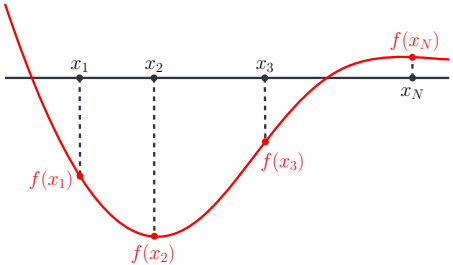


- Approximate

$$\int_a^b f(x)dx \approx \sum_{n=1}^{N} w_n f(x_n), \quad x \in \mathbb{R}$$

- Nodes x_n and corresponding function values $f(x_n)$

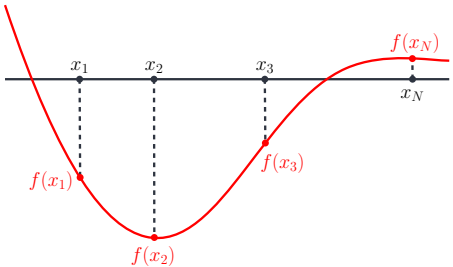
Numerical integration (quadrature)



Key idea

Approximate f using an interpolating function that is easy to integrate (e.g., polynomial)

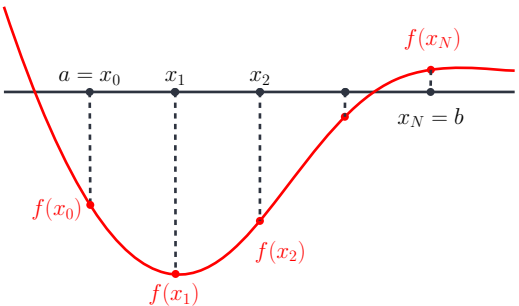
Quadrature approaches



Quadrature	Interpolant	Nodes
Newton–Cotes	low-degree polynomials	equidistant
Gaussian	orthogonal polynomials	roots of polynomial
Bayesian	Gaussian process	user defined

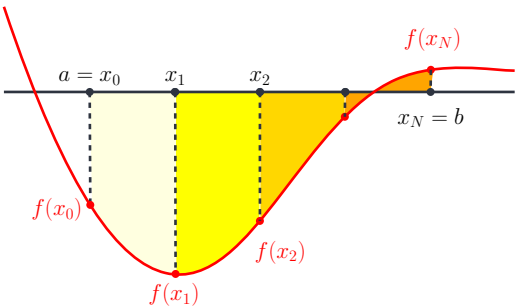
Newton–Cotes Quadrature

Overview



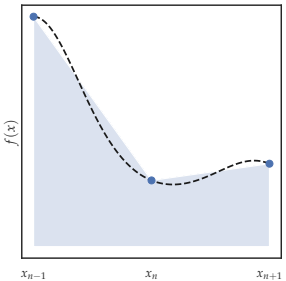
- Equidistant nodes $a = x_0, \ldots, x_N = b$ ⏩ Partition interval $[a, b]$
- Approximate f in each partition with a low-degree polynomial

Overview



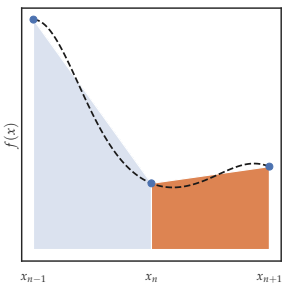
- Equidistant nodes $a = x_0, \ldots, x_N = b$ ⏩ Partition interval $[a, b]$
- Approximate f in each partition with a low-degree polynomial
- Compute integral for each partition analytically and sum them up

Trapezoidal rule



- Partition $[a, b]$ into N segments with equidistant nodes x_n
- **Locally linear approximation** of f between nodes

Trapezoidal rule (2)

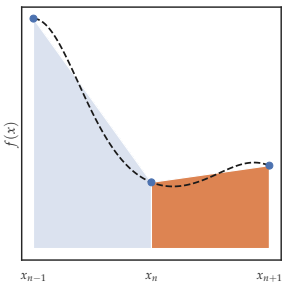


- Area of a trapezoid with corners $(x_n, x_{n+1}, f(x_{n+1}), f(x_n))$

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

$$h := |x_{n+1} - x_n| \quad \text{⏩Distance between nodes}$$

Trapezoidal rule (2)



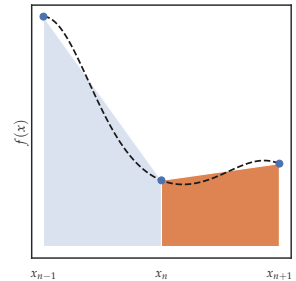
- Area of a trapezoid with corners $(x_n, x_{n+1}, f(x_{n+1}), f(x_n))$

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

$$h := |x_{n+1} - x_n| \quad \text{⏩Distance between nodes}$$

- Error $\mathcal{O}(h^2)$

Trapezoidal rule (2)



- Area of a trapezoid with corners $(x_n, x_{n+1}, f(x_{n+1}), f(x_n))$

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

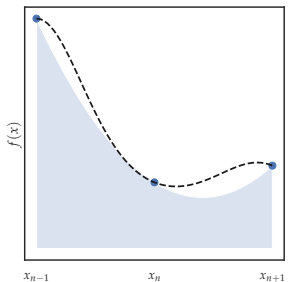
$$h := |x_{n+1} - x_n| \quad \text{⏩Distance between nodes}$$

- Error $\mathcal{O}(h^2)$

- Full integral:

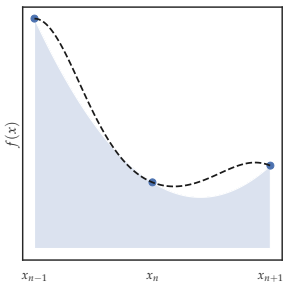
$$\int_a^b f(x)dx \approx \frac{h}{2}\big(f_0 + 2f_1 + \cdots + 2f_{N-1} + f_N\big), \quad f_n := f(x_n)$$

Simpson's rule



- Partition $[a, b]$ into N segments with equidistant nodes x_n
- **Locally quadratic approximation** of f connecting triplets $\big(f(x_{n-1}), f(x_n), f(x_{n+1})\big)$

Simpson's rule (2)

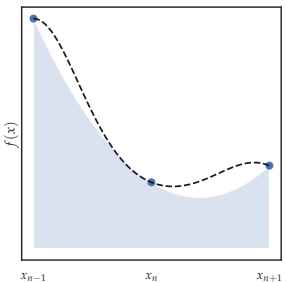


- Area of segment:

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

$h := |x_{n+1} - x_n|$ ⏩ Distance between nodes

Simpson's rule (2)



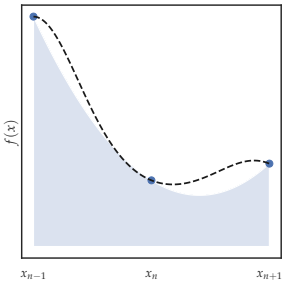
- Area of segment:

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

$h := |x_{n+1} - x_n|$ ⏩ Distance between nodes

- Error: $\mathcal{O}(h^4)$

Simpson's rule (2)



- Area of segment:

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

$$h := |x_{n+1} - x_n| \quad \text{⏩⏩Distance between nodes}$$

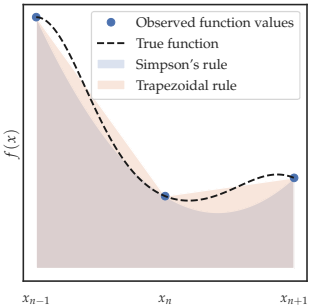
- Error: $\mathcal{O}(h^4)$

- Full integral:

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

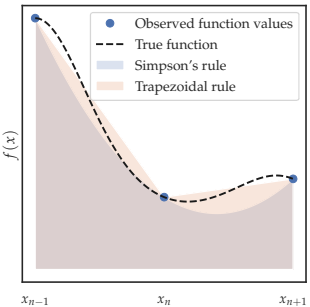
Example

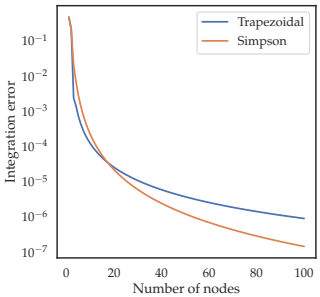
$$\int_0^1 \exp(-x^2 - \sin(3x)^2)dx$$



Example

$$\int_0^1 \exp(-x^2 - \sin(3x)^2)dx$$

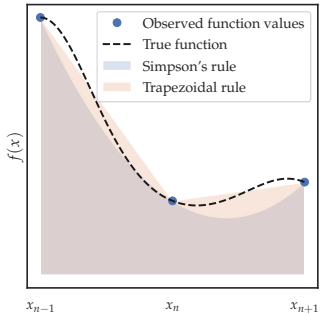




- Simpson's rule yields better approximations
- Very good approximations obtained fairly quickly

Summary: Newton–Cotes quadrature

- Approximate integrand between equidistant nodes with a low-degree polynomial (up to degree 6)
- Trapezoidal rule: linear interpolation
- Simpson's rule: quadratic interpolation
  ⏩ Better approximation and smaller integration error



Gaussian Quadrature

Gaussian quadrature

- Named after Carl Friedrich Gauß

Gaussian quadrature

- Named after Carl Friedrich Gauß
- Quadrature scheme that no longer relies on equidistant nodes ⏩ Higher accuracy

Gaussian quadrature

- Named after Carl Friedrich Gauß
- Quadrature scheme that no longer relies on equidistant nodes ⏩ Higher accuracy
- Central approximation

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

Gaussian quadrature

- Named after Carl Friedrich Gauß
- Quadrature scheme that no longer relies on equidistant nodes ⏩ Higher accuracy
- Central approximation

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

- Weight function $w(x) \geq 0$ (and some other integration-related properties, which are satisfied if $w(x)$ is a pdf)

Gaussian quadrature

- Named after Carl Friedrich Gauß
- Quadrature scheme that no longer relies on equidistant nodes ⏩ Higher accuracy
- Central approximation

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

- Weight function $w(x) \geq 0$ (and some other integration-related properties, which are satisfied if $w(x)$ is a pdf)
- Goal: Find nodes x_n and weights w_n, so that the approximation error is minimized

Central idea

- Quadrature nodes x_n are the roots of a family of **orthogonal polynomials**

Central idea

- Quadrature nodes x_n are the roots of a family of **orthogonal polynomials**
  - ⏩ Nodes no longer equidistant

Central idea

- Quadrature nodes x_n are the roots of a family of **orthogonal polynomials**
  ⏩ Nodes no longer equidistant
- Exact if f is a polynomial of degree $\leq 2N-1$, i.e.,

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

⏩ Integral can be computed exactly by evaluating f N times at the optimal locations x_n (roots of an orthogonal polynomial) with corresponding optimal weights w_n

Central idea

- Quadrature nodes x_n are the roots of a family of **orthogonal polynomials**
  ⏩ Nodes no longer equidistant
- Exact if f is a polynomial of degree $\leq 2N-1$, i.e.,

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

⏩ Integral can be computed exactly by evaluating f N times at the optimal locations x_n (roots of an orthogonal polynomial) with corresponding optimal weights w_n
⏩ More accurate than Newton–Cotes for the same number of evaluations (with some memory overhead)

Example: Gauß–Hermite quadrature

- Solve

$$\int f(x) \underbrace{\exp(-x^2)}_{w(x)} dx = \int f(x) \frac{\sqrt{2\pi}}{\exp(-x^2/2)} \mathcal{N}\left(x \middle| 0, 1\right) dx$$

$$= \sqrt{2\pi} \mathbb{E}_{x \sim \mathcal{N}(0,1)} \left[\frac{f(x)}{\exp(-x^2/2)} \right]$$

Example: Gauß–Hermite quadrature

- Solve

$$\int f(x) \underbrace{\exp(-x^2)}_{w(x)} dx = \int f(x) \frac{\sqrt{2\pi}}{\exp(-x^2/2)} \mathcal{N}\big(x \big| 0, 1\big) dx$$

$$= \sqrt{2\pi} \mathbb{E}_{x \sim \mathcal{N}(0,1)} \left[\frac{f(x)}{\exp(-x^2/2)} \right]$$

- With change-of-variables trick ⏭ Expectation w.r.t. a Gaussian measure

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

Example: Gauß–Hermite quadrature (2)

- Follow general approximation scheme

$$\int f(x) \underbrace{\exp(-x^2)}_{w(x)} dx \approx \sum_{n=1}^{N} w_n f(x_n)$$

Example: Gauß–Hermite quadrature (2)

- Follow general approximation scheme

$$\int f(x) \underbrace{\exp(-x^2)}_{w(x)} dx \approx \sum_{n=1}^{N} w_n f(x_n)$$

- Nodes $x_1, \ldots, x_N$ are the roots of Hermite polynomial

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

Example: Gauß–Hermite quadrature (2)

- Follow general approximation scheme

$$\int f(x) \underbrace{\exp(-x^2)}_{w(x)} dx \approx \sum_{n=1}^{N} w_n f(x_n)$$

- Nodes $x_1, \ldots, x_N$ are the roots of Hermite polynomial

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

- Weights w_n are

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

Overview (Stoer & Bulirsch, 2002)

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

$[a, b]$	$w(x)$	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

Application areas

- Probabilities for rectangular bivariate/trivariate Gaussian and t distributions (Genz, 2004)
- Integrating out (marginalizing) a few hyper-parameters in a latent-variable model (INLA; Rue et al., 2009)
- Predictions with a Gaussian process classifier (GPFlow; Matthews et al., 2017)

Summary: Gaussian quadrature

- Orthogonal polynomials to approximate f
- Nodes are the roots of the polynomial
- Higher accuracy than Newton–Cotes
- **Method of choice** for low-dimensional problems (1–3 dimensions)

Summary: Gaussian quadrature

- Orthogonal polynomials to approximate f
- Nodes are the roots of the polynomial
- Higher accuracy than Newton–Cotes
- **Method of choice** for low-dimensional problems (1–3 dimensions)
- Can't naturally deal with noisy observations
- Only works in low dimensions

Summary: Gaussian quadrature

- Orthogonal polynomials to approximate f
- Nodes are the roots of the polynomial
- Higher accuracy than Newton–Cotes
- **Method of choice** for low-dimensional problems (1–3 dimensions)
- Can't naturally deal with noisy observations
- Only works in low dimensions
- Approaches that scale better with dimensionality
  - ⏩ **Bayesian quadrature** (up to $\approx$ 10 dimensions)
  - ⏩ **Monte Carlo estimation** (high dimensions)

Bayesian Quadrature

Bayesian quadrature: Setting and key idea

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

- Function f is expensive to evaluate
- Integration in moderate (≤ 10) dimensions
- Deal with noisy function observations

Bayesian quadrature: Setting and key idea

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

- Function f is expensive to evaluate
- Integration in moderate (≤ 10) dimensions
- Deal with noisy function observations

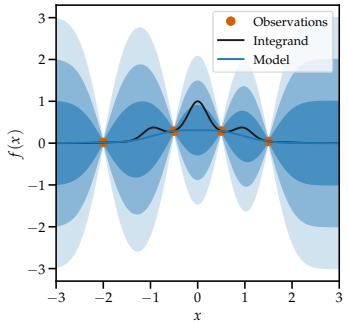
Key idea

- Phrase quadrature as a statistical inference problem
  ⏩ Probabilistic numerics (e.g., Hennig et al., 2015; Briol et al., 2015)
- Estimate distribution on Z using a dataset $\mathcal{D} := \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N))\}$

Bayesian quadrature: How it works

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

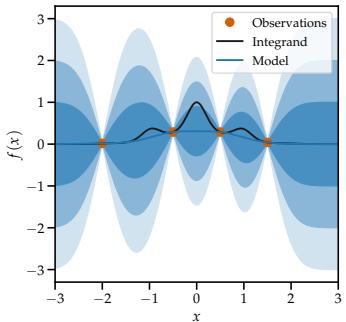
- Estimate distribution on Z using a dataset $\mathcal{D} := \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N))\}$



Bayesian quadrature: How it works

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

- Estimate distribution on Z using a dataset $\mathcal{D} := \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N))\}$
- Place (Gaussian process) prior distribution on f and determine the posterior via Bayes' theorem (Diaconis 1988; O'Hagan 1991; Rasmussen & Ghahramani 2003)
  - ⏩ Distribution on f induces a distribution on Z

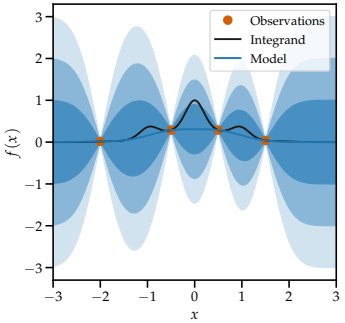


Bayesian quadrature: How it works

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

- Estimate distribution on Z using a dataset $\mathcal{D} := \{(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_N, f(\boldsymbol{x}_N))\}$
- Place (Gaussian process) prior distribution on f and determine the posterior via Bayes' theorem (Diaconis 1988; O'Hagan 1991; Rasmussen & Ghahramani 2003)
  - ⏩ Distribution on f induces a distribution on Z
- Generalizes to noisy function observations

$$y = f(\boldsymbol{x}) + \epsilon$$



Bayesian quadrature: Details

$$Z := \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}), \quad f \sim GP(0, k)$$

Bayesian quadrature: Details

$$Z := \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}), \quad f \sim GP(0, k)$$

- Exploit linearity of the integral (integral of a GP is another GP)

Bayesian quadrature: Details

$$Z := \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}), \quad f \sim GP(0, k)$$

- Exploit linearity of the integral (integral of a GP is another GP)

$$p(Z) = p\left(\int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x})\right) = \mathcal{N}\left(Z \middle| \mu_Z, \sigma_Z^2\right)$$

Bayesian quadrature: Details

$$Z := \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}, \quad f \sim GP(0, k)$$

- Exploit linearity of the integral (integral of a GP is another GP)

$$p(Z) = p\left(\int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}\right) = \mathcal{N}\left(Z \middle| \mu_Z, \sigma_Z^2\right)$$

$$\mu_Z = \int \mu_{\text{post}}(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{x}}[\mu_{\text{post}}(\boldsymbol{x})]$$

Bayesian quadrature: Details

$$Z := \int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}), \quad f \sim GP(0, k)$$

- Exploit linearity of the integral (integral of a GP is another GP)

$$p(Z) = p\left(\int f(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x}\right) = \mathcal{N}\left(Z \middle| \mu_Z, \sigma_Z^2\right)$$

$$\mu_Z = \int \mu_{\text{post}}(\boldsymbol{x}) p(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{x}}[\mu_{\text{post}}(\boldsymbol{x})]$$

$$\sigma_Z^2 = \iint k_{\text{post}}(\boldsymbol{x}, \boldsymbol{x}') p(\boldsymbol{x}) p(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{x}' = \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k_{\text{post}}(\boldsymbol{x}, \boldsymbol{x}')]$$

Bayesian quadrature: Mean

$$\mathbb{E}_f[Z] = \mu_Z = \overbrace{\mathbb{E}_{\boldsymbol{x}\sim p}[\mu_{\text{post}}(\boldsymbol{x})]}^{\text{expected predictive mean}}$$

$$Z = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

$$f \sim GP(0, k)$$

$$p(Z) = \mathcal{N}\left(Z \middle| \mu_Z, \sigma_Z^2\right)$$

Training data: $\boldsymbol{X}, \boldsymbol{y}$

Bayesian quadrature: Mean

$$\mathbb{E}_f[Z] = \mu_Z = \overbrace{\mathbb{E}_{\boldsymbol{x}\sim p}[\mu_{\text{post}}(\boldsymbol{x})]}^{\text{expected predictive mean}}$$

$$\mu_{\text{post}}(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{X}) \underbrace{\boldsymbol{K}^{-1}\boldsymbol{y}}_{=:\boldsymbol{\alpha}}, \quad \boldsymbol{K} := k(\boldsymbol{X}, \boldsymbol{X})$$

$$Z = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

$$f \sim GP(0, k)$$

$$p(Z) = \mathcal{N}\left(Z \middle| \mu_Z, \sigma_Z^2\right)$$

Training data: $\boldsymbol{X}, \boldsymbol{y}$

Bayesian quadrature: Mean

$$\mathbb{E}_f[Z] = \mu_Z = \overbrace{\mathbb{E}_{\boldsymbol{x}\sim p}[\mu_{\text{post}}(\boldsymbol{x})]}^{\text{expected predictive mean}}$$

$$\mu_{\text{post}}(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{X})\underbrace{\boldsymbol{K}^{-1}\boldsymbol{y}}_{=:\boldsymbol{\alpha}}, \quad \boldsymbol{K} := k(\boldsymbol{X}, \boldsymbol{X})$$

$$\mathbb{E}_f[Z] = \overbrace{\int k(\boldsymbol{x}, \boldsymbol{X})p(\boldsymbol{x})d\boldsymbol{x}}^{=:\boldsymbol{z}^\top}\boldsymbol{\alpha} = \boldsymbol{z}^\top\boldsymbol{\alpha}$$

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

$$Z = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

$$f \sim GP(0, k)$$

$$p(Z) = \mathcal{N}\left(Z\middle|\mu_Z, \sigma_Z^2\right)$$

Training data: $\boldsymbol{X}, \boldsymbol{y}$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \iint k(\boldsymbol{x},\boldsymbol{x}')p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')]$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \iint k(\boldsymbol{x},\boldsymbol{x}')p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}' - \underbrace{\int k(\boldsymbol{x},\boldsymbol{X})p(\boldsymbol{x})d\boldsymbol{x}}_{=\boldsymbol{z}^\top}$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')] - \boldsymbol{z}^\top$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \iint k(\boldsymbol{x},\boldsymbol{x}')p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}' - \underbrace{\int k(\boldsymbol{x},\boldsymbol{X})p(\boldsymbol{x})d\boldsymbol{x}}_{=\boldsymbol{z}^\top}\boldsymbol{K}^{-1}$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')] - \boldsymbol{z}^\top\boldsymbol{K}^{-1}$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \iint k(\boldsymbol{x},\boldsymbol{x}')p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}' - \underbrace{\int k(\boldsymbol{x},\boldsymbol{X})p(\boldsymbol{x})d\boldsymbol{x}}_{=\boldsymbol{z}^\top}\boldsymbol{K}^{-1}\underbrace{\int k(\boldsymbol{X},\boldsymbol{x}')p(\boldsymbol{x}')d\boldsymbol{x}'}_{=\boldsymbol{z}'}$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')] - \boldsymbol{z}^\top\boldsymbol{K}^{-1}\boldsymbol{z}'$$

Bayesian quadrature: Variance

$$\mathbb{V}_f[Z] = \sigma_Z^2 = \overbrace{\mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k_{\text{post}}(\boldsymbol{x},\boldsymbol{x}')]}^{\text{expected posterior covariance}}$$

$$= \iint \underbrace{k(\boldsymbol{x},\boldsymbol{x}')}_{\text{prior covariance}} - \underbrace{k(\boldsymbol{x},\boldsymbol{X})\boldsymbol{K}^{-1}k(\boldsymbol{X},\boldsymbol{x}')}_{\text{information from training data}} p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}'$$

$$= \iint k(\boldsymbol{x},\boldsymbol{x}')p(\boldsymbol{x})p(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{x}' - \underbrace{\int k(\boldsymbol{x},\boldsymbol{X})p(\boldsymbol{x})d\boldsymbol{x}}_{=\boldsymbol{z}^\top}\boldsymbol{K}^{-1}\underbrace{\int k(\boldsymbol{X},\boldsymbol{x}')p(\boldsymbol{x}')d\boldsymbol{x}'}_{=\boldsymbol{z}'}$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')] - \boldsymbol{z}^\top\boldsymbol{K}^{-1}\boldsymbol{z}'$$

$$= \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'}[k(\boldsymbol{x},\boldsymbol{x}')] - \mathbb{E}_{\boldsymbol{x}}[k(\boldsymbol{x},\boldsymbol{X})]\boldsymbol{K}^{-1}\mathbb{E}_{\boldsymbol{x}'}[k(\boldsymbol{X},\boldsymbol{x}')]$$

Computing kernel expectations

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

- Solve a different (easier) integration problem

Computing kernel expectations

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

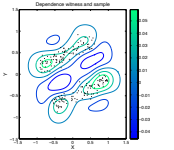
- Solve a different (easier) integration problem

	Input distribution p	
Kernel k	Gaussian	non-Gaussian
RBF/ polynomial/ trigonometric	analytical	analytical via importance-sampling trick
otherwise	Monte Carlo (numerical integration)	Monte Carlo (numerical integration)

Kernel expectations in other areas

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

- Kernel MMD
  (e.g., Gretton et al., 2012)

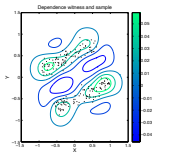


from Gretton et al. (2012)

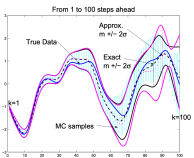
Kernel expectations in other areas

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

- Kernel MMD
  (e.g., Gretton et al., 2012)
- Time-series analysis with Gaussian processes
  (e.g., Girard et al., 2003)



from Gretton et al. (2012)

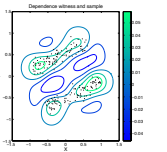


from Girard et al. (2003)

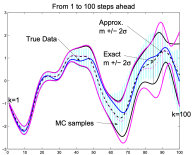
Kernel expectations in other areas

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

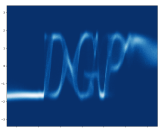
- Kernel MMD
  (e.g., Gretton et al., 2012)
- Time-series analysis with Gaussian processes
  (e.g., Girard et al., 2003)
- Deep Gaussian processes
  (e.g., Damianou & Lawrence, 2013)



from Gretton et al. (2012)



from Girard et al. (2003)

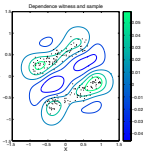


from Salimbeni et al. (2019)

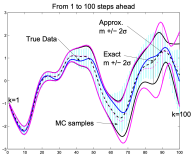
Kernel expectations in other areas

$$\mathbb{E}_{\boldsymbol{x}\sim p}[k(\boldsymbol{x}, \boldsymbol{X})], \quad \mathbb{E}_{\boldsymbol{x},\boldsymbol{x}'\sim p}[k(\boldsymbol{x}, \boldsymbol{x}')]$$

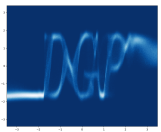
- Kernel MMD
  (e.g., Gretton et al., 2012)
- Time-series analysis with Gaussian processes
  (e.g., Girard et al., 2003)
- Deep Gaussian processes
  (e.g., Damianou & Lawrence, 2013)
- Model-based RL with Gaussian processes
  (e.g., Deisenroth & Rasmussen, 2011)



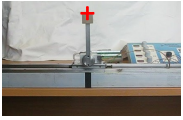
from Gretton et al. (2012)



from Girard et al. (2003)



from Salimbeni et al. (2019)



from Deisenroth & Rasmussen (2011)

Iterative procedure: Where to measure f next?

- Define an **acquisition function** (similar to Bayesian optimization)

Iterative procedure: Where to measure f next?

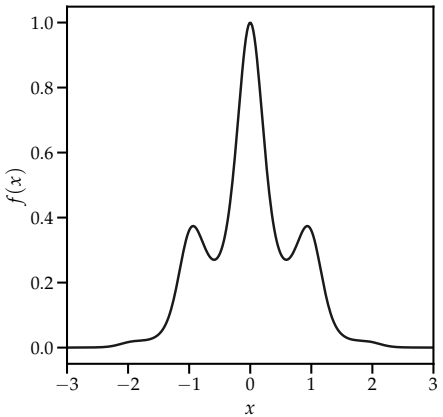
- Define an **acquisition function** (similar to Bayesian optimization)
- Example: Choose next node $\boldsymbol{x}_{n+1}$ so that the variance of the estimator is reduced maximally (e.g., O'Hagan, 1991; Gunter et al., 2014)

$$\boldsymbol{x}_{n+1} = \text{argmax}_{\boldsymbol{x}_*} \overbrace{\mathbb{V}[Z|\mathcal{D}]}^{\text{current variance}} - \overbrace{\mathbb{E}_{y_*}\Big[\mathbb{V}[Z|\mathcal{D}\cup\{(\boldsymbol{x}_*, y_*)\}]\Big]}^{\text{new variance}}$$

Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

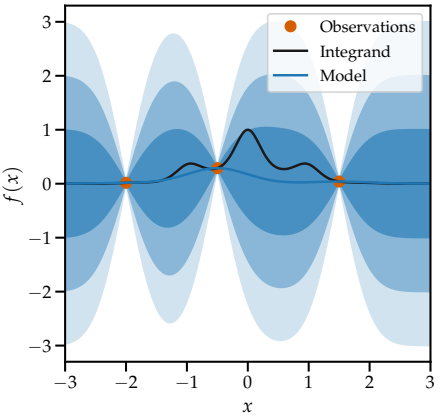


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$

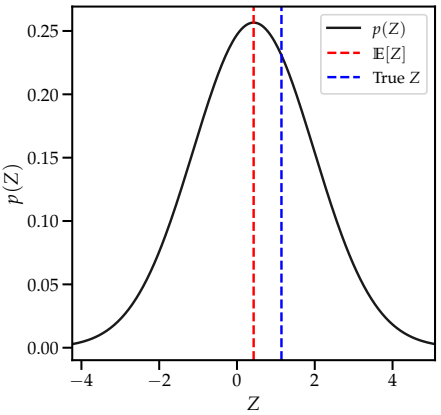


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$
- Determine $p(Z)$

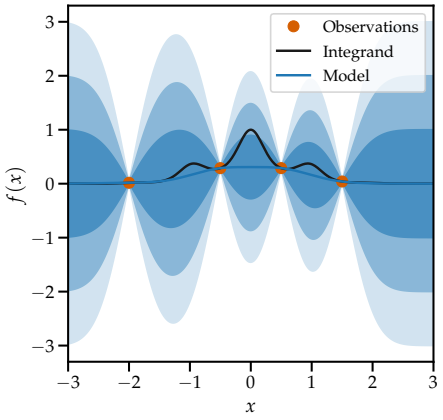


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$
- Determine $p(Z)$
- Find and include new measurement
  1. Find optimal node x_{n+1} by maximizing an acquisition function
  2. Evaluate integrand at x_{n+1}
  3. Update GP with $\big(x_{n+1}, f(x_{n+1})\big)$

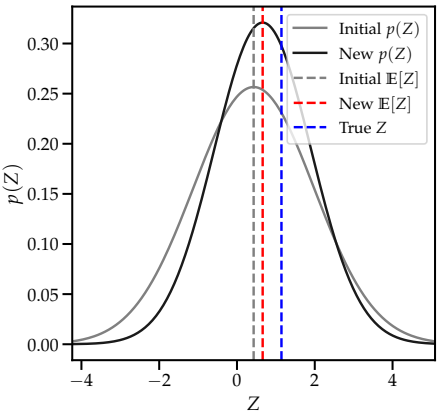


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$
- Determine $p(Z)$
- Find and include new measurement
- Compute updated $p(Z)$

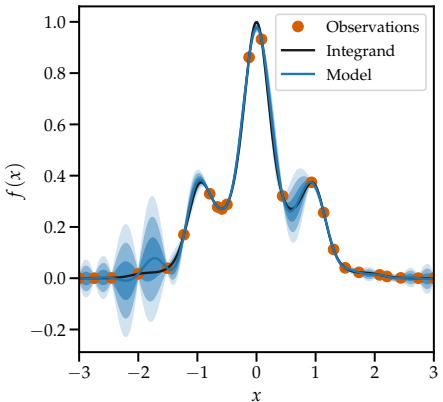


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$
- Determine $p(Z)$
- Find and include new measurement
- Compute updated $p(Z)$
- Repeat

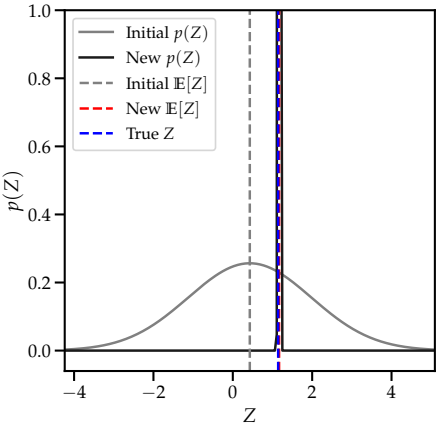


Example with EmuKit (Paleyes et al., 2019)

Compute

$$Z = \int_{-3}^{3} e^{-x^2 - \sin^2(3x)} dx$$

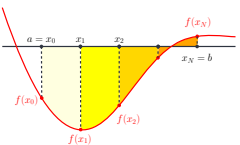
- Fit Gaussian process to observations $f(x_1), \ldots, f(x_n)$ at nodes $x_1, \ldots, x_n$
- Determine $p(Z)$
- Find and include new measurement
- Compute updated $p(Z)$
- Repeat



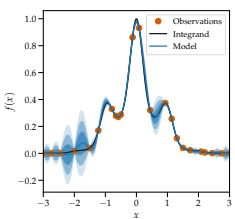
Summary

- Central approximation

$$\int f(\boldsymbol{x})d\boldsymbol{x} \approx \sum_{n=1}^{N} w_n f(\boldsymbol{x}_n)$$



- **Newton–Cotes:** Equidistant nodes $\boldsymbol{x}_n$, low-degree polynomial approximation of f
- **Gaussian quadrature:** Nodes $\boldsymbol{x}_n$ as the roots of interpolating orthogonal polynomials of f
- **Bayesian quadrature:** Integration as a statistical inference problem; Global approximation of f using a Gaussian process; scales to moderate dimensions



⏭ **Numerical integration is a really good idea in low dimensions**

References

Briol, F.-X., Oates, C., Girolami, M., and Osborne, M. A. (2015). Frank–Wolfe Bayesian Quadrature: Probabilistic Integration with Theoretical Guarantees. In *Advances in Neural Information Processing Systems*.

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

Damianou, A. and Lawrence, N. D. (2013). Deep Gaussian Processes. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

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

Deisenroth, M. P. and Mohamed, S. (2012). Expectation Propagation in Gaussian Process Dynamical Systems. In *Advances in Neural Information Processing Systems*, pages 2618–2626.

References (cont.)

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

Deisenroth, M. P., Turner, R., Huber, M., Hanebeck, U. D., and Rasmussen, C. E. (2012). Robust Filtering and Smoothing with Gaussian Processes. *IEEE Transactions on Automatic Control*, 57(7):1865–1871.

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

Eleftheriadis, S., Nicholson, T. F. W., Deisenroth, M. P., and Hensman, J. (2017). Identification of Gaussian Process State Space Models. In *Advances in Neural Information Processing Systems*.

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

References (cont.)

Girard, A., Rasmussen, C. E., Quiñonero Candela, J., and Murray-Smith, R. (2003). Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In *Advances in Neural Information Processing Systems*.

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

Gunter, T., Osborne, M. A., Garnett, R., Hennig, P., and Roberts, S. J. (2014). Sampling for Inference in Probabilistic Models with Fast Bayesian Quadrature. In *Advances in Neural Information Processing Systems*.

Hennig, P., Osborne, M. A., and Girolami, M. (2015). Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471:20150142.

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

References (cont.)

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

Paleyes, A., Pullin, M., Mahsereci, M., Lawrence, N., and González, J. (2019). Emulation of Physical Processes with Emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*.

Salimbeni, H. and Deisenroth, M. P. (2017). Doubly Stochastic Variational Inference for Deep Gaussian Processes. In *Advances in Neural Information Processing Systems*.

Salimbeni, H., Dutordoir, V., Hensman, J., and Deisenroth, M. P. (2019). Deep Gaussian Processes with Importance-Weighted Variational Inference. In *Proceedings of the International Conference on Machine Learning*.

Stoer, J. and Bulirsch, R. (2002). *Introduction to Numerical Analysis*. Texts in Applied Mathematics. Springer-Verlag, 3rd edition.

Inference in Time Series

Cheng Soon Ong
Marc Peter Deisenroth

December 2020

Setting

- Time-series model

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

Setting

- Time-series model

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

- Compute an expected utility of a state trajectory $\boldsymbol{\tau} := (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$

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

Setting

- Time-series model

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

- Compute an expected utility of a state trajectory $\boldsymbol{\tau} := (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$

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

  - Reinforcement learning and optimal control
  - Demand forecasting (logistics)
  - Weather/climate forecasts

Setting

- Time-series model

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

- Compute an expected utility of a state trajectory $\boldsymbol{\tau} := (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_T)$

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

  - Reinforcement learning and optimal control
  - Demand forecasting (logistics)
  - Weather/climate forecasts

- Challenge: Long-term predictions and uncertainty propagation

Approaches

- **Deterministic inference via iterative computation**
  - Iteratively determine marginal distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$
  - Compute expectations $\mathbb{E}_{\boldsymbol{x}_t}[u(\boldsymbol{x}_t)]$ and compute utilities of the form

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

Approaches

- **Deterministic inference via iterative computation**
  - Iteratively determine marginal distributions $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$
  - Compute expectations $\mathbb{E}_{\boldsymbol{x}_t}[u(\boldsymbol{x}_t)]$ and compute utilities of the form

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

- **Stochastic inference via trajectory sampling**
  - Generate sample trajectories $\boldsymbol{\tau}^{(s)} = (\boldsymbol{x}_0^{(s)}, \ldots, \boldsymbol{x}_T^{(s)})$
  - Monte-Carlo integration

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

Deterministic Approximate Inference

Deterministic approximate inference

- Iteratively compute marginals

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

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

Deterministic approximate inference

- Iteratively compute marginals

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

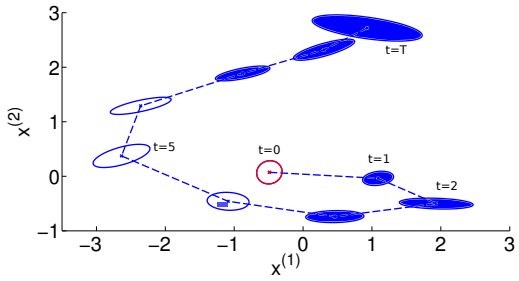
$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\big(\mathbf{0},\, \boldsymbol{Q}\big)$$

⏩ No closed-form solution for nonlinear f

Iterative Gaussian approximation

- Common approach: Iterative Gaussian approximation of marginals:

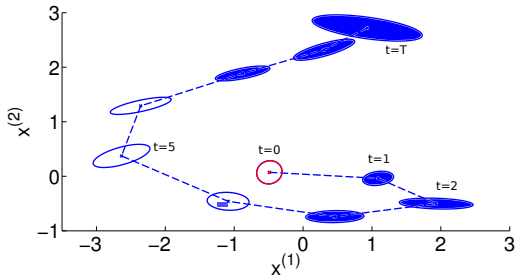
$$p(\boldsymbol{x}_t) \approx \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$$



Iterative Gaussian approximation

- Common approach: Iterative Gaussian approximation of marginals:

$$p(\boldsymbol{x}_t) \approx \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$$



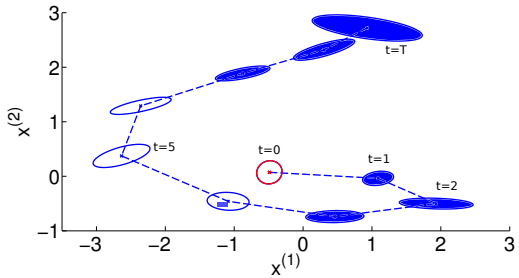
- **Linearization**

⏩ Extended Kalman filter

Iterative Gaussian approximation

- Common approach: Iterative Gaussian approximation of marginals:

$$p(\boldsymbol{x}_t) \approx \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$$



- **Linearization**
- **Unscented transformation**

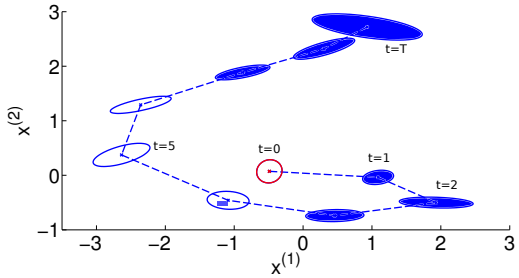
⏩ Extended Kalman filter

⏩ Unscented Kalman filter

Iterative Gaussian approximation

- Common approach: Iterative Gaussian approximation of marginals:

$$p(\boldsymbol{x}_t) \approx \mathcal{N}\big(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t\big)$$



- **Linearization**
- **Unscented transformation**
- **Moment matching**

- ⏩ Extended Kalman filter
- ⏩ Unscented Kalman filter
- ⏩ Assumed density filter

Two approaches

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

Two approaches

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

- Approximate f
- Approximate $p(\boldsymbol{x}_t)$

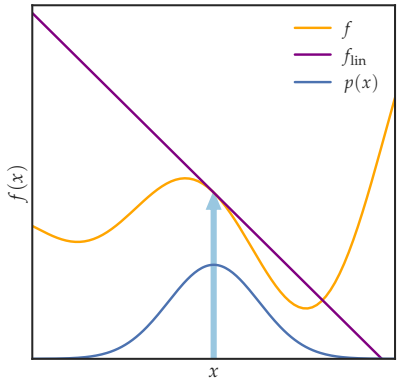
⏩ Linearization (e.g., Smith et al., 1962)

⏩ Unscented transformation (Julier & Uhlmann, 1995)

Approach 1: Linearization

Key idea (e.g., Smith et al., 1962; Ohab & Stubberud, 1965)

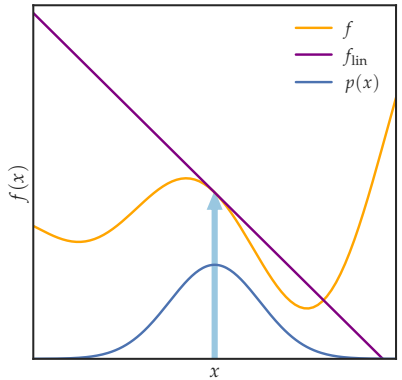
1. Locally linearize f around mean $\boldsymbol{\mu}_t$
2. Compute predictive distribution (Gaussian) for linearized function in closed form



Approach 1: Linearization

Key idea (e.g., Smith et al., 1962; Ohab & Stubberud, 1965)

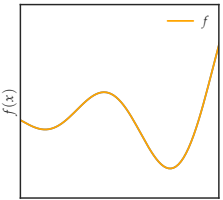
1. Locally linearize f around mean $\boldsymbol{\mu}_t$
2. Compute predictive distribution (Gaussian) for linearized function in closed form

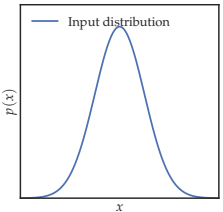


- Linearization: First-order Taylor-series expansion around $\boldsymbol{\mu}_t$
  - ⏩ Gradient (Jacobian) $df/d\boldsymbol{x}_t$ of f evaluated at $\boldsymbol{\mu}_t$
- Key insight: Gaussians can be pushed through linear functions in closed form

How it works

- Compute gradient $\boldsymbol{J}_t := df/d\boldsymbol{x}_t|_{\boldsymbol{x}_t=\boldsymbol{\mu}_t}$

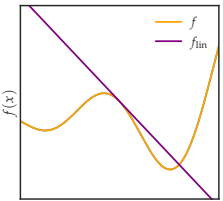


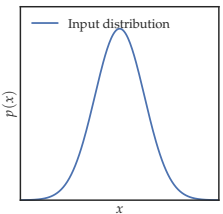


How it works

- Compute gradient $\boldsymbol{J}_t := df/d\boldsymbol{x}_t|_{\boldsymbol{x}_t=\boldsymbol{\mu}_t}$
- Linearized model:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{\mu}_t) + \boldsymbol{J}_t(\boldsymbol{x} - \boldsymbol{\mu}_t)$$





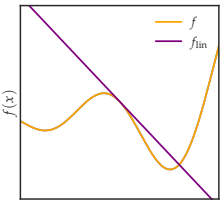
How it works

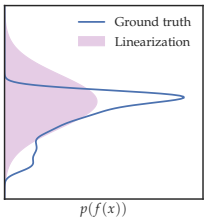
- Compute gradient $\boldsymbol{J}_t := df/d\boldsymbol{x}_t|_{\boldsymbol{x}_t=\boldsymbol{\mu}_t}$
- Linearized model:

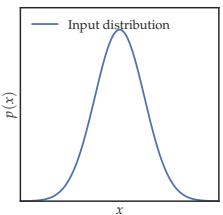
$$f(\boldsymbol{x}) \approx f(\boldsymbol{\mu}_t) + \boldsymbol{J}_t(\boldsymbol{x} - \boldsymbol{\mu}_t)$$

- Approximate predictive distribution is Gaussian:

$$p(f(\boldsymbol{x}_t)) \approx \mathcal{N}\big(f(\boldsymbol{\mu}_t),\, \boldsymbol{J}_t\boldsymbol{\Sigma}_t\boldsymbol{J}_t^\top\big)$$
$$p(\boldsymbol{x}_{t+1}) \approx \mathcal{N}\big(f(\boldsymbol{\mu}_t),\, \boldsymbol{J}_t\boldsymbol{\Sigma}_t\boldsymbol{J}_t^\top + \boldsymbol{Q}\big)$$

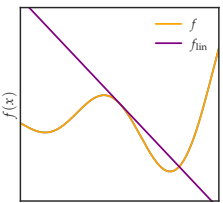


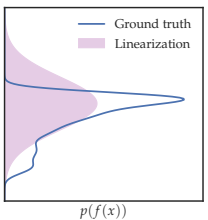


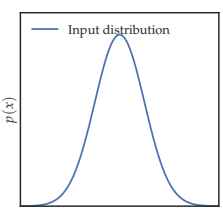


Linearization: Properties

- Conceptually straightforward
- Requires differentiable f
- Tends to underestimate true covariance matrix ⏩ Overconfidence
- Scales cubically in the dimension of $\boldsymbol{x}$
- Widely used in engineering (e.g., navigation systems, GPS, Apollo missions)



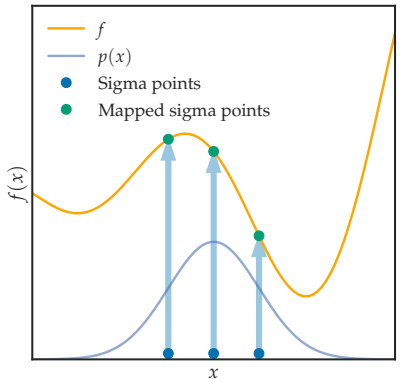




Approach 2: Unscented transformation

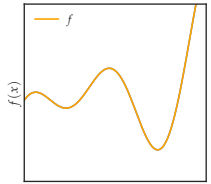
Key idea (Julier & Uhlmann, 1995)

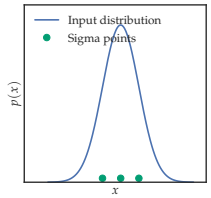
1. Approximate $p(\boldsymbol{x}_t)$ using a small set of deterministically chosen sigma points
2. Map sigma points through f
3. Compute a weighted average of the mean and covariance of the predictive distribution.



How it works

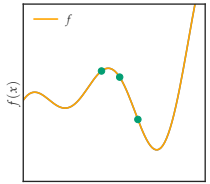
- Determine $2D + 1$ sigma points
  $\mathcal{X}_t = \{\boldsymbol{\mu}_t \pm \alpha(\sqrt{\boldsymbol{\Sigma}_t})_i,\ i = 1, \ldots, D\}$

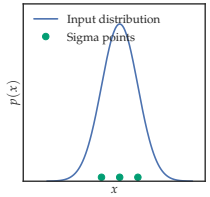




How it works

- Determine $2D + 1$ sigma points
  $\mathcal{X}_t = \{\boldsymbol{\mu}_t \pm \alpha(\sqrt{\boldsymbol{\Sigma}_t})_i,\ i = 1, \ldots, D\}$
- Map sigma points through f to get $f(\mathcal{X}_t)$



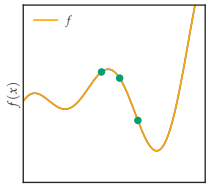


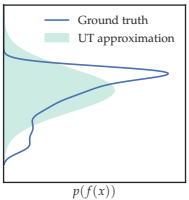
How it works

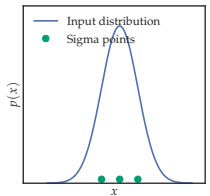
- Determine $2D + 1$ sigma points $\mathcal{X}_t = \{\boldsymbol{\mu}_t \pm \alpha(\sqrt{\boldsymbol{\Sigma}_t})_i,\ i = 1, \ldots, D\}$
- Map sigma points through f to get $f(\mathcal{X}_t)$
- Compute mean/covariance of predictive distribution $p(f(\boldsymbol{x}_t))$ as a weighted average

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

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

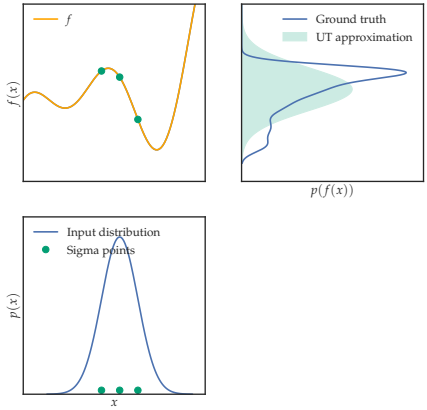






Unscented transformation: Properties

- Not a Monte-Carlo method: Sigma points are deterministic, not random
- No explicit calculation of Jacobians
  ⏩ f can be non-differentiable
- Input distribution does not need to be Gaussian
- Higher accuracy (covariance) than linearization (Julier & Uhlmann, 2004)



Stochastic Approximate Inference

Stochastic approximate inference

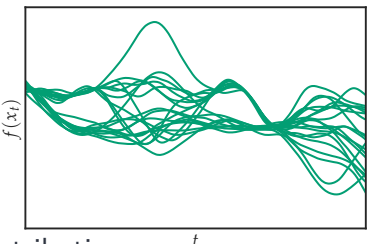
- Sample trajectories $\boldsymbol{\tau}^{(i)} = \left(\boldsymbol{x}_0^{(i)}, \ldots, \boldsymbol{x}_T^{(i)}\right)$:
  1. Sample initial state: $\boldsymbol{x}_0^{(i)} \sim p(\boldsymbol{x}_0)$
  2. For $t = 1, \ldots, T$:

$$\boldsymbol{x}_t^{(i)} \sim p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}^{(i)}) = \mathcal{N}\left(\boldsymbol{x}_t \middle| f(\boldsymbol{x}_{t-1}^{(i)}), \boldsymbol{Q}\right)$$

Stochastic approximate inference

- Sample trajectories $\boldsymbol{\tau}^{(i)} = \left(\boldsymbol{x}_0^{(i)}, \ldots, \boldsymbol{x}_T^{(i)}\right)$:
  1. Sample initial state: $\boldsymbol{x}_0^{(i)} \sim p(\boldsymbol{x}_0)$
  2. For $t = 1, \ldots, T$:

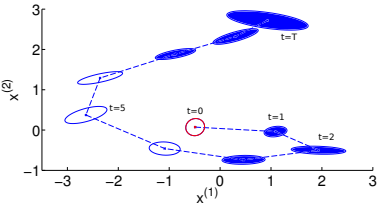
$$\boldsymbol{x}_t^{(i)} \sim p(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}^{(i)}) = \mathcal{N}\left(\boldsymbol{x}_t \middle| f(\boldsymbol{x}_{t-1}^{(i)}), \boldsymbol{Q}\right)$$

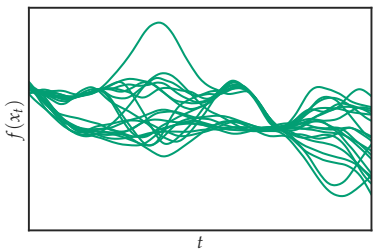


- No parametric restriction to a specific kind of distribution
- Have to store all samples (particles) ⏩ Potential memory issue
- Sequential Monte Carlo (particle filter)
  (e.g., Doucet et al., 2000; Thrun et al., 2005;)

Discussion: Long-term predictions

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





Inference in Gaussian Process Time Series Models

Setting

- Time-series model

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

Setting

- Time-series model

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

- Two approaches for long-term predictions:
  - Deterministic approximate inference of the marginals $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$
  - Stochastic approximate inference by sampling trajectories $\boldsymbol{\tau}^{(i)} = (\boldsymbol{x}_0^{(i)}, \ldots, \boldsymbol{x}_T^{(i)})$

Deterministic approximate inference

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

Deterministic approximate inference

$$
\begin{aligned}
p(\boldsymbol{x}_{t+1}) &= \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t)d\boldsymbol{x}_t \\
&= \int \int p(f(\boldsymbol{x}_t)|f, \boldsymbol{x}_t)p(f)df p(\boldsymbol{x}_t)d\boldsymbol{x}_t
\end{aligned}
$$

Deterministic approximate inference

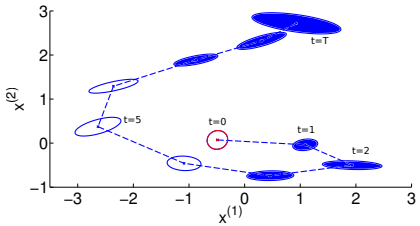
$$
\begin{aligned}
p(\boldsymbol{x}_{t+1}) &= \int p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)p(\boldsymbol{x}_t)d\boldsymbol{x}_t \\
&= \int \int p(f(\boldsymbol{x}_t)|f, \boldsymbol{x}_t)p(f)df p(\boldsymbol{x}_t)d\boldsymbol{x}_t
\end{aligned}
$$

Approaches:

- Linearization (e.g., Ko & Fox, 2009)
- Unscented transformation (e.g., Ko & Fox, 2009)
- Moment matching (e.g., Deisenroth et al., 2009)

Long-term predictions

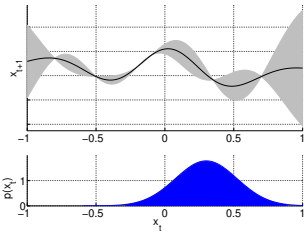
- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$



Long-term predictions

- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$

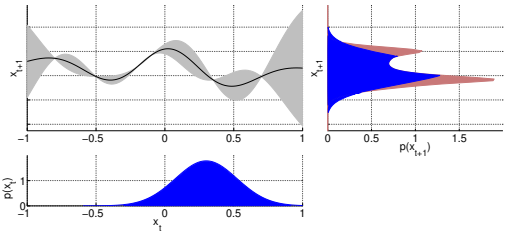
$$\underbrace{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)}_{\text{GP prediction}} \quad \underbrace{p(\boldsymbol{x}_t)}_{\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)}$$



Long-term predictions

- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$

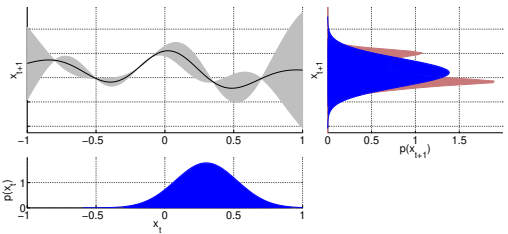
$$p(\boldsymbol{x}_{t+1}) = \iint \underbrace{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)}_{\text{GP prediction}} \underbrace{p(\boldsymbol{x}_t)}_{\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} df\, d\boldsymbol{x}_t$$



Long-term predictions

- Iteratively compute $p(\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_T)$

$$p(\boldsymbol{x}_{t+1}) = \iint \underbrace{p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)}_{\text{GP prediction}} \underbrace{p(\boldsymbol{x}_t)}_{\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} df \, d\boldsymbol{x}_t$$

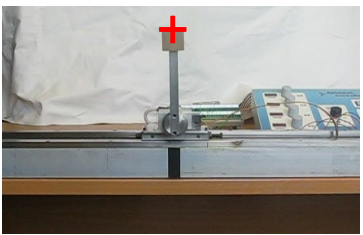


- GP moment matching (Girard et al., 2003; Quiñonero-Candela et al., 2003)
- Key ingredient: Computing kernel expectations

Example: Model-based reinforcement learning

- Learn dynamics of a physical system from data ⏭ Gaussian process

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}, \quad f \sim GP$$
$$\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\theta})$$



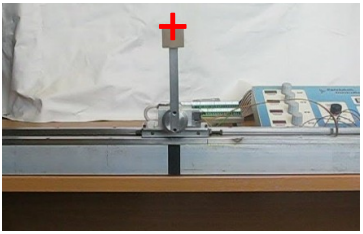
From Deisenroth & Rasmussen (2011)
https://www.youtube.com/PilcoLearner

Example: Model-based reinforcement learning

- Learn dynamics of a physical system from data ⏭ Gaussian process
- Given the learned system, find policy parameters $\boldsymbol{\theta}^*$ that minimize an expected long-term cost

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$$

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}, \quad f \sim GP$$
$$\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\theta})$$



From Deisenroth & Rasmussen (2011)
https://www.youtube.com/PilcoLearner

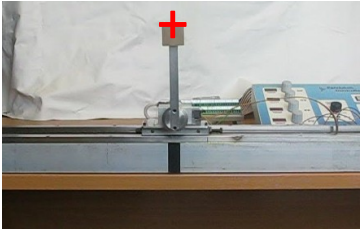
Example: Model-based reinforcement learning

- Learn dynamics of a physical system from data ⏭ Gaussian process
- Given the learned system, find policy parameters $\boldsymbol{\theta}^*$ that minimize an expected long-term cost

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$$

- GP moment matching for long-term predictions

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}, \quad f \sim GP$$
$$\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\theta})$$



From Deisenroth & Rasmussen (2011)
https://www.youtube.com/PilcoLearner

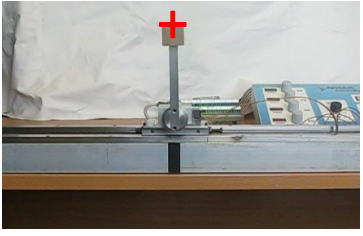
Example: Model-based reinforcement learning

- Learn dynamics of a physical system from data ⏭ Gaussian process
- Given the learned system, find policy parameters $\boldsymbol{\theta}^*$ that minimize an expected long-term cost

$$\mathbb{E}_{\boldsymbol{\tau}}[U(\boldsymbol{\tau})] = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)]$$

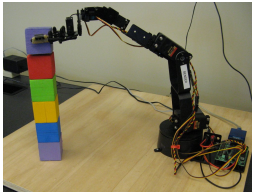
- GP moment matching for long-term predictions
- Gradient descent to find $\boldsymbol{\theta}^*$

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\epsilon}, \quad f \sim GP$$
$$\boldsymbol{u}_t = \pi(\boldsymbol{x}_t; \boldsymbol{\theta})$$

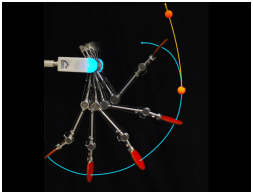


From Deisenroth & Rasmussen (2011)
https://www.youtube.com/PilcoLearner

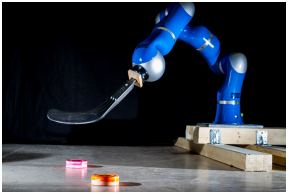
Wide Applicability



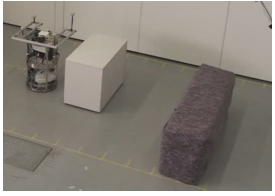
Deisenroth et al. (2011)



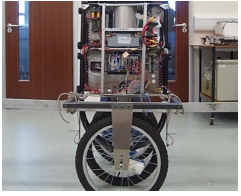
Englert et al. (2013)



Kupcsik et al. (2017)



Bischoff et al. (2013b)



McHutchon (2014)

Bischoff et al. (2013a)

- Application to a wide range of robotic systems

Stochastic approximate inference

- Generating a function draw: Sampling from a T-dimensional multivariate Gaussian
  T: Number of query points

Figure: Generated with GPflow (Matthews et al., 2017)

Stochastic approximate inference

- Generating a function draw: Sampling from a T-dimensional multivariate Gaussian
  T: Number of query points
- Drawing a sample from a GP **scales cubically in T**

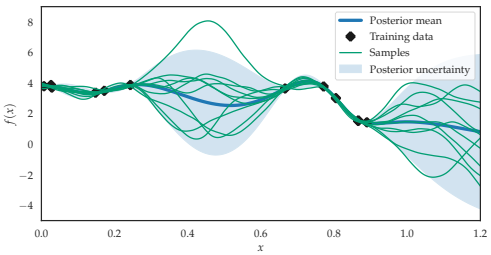


Figure: Generated with GPflow (Matthews et al., 2017)

Stochastic approximate inference

- Generating a function draw: Sampling from a T-dimensional multivariate Gaussian
  T: Number of query points
- Drawing a sample from a GP **scales cubically in T**
- There are some ways around this in low dimensions (e.g., Särkkä et al., 2013; Solin et al., 2018) or by making structural assumptions (e.g., Pleiss et al., 2018)

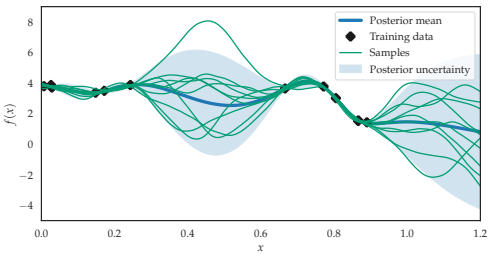


Figure: Generated with GPflow (Matthews et al., 2017)

Stochastic approximate inference

- Generating a function draw: Sampling from a T-dimensional multivariate Gaussian
  T: Number of query points
- Drawing a sample from a GP **scales cubically in T**
- There are some ways around this in low dimensions (e.g., Särkkä et al., 2013; Solin et al., 2018) or by making structural assumptions (e.g., Pleiss et al., 2018)

  ⏩ Let's try something else

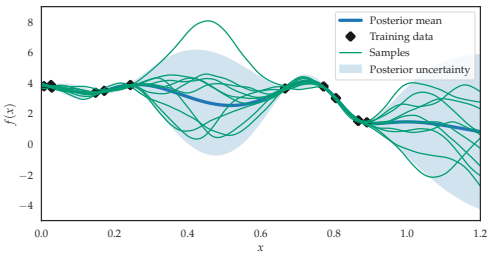


Figure: Generated with GPflow (Matthews et al., 2017)

Decoupled sampling (Wilson et al., 2020a)

Key idea

Sample functions from a Gaussian process by exploiting **Matheron's rule** (for Gaussian random variables):

$$\text{posterior} = \text{prior} + \text{data-dependent update}$$

Decoupled sampling (Wilson et al., 2020a)

Key idea

Sample functions from a Gaussian process by exploiting **Matheron's rule** (for Gaussian random variables):

$$\text{posterior} = \text{prior} + \text{data-dependent update}$$

- Think about the posterior in terms of samples, not in terms of (conditional) distributions

Decoupled sampling (Wilson et al., 2020a)

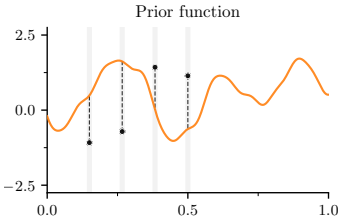
Key idea

Sample functions from a Gaussian process by exploiting **Matheron's rule** (for Gaussian random variables):

$$\text{posterior} = \text{prior} + \text{data-dependent update}$$

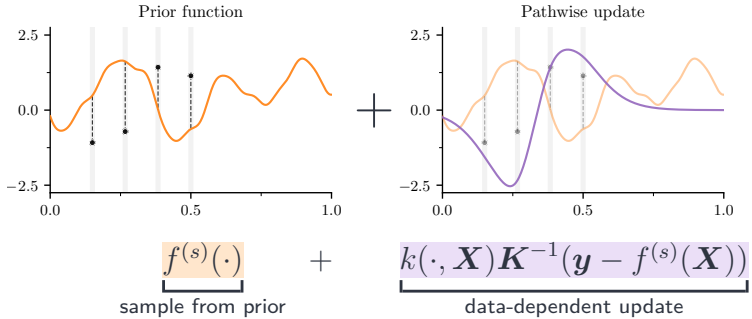
- Think about the posterior in terms of samples, not in terms of (conditional) distributions
- Samples from the posterior can be obtained through a two-step procedure:
  1. Sample from prior ⏭ Source of randomness
  2. "Correct" sample using a data-dependent update term ⏭ Deterministic transformation

Illustration: Decoupled sampling (Wilson et al., 2020a)



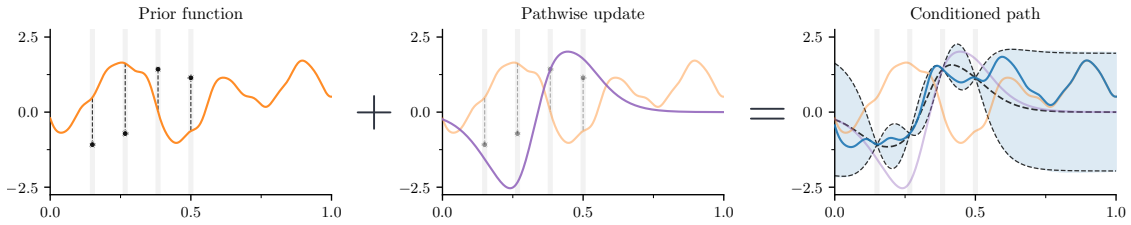
1. Sample from the prior

Illustration: Decoupled sampling (Wilson et al., 2020a)



1. Sample from the prior
2. Add data-dependent update term

Illustration: Decoupled sampling (Wilson et al., 2020a)

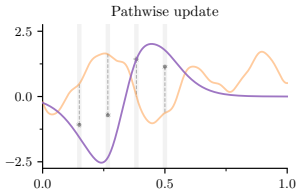


1. Sample from the prior
2. Add data-dependent update term

⏩ Sample from the posterior

Properties

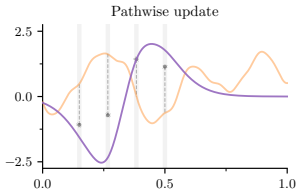
$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$

Properties

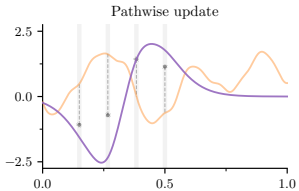
$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$
- Update term is a mapping from prior to posterior

Properties

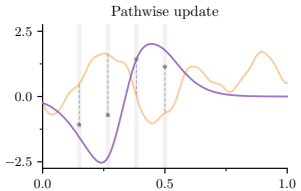
$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$
- Update term is a mapping from prior to posterior
- Different representations for prior and update terms
  (e.g., RFF for prior and finite basis-function representation for update)

Properties

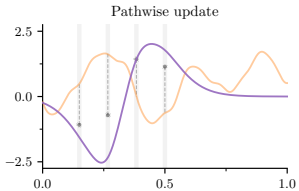
$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$
- Update term is a mapping from prior to posterior
- Different representations for prior and update terms
  (e.g., RFF for prior and finite basis-function representation for update)
  - Sampling from RFF prior scales linearly in the number T of test inputs

Properties

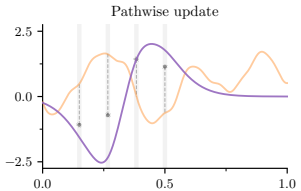
$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$
- Update term is a mapping from prior to posterior
- Different representations for prior and update terms
  (e.g., RFF for prior and finite basis-function representation for update)
  - Sampling from RFF prior scales linearly in the number T of test inputs
  - Update term can be computed linearly in the number T of test inputs

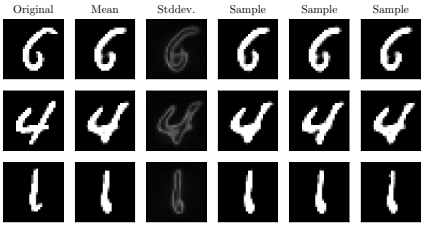
Properties

$$\underbrace{f^{(s)}(\cdot)}_{\text{sample from prior}} + \underbrace{k(\cdot, \boldsymbol{X})\boldsymbol{K}^{-1}(\boldsymbol{y} - f^{(s)}(\boldsymbol{X}))}_{\text{data-dependent update}} = \underbrace{f^{(s)}(\cdot)|\boldsymbol{X}, \boldsymbol{y}}_{\text{sample from posterior}}$$



- Update term depends on error/residual between the prior sample and data $\boldsymbol{y}$
- Update term is a mapping from prior to posterior
- Different representations for prior and update terms
  (e.g., RFF for prior and finite basis-function representation for update)
  - Sampling from RFF prior scales linearly in the number T of test inputs
  - Update term can be computed linearly in the number T of test inputs
- **Functions can be sampled efficiently** (linearly in the number of test inputs)

Applications

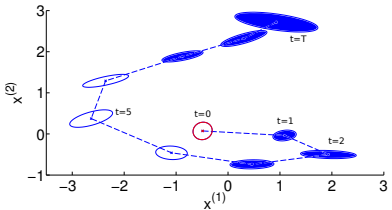


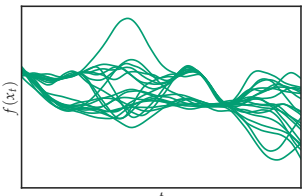
From Wilson et al. (2020b)

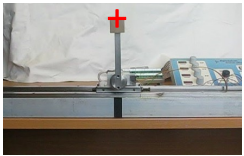
From Borovitskiy et al. (2020)

- 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)
- Model-based reinforcement learning

Summary







- Propagate uncertainty through a nonlinear dynamical system
- Deterministic approximate inference (linearization, unscented transformation)
- Stochastic approximate inference (sampling)
- Examples in the context of GP dynamical systems

References

Bischoff, B., Nguyen-Tuong, D., Koller, T., Markert, H., and Knoll, A. (2013a). Learning Throttle Valve Control Using Policy Search. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*.

Bischoff, B., Nguyen-Tuong, D., Markert, H., and Knoll, A. (2013b). Learning Control Under Uncertainty: A Probabilistic Value-Iteration Approach. In *Proceedings of the European Symposium on Artificial Neural Networks*.

Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. (2020). Matern Gaussian Processes on Riemannian Manifolds. In *Advances in Neural Information Processing Systems*.

Deisenroth, M. P., Huber, M. F., and Hanebeck, U. D. (2009). Analytic Moment-based Gaussian Process Filtering. In *Proceedings of the International Conference on Machine Learning*.

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

References (cont.)

Deisenroth, M. P., Rasmussen, C. E., and Fox, D. (2011). Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *Proceedings of Robotics: Science and Systems*.

Doucet, A., Godsill, S. J., and Andrieu, C. (2000). On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10:197–208.

Englert, P., Paraschos, A., Peters, J., and Deisenroth, M. P. (2013). Model-based Imitation Learning by Probabilistic Trajectory Matching. In *Proceedings of the International Conference on Robotics and Automation*.

Girard, A., Rasmussen, C. E., Quiñonero Candela, J., and Murray-Smith, R. (2003). Gaussian Process Priors with Uncertain Inputs—Application to Multiple-Step Ahead Time Series Forecasting. In *Advances in Neural Information Processing Systems*.

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

References (cont.)

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

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

Kupcsik, A., Deisenroth, M. P., Peters, J., Poha, L. A., Vadakkepata, P., and Neumann, G. (2017). Model-based Contextual Policy Search for Data-Efficient Generalization of Robot Skills. *Artificial Intelligence*.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian Process Library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6.

McHutchon, A. (2014). *Nonlinear Modelling and Control usingGaussian Processes*. PhD thesis, University of Cambridge.

References (cont.)

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

Pleiss, G., Gardner, J. R., Weinberger, K. Q., and Wilson, A. G. (2018). Constant-Time Predictive Distributions for Gaussian Processes. In *Proceedings of the International Conference on Machine Learning*.

Quiñonero-Candela, J., Girard, A., Larsen, J., and Rasmussen, C. E. (2003). Propagation of Uncertainty in Bayesian Kernel Models—Application to Multiple-Step Ahead Forecasting. In *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Särkkä, S., Solin, A., and Hartikainen, J. (2013). Spatiotemporal Learning via Infinite-Dimensional Bayesian Filtering and Smoothing: A Look at Gaussian Process Regression Through Kalman Filtering,. *IEEE Signal Processing Magazine*, 30(4):51–61.

Smith, G. L., Schmidt, S. F., and McGee, L. A. (1962). Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity on Board a Circumlunar Vehicle. Technical report, NASA.

References (cont.)

Solin, A., Hensman, J., and Turner, R. E. (2018). Infinite-Horizon Gaussian Processes. In *Advances in Neural Information Processing Systems*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press.

Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. (2020a). Efficiently Sampling Functions from Gaussian Process Posteriors. In *Proceedings of the International Conference on Machine Learning*.

Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. (2020b). Pathwise Conditioning of Gaussian Processes. *arXiv:2011.04026*.