Probabilistic Numerics

VI – Outlook & Conclusions

Philipp Hennig

Dobbiaco Summer School
23 June 2017

Research Group for Probabilistic Numerics
Max Planck Institute for Intelligent Systems
Tübingen, Germany

Some of the presented work was supported by
the Emmy Noether Programme of the DFG

this lecture:

- two less-explored directions of probabilistic numerics
- summary
- discussion!

Computation as Inference

New directions arising from probabilistic interpretation of computing

Estimate z from computations c, under model m.

Prior: structural knowledge reduces complexity

Likelihood: modeling imprecise computation reduces cost

$$p(z \mid c, m) = \frac{p(z \mid m)p(c \mid z, m)}{\int p(z \mid m)p(c \mid z, m)\, dz}$$

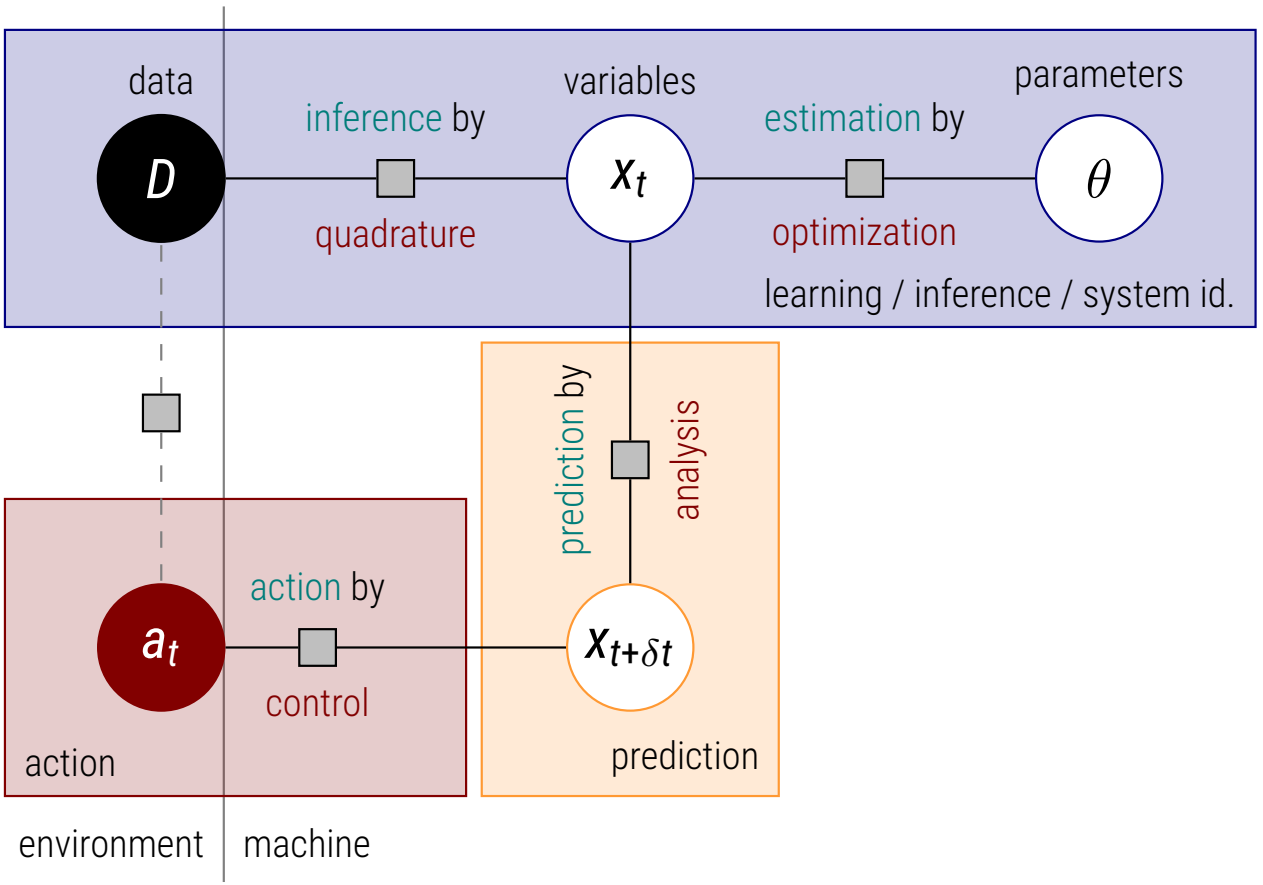
Posterior: tracking uncertainty for robustness

Evidence: checking models for safety

Chains of Computation

Probabilistic Numerics in the Loop

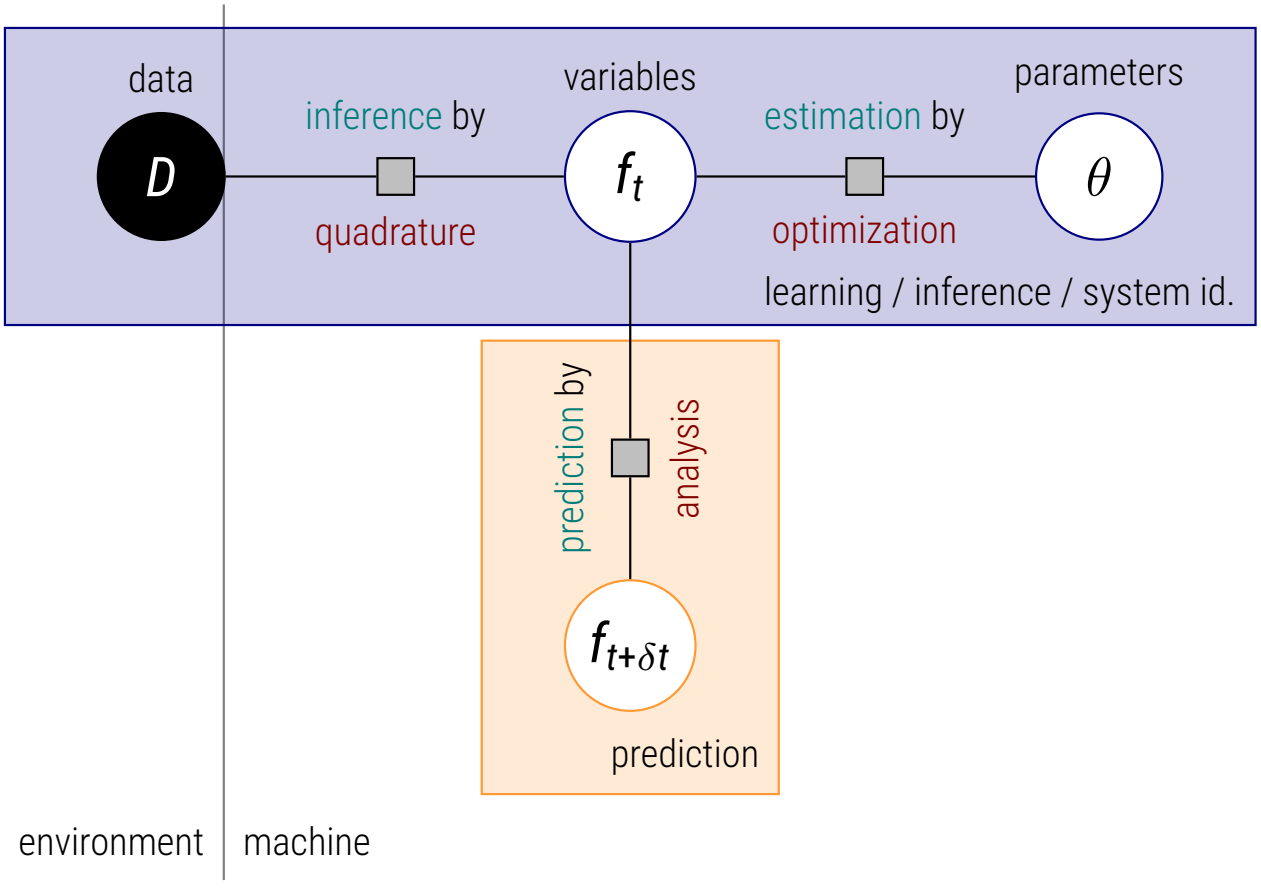
Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015



Example : GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

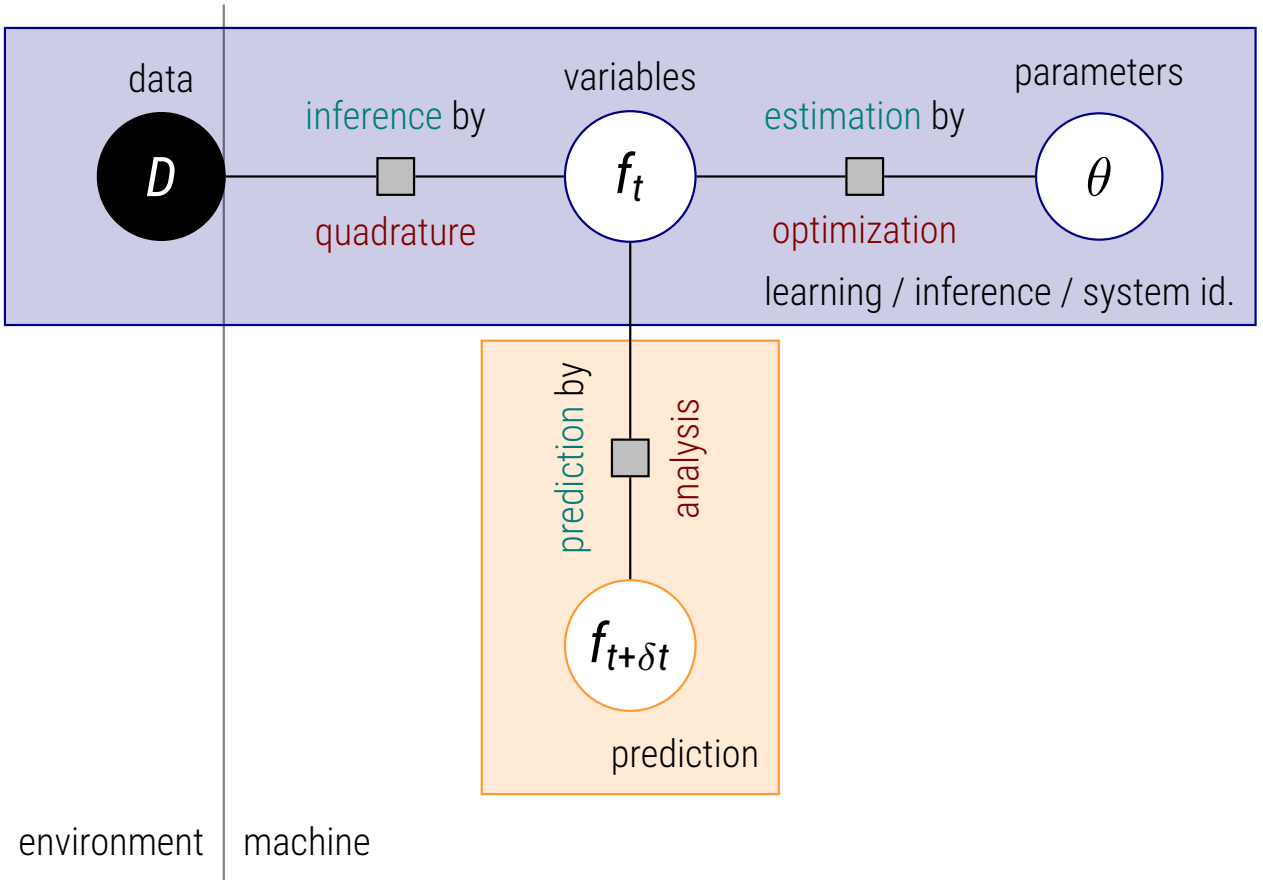


- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta)$, $p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2)$, $p(f_{t+\delta t} \mid f_t) = \mathcal{N}(f_{t+\delta t}, Af_t, Q)$

Example : GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

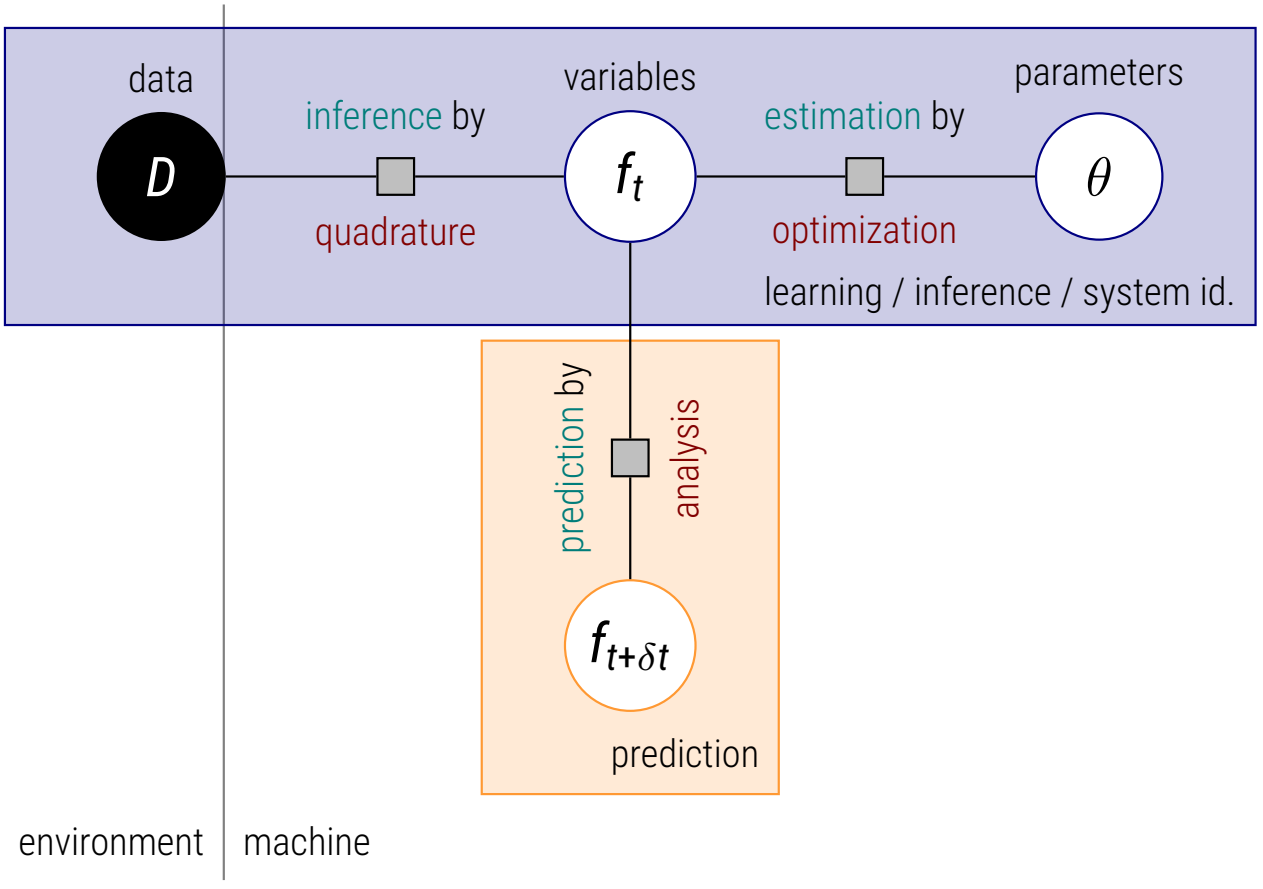


- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta)$, $p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2)$, $p(f_{t+\delta t} \mid f_t) = \mathcal{N}(f_{t+\delta t}, Af_t, Q)$
- inference: lin. algebra. optimization: nonlinear. prediction: lin. algebra.

Example : GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

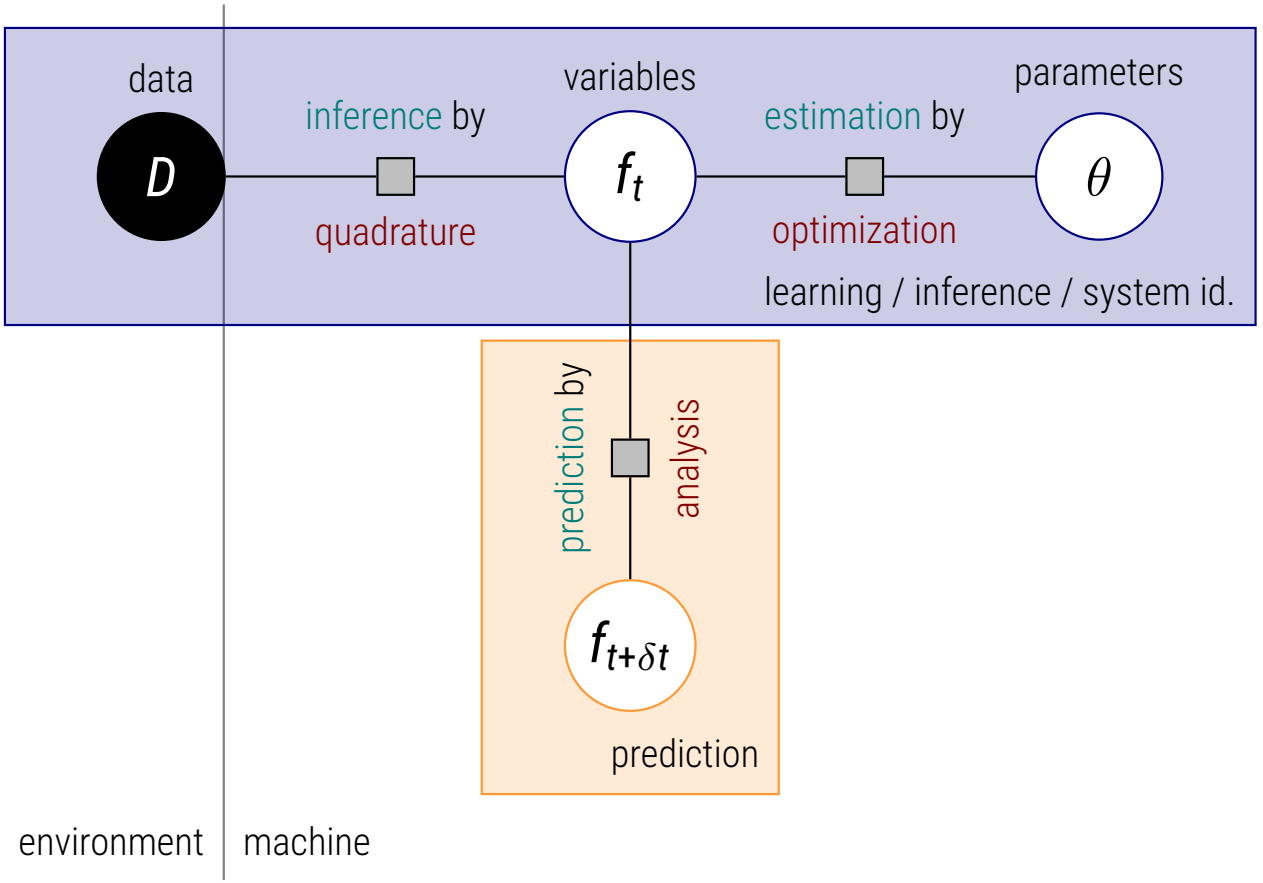


- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta), p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2), f'(t) = g(f_t, t)$

Example : GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

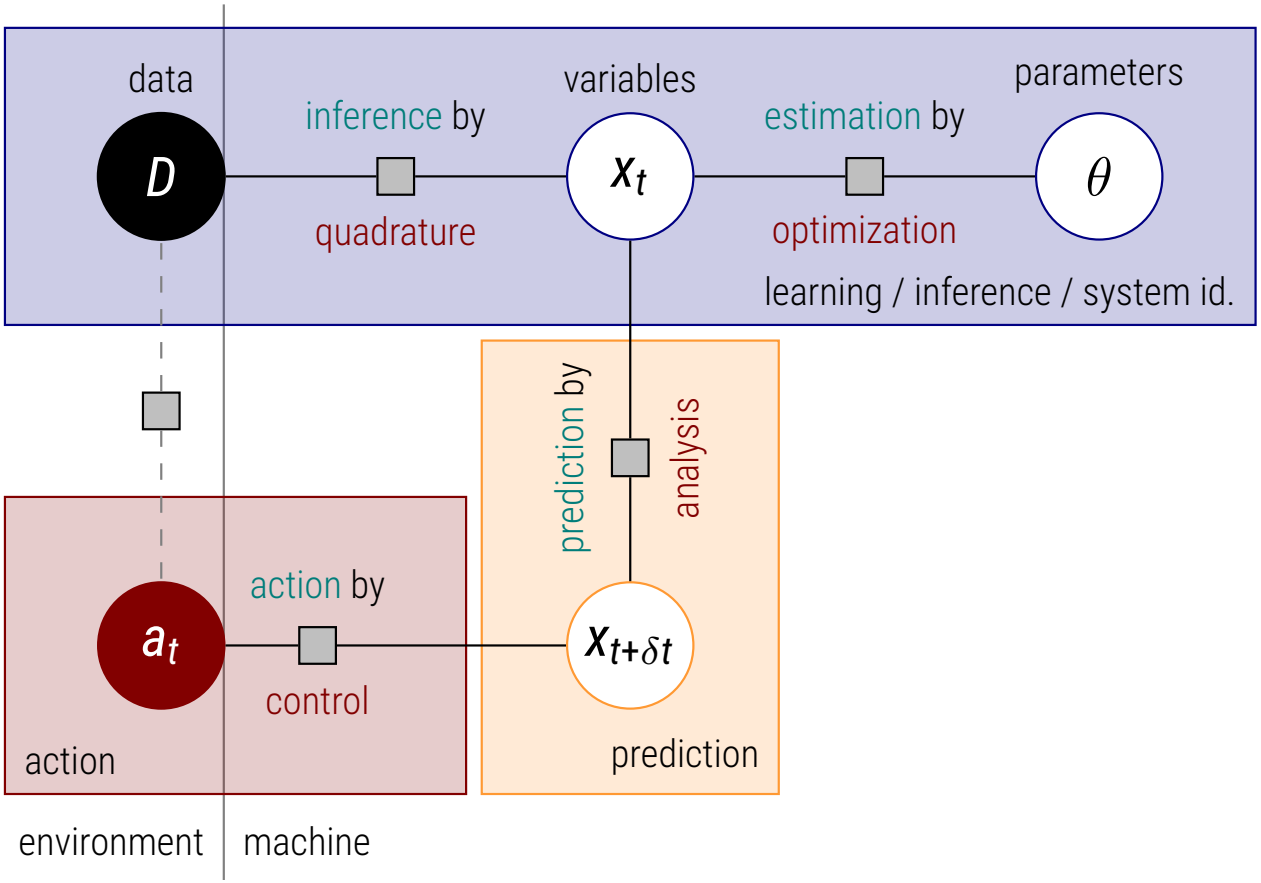


- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta)$, $p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2)$, $f'(t) = g(f_t, t)$
- inference: lin. algebra. optimization: nonlinear. prediction: ODE.

Example: GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

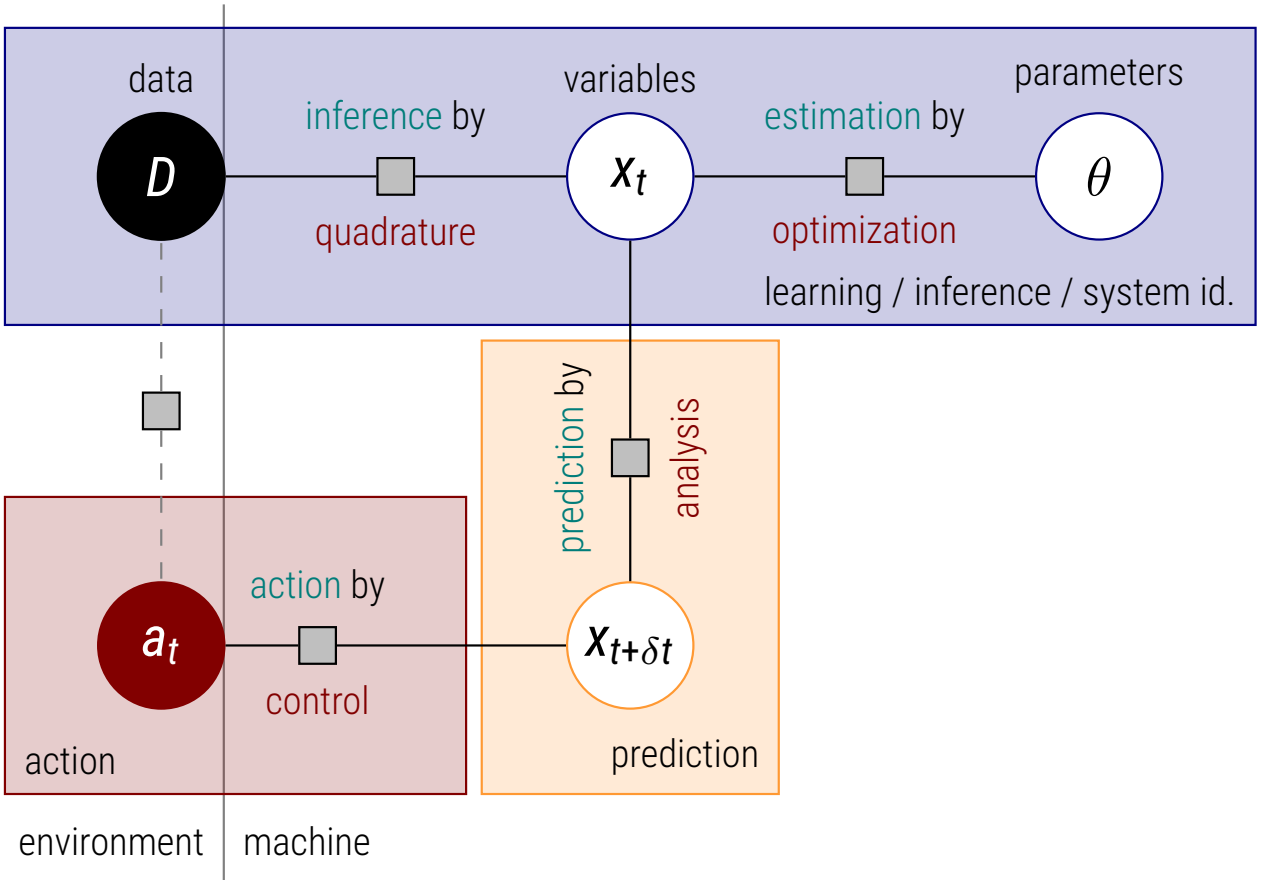


- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta), p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2), f'(t) = g(f_t, t, a_t)$

Example: GP regression

not all models are equal, but the high-level view fits

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015



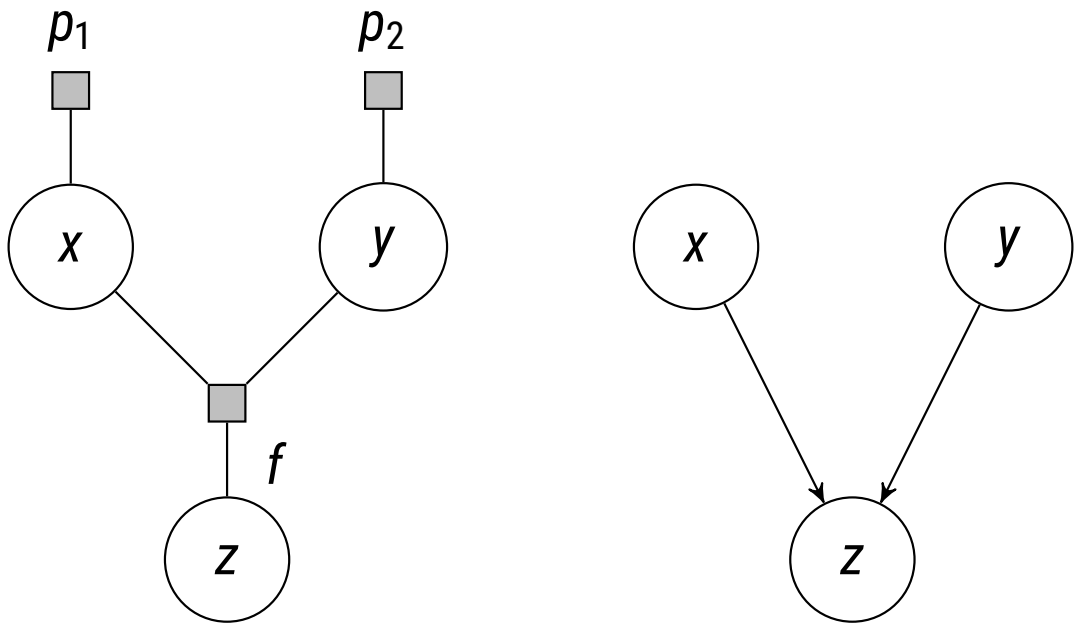
- $p(f) = \mathcal{GP}(f_t; m_\theta, k_\theta), p(D \mid f) = \mathcal{N}(D; f_t(X), \sigma^2), f'(t) = g(f_t, t, a_t)$
- inference: lin. alg., optimization: nonlinear, prediction: model pred. control.

Factor Graphs

a minimal introduction

Frey et al. 1997

A **factor graph** is a graphical representation of a structured equation, in which each *variable* is represented by a circular *node*, each *function* by a square *factor*, and *edges* connect functions to all their variables.



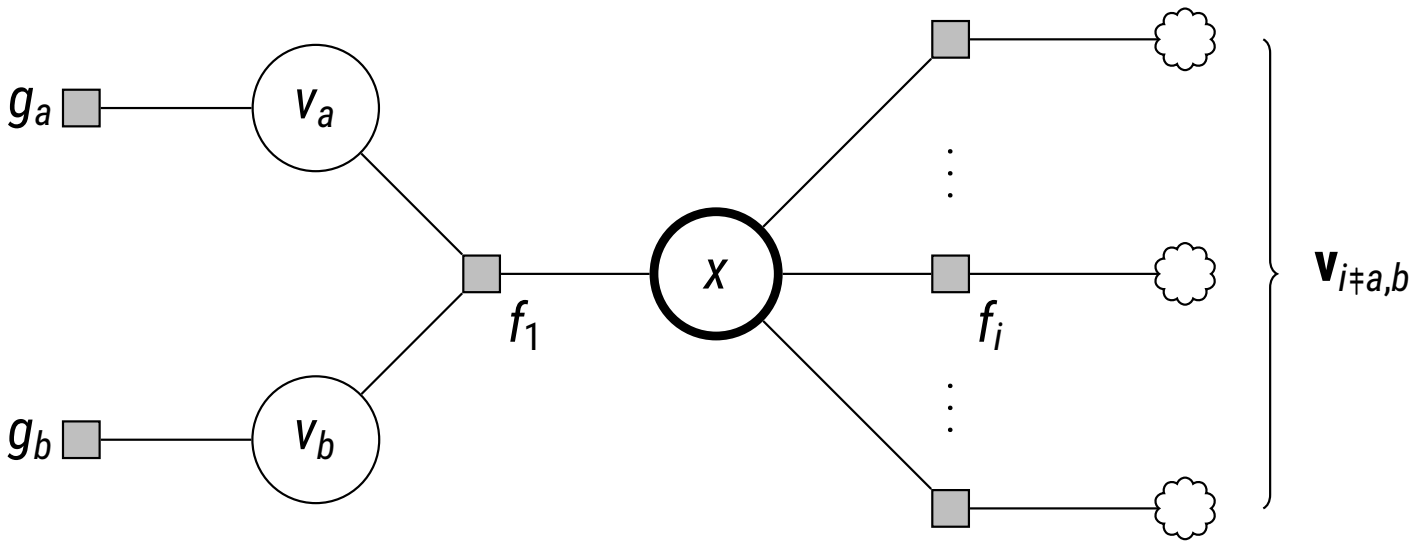
$$p(x, y, z) = p_f(z \mid x, y)p_1(x)p_2(y)$$

Belief Propagation

a concise introduction to the sum-product algorithm

Lauritzen & Spiegelhalter, 1988 Pearl, 1988

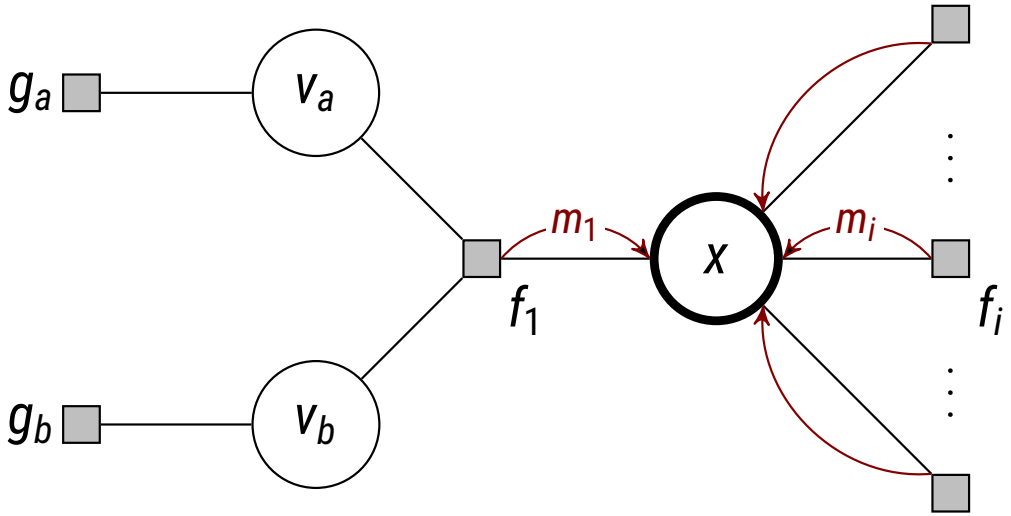
In factor graphs that are **trees**, inference has a mechanical structure.



$$p(x, v_a, v_b, \{\mathbf{v}_{i \neq a,b}\}) = Z^{-1} p_{f_1}(x \mid v_a, v_b) p_{g_a}(v_a) p_{g_b}(v_b) \prod_{i \neq a,b} p_{f_i}(x \mid \mathbf{v}_i) p_{g_i}(\mathbf{v}_i) =: Z^{-1} \tilde{p}(x, \mathbf{v}_i)$$

Belief Propagation

a concise introduction to the sum-product algorithm — Lauritzen & Spiegelhalter, 1988 — Pearl, 1988



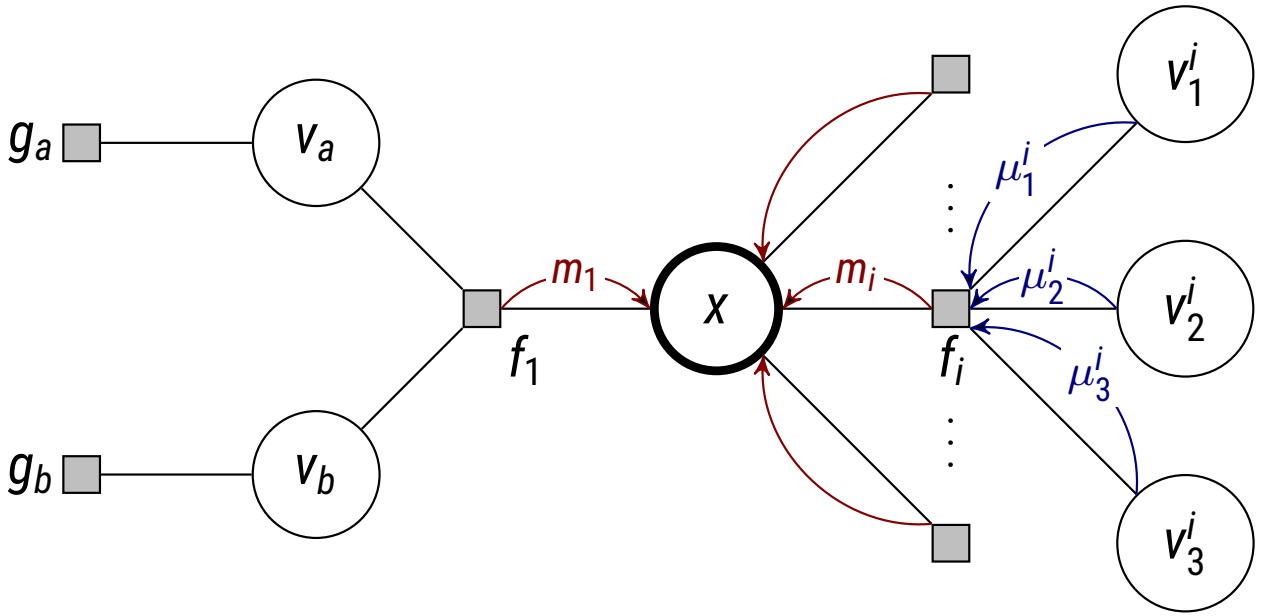
$$\tilde{p}(x) = \int \cdots \int \prod_i p_{f_i}(x \mid \mathbf{v}_i) g_i(\mathbf{v}_i)\, d\mathbf{v}_i$$

$$\overset{\text{due to graph}}{=} \int p_{f_1}(x \mid v_a, v_b) g_a(v_a) g_b(v_b)\, dv_a\, dv_b \prod_i \int p_{f_i}(x \mid \mathbf{v}_i) g_i(\mathbf{v}_i)\, d\mathbf{v}_i$$

$$= \prod_i m_i(x) \qquad \textbf{messages} \text{ passed from } \textit{factors} \text{ to } x$$

Belief Propagation

a concise introduction to the sum-product algorithm — Lauritzen & Spiegelhalter, 1988 — Pearl, 1988



Assume each f_i is connected to K_i variables v_k^i other than x. Because the graph is a tree, v_i^k are themselves connected to separate sub-graphs, containing other variables $\mathbf{w}_k^i$, connected through a (potentially factorizing) $h_k^i(v_k^i, \mathbf{w}_k^i)$.

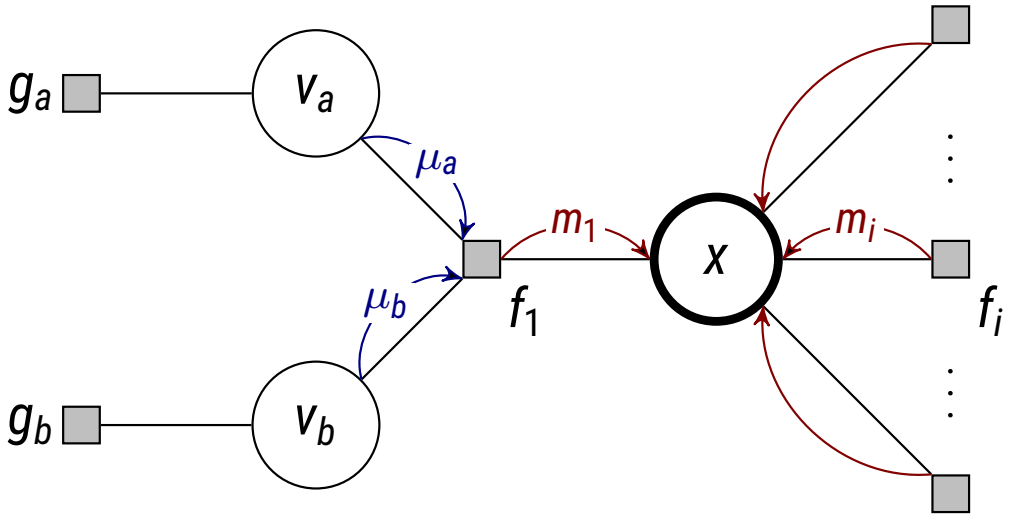
$$m_i(x) = \int dv_1^i \cdots \int dv_{K_i}^i p_{f_i}(x \mid v_1^i, \ldots, v_{K_i}^i) \prod_k^{K_i} \left[\int h_k^i(v_k^i, \mathbf{w}_k^i)\, d\mathbf{w}_k^i \right]$$

$$= \int dv_1^i \cdots \int dv_{K_i}^i p_{f_i}(x \mid v_1^i, \ldots, v_{K_i}^i) \prod_k^{K_i} \mu_k^i(v_k^i)$$ ← **messages** from variables to f_i

Belief Propagation

a concise introduction to the sum-product algorithm

Lauritzen & Spiegelhalter, 1988 Pearl, 1988

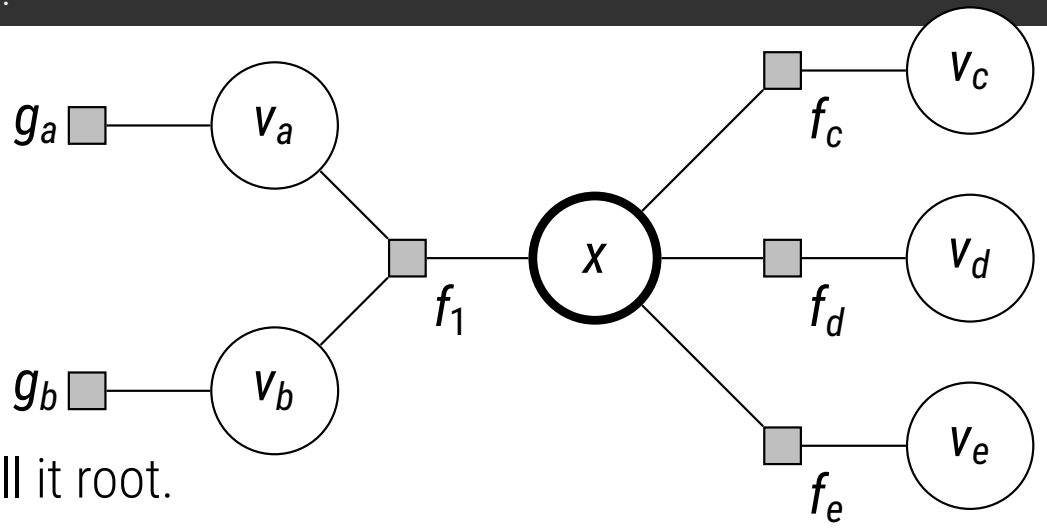


$$m_1(x) = \int dv_a \int dv_b \; p_{f_i}(x \mid v_a, v_b) g_a(v_a) g_b(v_b)$$

$$= \int dv_a \int dv_b \; p(f_i)(x \mid v_a, v_b) \mu_a^1(v_a) \mu_b^1(v_b)$$

The Sum-Product Algorithm

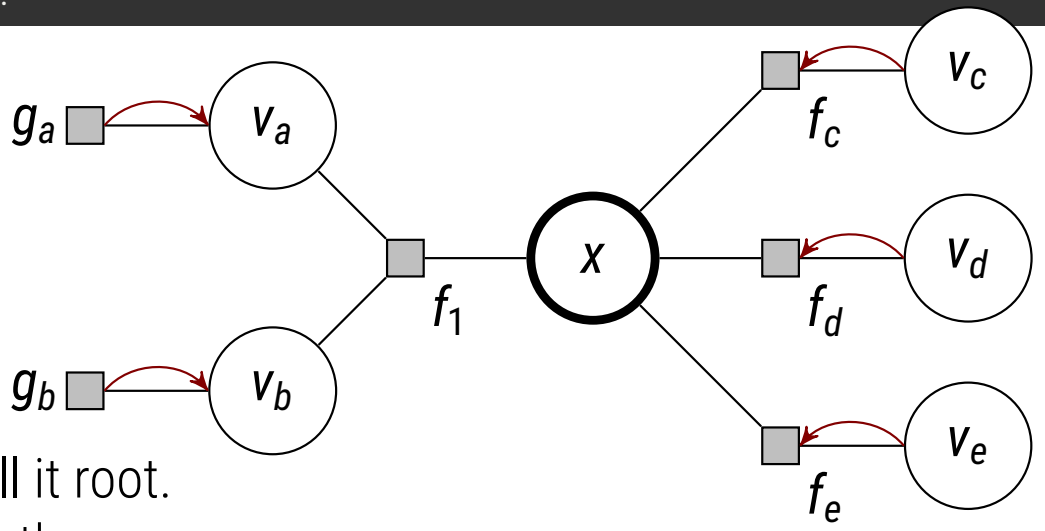
linear-time computation of marginals in trees.



1. pick any variable node, call it root.

The Sum-Product Algorithm

linear-time computation of marginals in trees.

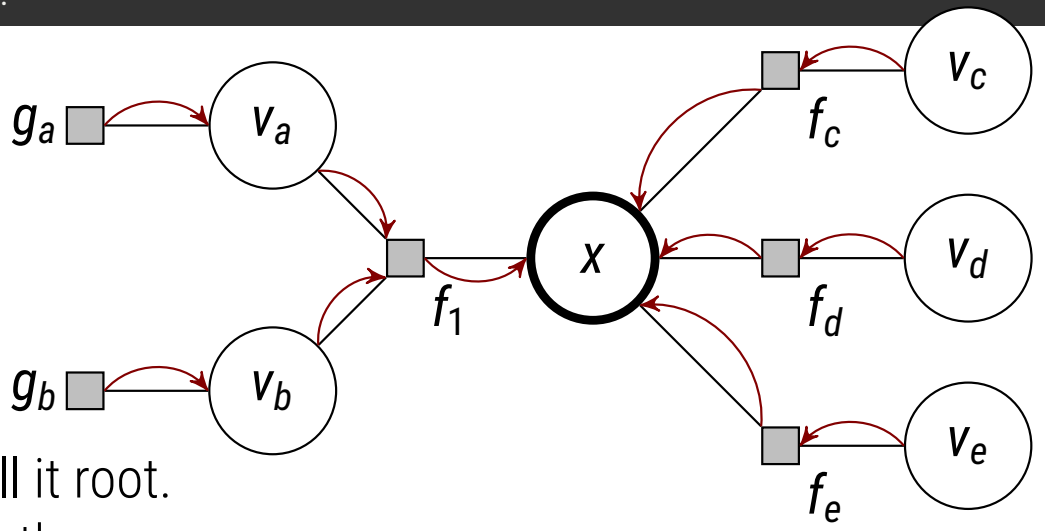


1. pick any variable node, call it root.
2. for all leaf nodes, initialize the messages

$\mu_{v \to f}(v) = 1$ from VAR to FUN, and $m_{f \to v}(v) = p_f(v)$ from FUN to VAR.

The Sum-Product Algorithm

linear-time computation of marginals in trees.



1. pick any variable node, call it root.
2. for all leaf nodes, initialize the messages

   $\mu_{v \to f}(v) = 1$ from VAR to FUN, and $m_{f \to v}(v) = p_f(v)$ from FUN to VAR.

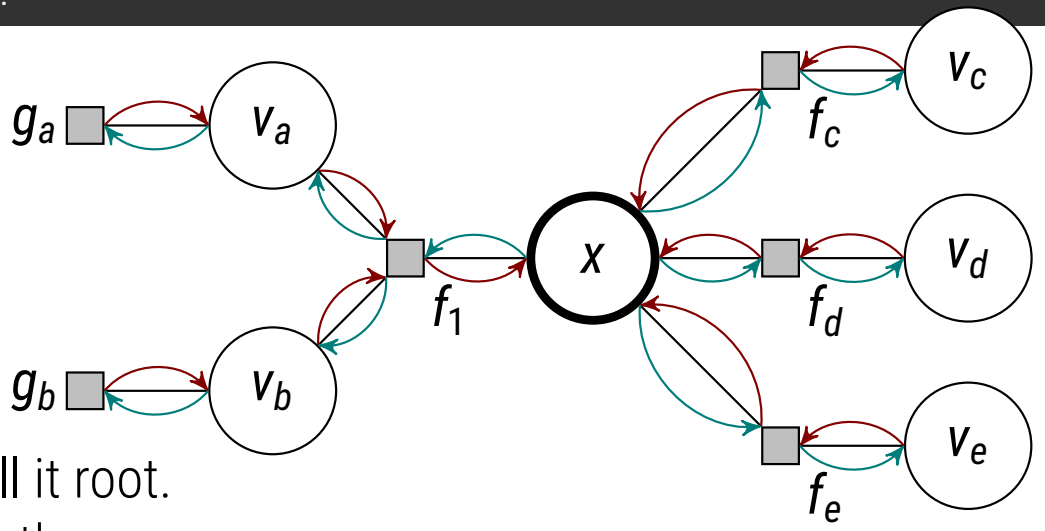
3. pass messages toward root (in N steps), given by

$$\mu_{v_i \to f_k}(v_i) = \prod_{\ell \in \mathrm{ne}(v_i) \setminus f_k} m_{f_\ell \to v_i}(v_i) \quad \text{and}$$

$$m_{f_k \to v_i}(v_i) = \int p_f(v_{h \in \mathrm{ne}(f_k) \setminus v_i}, v_i) \prod_{h \in \mathrm{ne}(f_k) \setminus v_i} \mu_{v_h \to f_k}(v_h)\, dv_h$$

The Sum-Product Algorithm

linear-time computation of marginals in trees.



1. pick any variable node, call it root.
2. for all leaf nodes, initialize the messages

   $\mu_{v \to f}(v) = 1$ from VAR to FUN, and $m_{f \to v}(v) = p_f(v)$ from FUN to VAR.

3. pass messages toward root (in N steps), given by

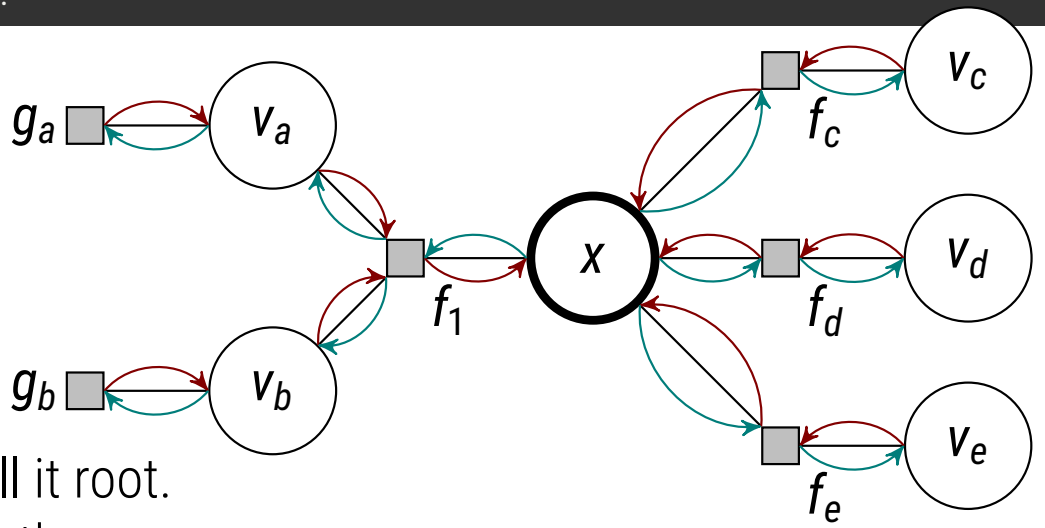
$$\mu_{v_i \to f_k}(v_i) = \prod_{\ell \in \mathrm{ne}(v_i) \setminus f_k} m_{f_\ell \to v_i}(v_i) \quad \text{and}$$

$$m_{f_k \to v_i}(v_i) = \int p_f(v_{h \in \mathrm{ne}(f_k) \setminus v_i}, v_i) \prod_{h \in \mathrm{ne}(f_k) \setminus v_i} \mu_{v_h \to f_k}(v_h)\, dv_h$$

4. pass messages back to leaves (in N steps).

The Sum-Product Algorithm

linear-time computation of marginals in trees.



1. pick any variable node, call it root.
2. for all leaf nodes, initialize the messages

   $\mu_{v \to f}(v) = 1$ from VAR to FUN, and $m_{f \to v}(v) = p_f(v)$ from FUN to VAR.

3. pass messages toward root (in N steps), given by

$$\mu_{v_i \to f_k}(v_i) = \prod_{\ell \in \mathrm{ne}(v_i) \setminus f_k} m_{f_\ell \to v_i}(v_i) \quad \text{and}$$

$$m_{f_k \to v_i}(v_i) = \int p_f(v_{h \in \mathrm{ne}(f_k) \setminus v_i}, v_i) \prod_{h \in \mathrm{ne}(f_k) \setminus v_i} \mu_{v_h \to f_k}(v_h)\, dv_h$$

4. pass messages back to leaves (in N steps).
5. normalize locally: $p(v) = \prod_\ell m_{f_\ell \to v} / \int \prod_\ell m_{f_\ell \to v}\, dv$. Done.

Gaussians, again, make everything is easy

Gaussian belief propagation

If all functional relationships are **linear** (or affine), and all initial messages are **Gaussian**, then all marginals are Gaussian.
Denote the **naturally parametrized Gaussian**:

$$\mathcal{N}(x; \mu, \Sigma) = \mathfrak{N}(x; \mathfrak{m}, \mathfrak{P}) \quad \text{with} \quad \mathfrak{m} = \Sigma^{-1}\mu \quad \text{and} \quad \mathfrak{P} = \Sigma^{-1}.$$

Because products of Gaussians are Gaussians, and convolutions of Gaussians are Gaussians (allowing for $\mathfrak{P} = 0$).

$$\mu_{v_i \to f_k}(v_i) = \prod_{\ell \in \mathrm{ne}(v_i) \setminus f_k} m_{f_\ell \to v_i}(v_i) = \prod_{\ell} \mathfrak{N}(v_i; \mathfrak{m}_\ell, \mathfrak{P}_\ell) = \mathfrak{N}\left(v_i; \sum_\ell \mathfrak{m}_\ell, \sum_\ell \mathfrak{P}_\ell\right)$$

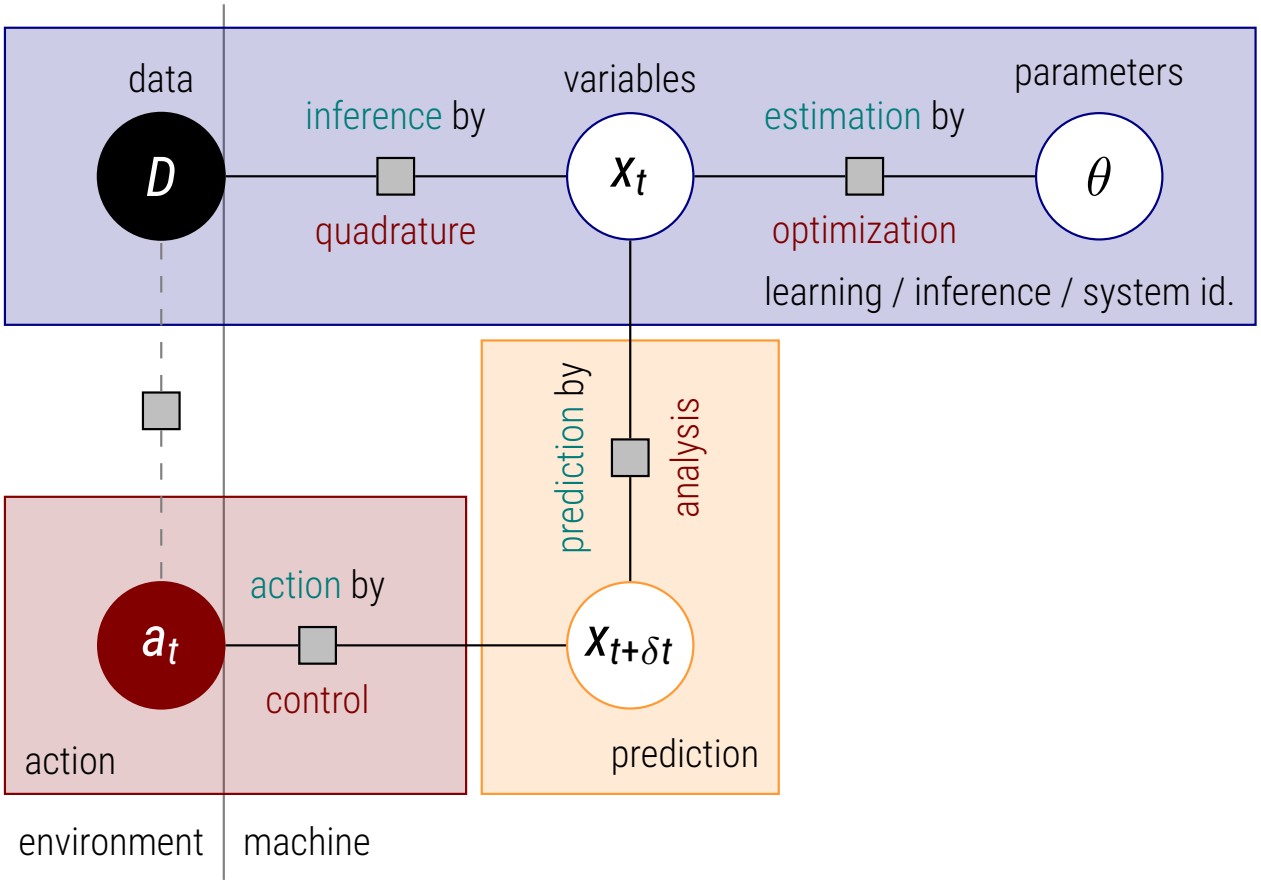
$$m_{f_k \to v_i}(v_i) = \int \delta\left(\mathbf{c} - \begin{bmatrix} F_{ci} & F_{c\mathbf{h}} \end{bmatrix} \begin{bmatrix} v_i \\ \mathbf{v_h} \end{bmatrix}\right) \prod_{h \in \mathrm{ne}(f_k) \setminus v_i} \underbrace{\mu_{v_h \to f_k}(v_h)}_{\mathcal{N}(v_h; \mu_h, \Sigma_h)} dv_h$$

$$= \mathfrak{N}(v_i; F_{ci}^{\mathsf{T}}(c - \bar{\mu}), F_{ci}^{\mathsf{T}} F_{ci} \bar{\Sigma}^{-1}), \text{ with } \bar{\mu} = \sum_h F_{ch}\mu_h \text{ and } \bar{\Sigma} = \sum_h F_{ch}\Sigma_h F_{ch}^{\mathsf{T}}$$

Computation as Message Passing

in graphical models

Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015



- a tree (up to feedback through environment ⇾ reinforcement learning)
- all algorithms discussed so far (can) return and take in Gaussians and have comparable complexity to classic methods

⇒ uncertainty propagation at same complexity as classic point estimates

- see also Chris' lectures for some caveats

A Role for the Evidence

Model Tests in Computation

Estimate z from computations c, under model m.

Prior: structural knowledge reduces complexity

Likelihood: modeling imprecise computation reduces cost

$$p(z \mid c, m) = \frac{p(z \mid m)p(c \mid z, m)}{\int p(z \mid m)p(c \mid z, m)\, dz}$$

Posterior: tracking uncertainty for robustness

Evidence: checking models for safety

cf. Hennig, Osborne, Girolami, Proc. Royal Soc. A, 2015

Evidence Computation in Gaussian Models

a quick refresher

- under jointly Gaussian model ($y \in \mathbb{R}^N$):

$$p(y \mid f, \mathcal{M}) = \mathcal{N}(y; Af, \Lambda) \qquad p(f \mid \mathcal{M}) = \mathcal{GP}(f; m, k)$$

$$\Rightarrow \quad p(y \mid \mathcal{M}) = \int p(y \mid f, \mathcal{M}) p(f \mid \mathcal{M})\, df = \mathcal{N}(y; Am, AkA^\intercal + \Lambda)$$

$$\log p(y \mid \mathcal{M}) = -\frac{1}{2}(y - Am)^\intercal (AkA^\intercal + \Lambda)^{-1}(y - Am) - \frac{1}{2}\log |AkA^\intercal + \Lambda| - \frac{N}{2}\log 2\pi$$

- Under $\mathcal{M}$, the expected log evidence is simply

$$\mathrm{E}_{\tilde{y}|\mathcal{M}}[\log p(\tilde{y} \mid \mathcal{M})] = -\frac{1}{2}\operatorname{tr}(I) - \frac{1}{2}\log |AkA^\intercal + \Lambda| - \frac{N}{2}\log 2\pi$$

- hence an interesting (well-scaled) quantity is

$$2(\log p(y \mid \mathcal{M}) - \mathrm{E}_{\tilde{y}|\mathcal{M}}[\log p(\tilde{y} \mid \mathcal{M})]) = (y - Am)^\intercal (AkA^\intercal + \Lambda)^{-1}(y - Am) - N$$

Computational Consideration

Computing the Evidence adds only insignificant overhead to Gaussian computations — Särkkä, 2013

For Gauss-Markov models with

$$p(x_{1:T}) = \prod_{i=1}^{T-1} p(x_{i+1} \mid x_i) \qquad \text{and} \qquad p(y_{1:T} \mid x_{1:T}) = \prod_{i=1}^{T} p(y_i \mid x_i)$$

we have the **prediction-error decomposition**

$$p(y_{1:T} \mid \mathcal{M}) = \prod_{i=1}^{T} p(y_i \mid y_{1:i-1}, \mathcal{M}) = \prod_{i=1}^{T} \int p(y_i \mid x_i) p(x_i \mid y_{1:i-1}, \mathcal{M})\, dx_i$$

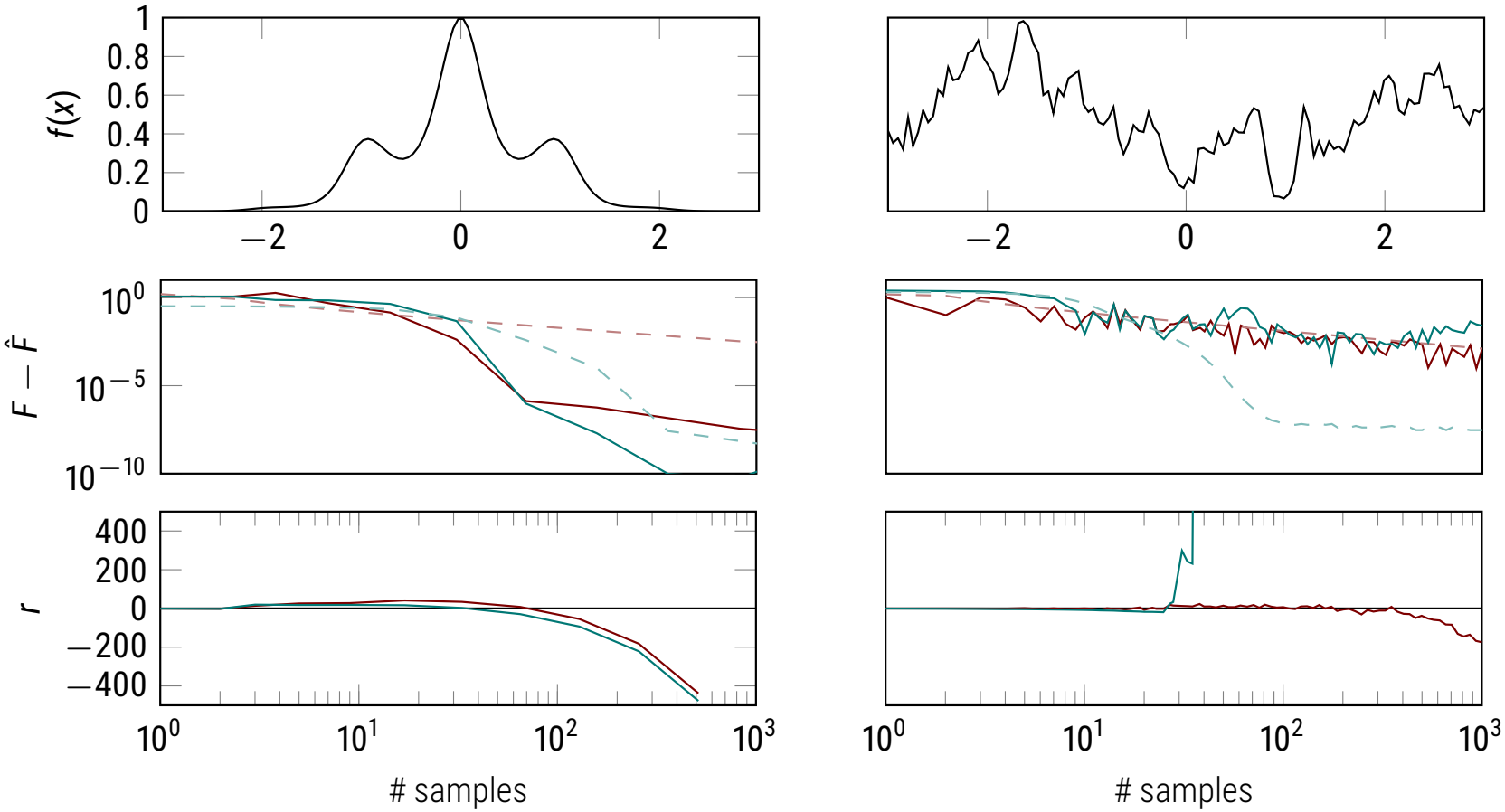
For Gaussian SDE models with quantities as defined in previous lectures:

$$p(y_{1:T} \mid \mathcal{M}) = \prod_{i=1}^{T} \mathcal{N}(y_i; Hm_i^-, \underbrace{HP_i^- H^\mathsf{T} + R_i}_{=:S_i}) \quad \text{i.e. (if } y_i \in \mathbb{R}^M\text{)}$$

$$\log p(y_{1:T} \mid \mathcal{M}) = -\frac{1}{2}\left(\sum_{i=1}^{T} (y_i - Hm_i^-)^\mathsf{T} S_i^{-1} (y_i - Hm_i^-) + \log |S_i| + M \log 2\pi\right)$$

A Simple Example

proof of concept: [Hennig, Osborne, Girolami. Proc. Royal Society A, 2015]



$$r = \mathrm{E}_{\tilde{f}}\left[\log \frac{p(\tilde{f}(\mathbf{x}))}{p(f(\mathbf{x}))}\right] = (f(\mathbf{x}) - \mu(\mathbf{x}))^{\mathsf{T}} K^{-1}(f(\mathbf{x}) - \mu(\mathbf{x})) - N$$

Summing Up

Linking Probabilistic and Classic Numerics

Many classic numerical methods are MAP estimates under specific priors and Dirac likelihoods.

- Gaussian **quadrature** rules: Gaussian process priors
- initial value problems for **ODEs**: Gaussian Filters
  - Runge-Kutta single-step as initial step in uninformative limit
  - Multistep (Nordsieck form) in steady-state limit
  - explicit tracking of various forms of uncertainty
- **linear algebra**: projection, conjugate direction, conjugate gradient methods as nested families of Gaussian priors. Action policy of algorithm intrinsically motivated by model. Extension to noisy observations or structured prior can be challenging (though not impossible)
- **quasi-Newton** methods as **non-linear optimization** extensions of Gaussian solvers (to filters with nontrivial diffusion)

Classic methods can serve as a **starting point** for probabilistic numerics.
Even just re-discovering classic methods is valuable, as it allows properly tracking (Gaussian) uncertainty.

New Probabilistic Functionality

Taking classic methods as starting points, Bayes' theorem provides novel functionality for the challenges of contemporary computer science.

- computation with **big datasets** involves nontrivial cost/precision tradeoff, which can be captured by non-trivial likelihoods
- **free algorithmic parameters** (step-sizes, batch-sizes, termination conditions) can become identified when **additional statistics** are computed alongside the classic quantities. In fact those quantities can even lead to improved overall performance.
- **belief propagation** provides principled framework for the **propagation of uncertainty** through computational pipelines
- **statistical tests** involving the evidence allow novel tests for **reliability, safety** of computations

Probabilistic Numerics
Uncertainty in Computation

- **Computation is Inference**
- probability measures, as **inputs and outputs**
- many classics are MAP estimates, under specific priors and likelihoods
- extending from this foundation allows novel functionality:

prior: structural knowledge reduces complexity
likelihood: imprecise computation lowers cost
posterior: uncertainty propagated through computations
evidence: model mismatch detectable at run-time

http://probnum.org
https://pn.is.tue.mpg.de
https://github.com/ProbabilisticNumerics
http://tinyurl.com/Dobbiaco-Hennig-6

Probabilistic Numerics — Uncertainty in Computation
Hennig, Osborne, Girolami. Cambridge University Press, expected 2018

Some Open Questions

The complexity of uncertainty

Complexity — even Gaussian models have structural limitations

- adding noise to linear algebra?
- tracking incomplete identified sub-spaces in quasi-Newton?
- convergence under heteroscedastic, controlled noise?

Limits of Calibration — what lies beyond the variance?

- bifurcation, chaos, in differential equations?
- scales of integrals?
- identifiability of priors?

More (including some answers!) in Chris' lectures!