

Probabilistic Programming with Gaussian Process Memoization

Ulrich Schaechtle

Department of Computer Science

Royal Holloway, University of London

ULRICH.SCHAECHTLE@RHUL.AC.UK

Ben Zinberg

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

BZINBERG@MIT.EDU

Alexey Radul

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

AXOFCH@GMAIL.COM

Kostas Stathis

Department of Computer Science

Royal Holloway, University of London

KOSTAS.STATHIS@RHUL.AC.UK

Vikash K. Mansinghka

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

VKM@MIT.EDU

Editor: N.A.

Abstract

1 This paper describes the *Gaussian process memoizer*, a probabilistic programming technique
 2 that uses Gaussian processes to provide a statistical alternative to memorization.
 3 Memoizing a target procedure results in a self-caching wrapper that remembers previously
 4 computed values. Gaussian process memoization additionally produces a statistical em-
 5 ulator based on a Gaussian process whose predictions automatically improve whenever
 6 a new value of the target procedure becomes available. This paper also introduces an
 7 efficient implementation, named `gpmem`, that can use kernels given by a broad class of prob-
 8 abilistic programs. The flexibility of `gpmem` is illustrated via three applications: (i) GP
 9 regression with hierarchical hyper-parameter learning, (ii) Bayesian structure learning via
 10 compositional kernels generated by a probabilistic grammar, and (iii) a bandit formulation
 11 of Bayesian optimization with automatic inference and action selection. All applications
 12 share a single 50-line Python library and require fewer than 20 lines of probabilistic code
 13 each.

¹⁴ 1. Introduction

15 Gaussian Processes (GP) are widely used tools in statistics (Barry, 1986), machine learn-
 16 ing (Neal, 1995; Williams and Barber, 1998; Kuss and Rasmussen, 2005; Rasmussen and
 17 Williams, 2006; Damianou and Lawrence, 2013), robotics (Ferris et al., 2006), computer
 18 vision (Kemmler et al., 2013), and scientific computation (Kennedy and O’Hagan, 2001;
 19 Schneider et al., 2008; Kwan et al., 2013). They are also central to probabilistic numerics,

20 an emerging effort to develop more computationally efficient numerical procedures, and to
21 Bayesian optimization, a family of meta-optimization techniques that are widely used to
22 tune parameters for deep learning algorithms (Snoek et al., 2012; Gelbart et al., 2014).
23 They have even seen use in artificial intelligence; for example, they provide the key tech-
24 nology behind a project that produces qualitative natural language descriptions of time
25 series (Duvenaud et al., 2013; Lloyd et al., 2014).

26 This paper describes Gaussian process memoization, a technique for integrating GPs into
27 a probabilistic programming language, and demonstrates its utility by re-implementing and
28 extending state-of-the-art applications of the GP. Memoization is a classic programming
29 technique in which a procedure is augmented with an input-output cache that is checked
30 each time before the function is invoked. This prevents unnecessary recomputation, poten-
31 tially saving time at the cost of increased storage requirements. Gaussian process memo-
32 ization, implemented by the `gpmem()` procedure, generalizes this idea to include a statistical
33 emulator that uses previously computed values as data in a statistical model that can accu-
34 rately forecast probable outputs. The covariance function for the Gaussian process is also
35 allowed to be an arbitrary probabilistic program.

36 This paper presents three applications of gpmem: (i) a replication of results Neal (1997)
37 on outlier rejection via hyper-parameter inference; (ii) a fully Bayesian extension to the
38 Automated Statistician project; and (iii) an implementation of Bayesian optimization via
39 Thompson sampling. The first application can in principle be replicated in several other
40 probabilistic languages embedding the proposal that is described in this paper. The re-
41 maining two applications rely on distinctive capabilities of Venture (Mansinghka et al.,
42 2014): support for fully Bayesian structure learning and language constructs for inference
43 programming. All applications share a single 50-line Python library and require fewer than
44 20 lines of probabilistic code each.

45 2. Background on Gaussian Processes

46 GPs are a Bayesian method for regression. We consider the regression input to be real-
47 valued scalars x_i and the regression output $f(x_i) = y_i$ as the value of a function f at x_i .
48 The complete training data will be denoted by column vectors \mathbf{x} and \mathbf{y} . Unseen test input
49 is denoted with \mathbf{x}' . GPs present a non-parametric way to express prior knowledge on the
50 space of all possible functions f modeling a regression relationship. Formally, a GP is an
51 infinite-dimensional extension of the multivariate Gaussian distribution.

52 The collection of random variables $\{f(x_i) = y_i\}$ (indexed by i) represents the values of
53 the function f at each location x_i . We write $f \sim \mathcal{GP}(\mu, k \mid \boldsymbol{\theta}_{\text{mean}}, \boldsymbol{\theta})$, where μ is the *mean*
54 *function* and k is the *covariance function* or *kernel*. That is, $\mu(x_i \mid \boldsymbol{\theta}_{\text{mean}})$ is the prior
55 mean of the random variable y_i , and $k(x_i, x_j \mid \boldsymbol{\theta})$ is the prior covariance of the random
56 variables y_i and y_j . The output of both mean and covariance function are conditioned on
57 a few free hyper-parameters parameterizing k and μ . We refer to these hyper-parameters
58 as $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_{\text{mean}}$ respectively. To simplify the calculation below, we will assume the prior
59 mean μ is identically zero; once the derivation is done, this assumption can be easily relaxed
60 via translation. We use upper case italic $K(\mathbf{x}, \mathbf{x}' \mid \boldsymbol{\theta})$ for a function that returns a matrix
61 of dimension $I \times J$ with entries $k(x_i, x_j \mid \boldsymbol{\theta})$ and with $x_i \in \mathbf{x}$ and $x_j \in \mathbf{x}'$ where I and
62 J indicate the length of the column vectors \mathbf{x} and \mathbf{x}' . Throughout, we write $\mathbf{K}_{(\boldsymbol{\theta}, \mathbf{x}, \mathbf{x}')}$ for

63 the prior covariance matrix that results from computing $K(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta})$. In the following, we
 64 will sometimes drop the subscript \mathbf{x}, \mathbf{x}' , writing only $\mathbf{K}_{\boldsymbol{\theta}}$, for clarity. Note that we do this
 65 only in cases when both input vectors are identical and correspond to training input \mathbf{x} .
 66 We differentiate two different situations leading to different ways samples can be generated
 67 with this setup:

- 68 1. \mathbf{y}' - the predictive posterior sample from a distribution conditioned on observed input
 69 \mathbf{x} with observed output \mathbf{y} and conditioned on $\boldsymbol{\theta}$.
- 70 2. \mathbf{y}^* - a sample from the predictive prior. We will describe situations, where the GP
 71 has not seen any data \mathbf{x}, \mathbf{y} yet. In this case, we sample from a Gaussian distribution
 72 with $\mathbf{y}^* \sim \mathcal{N}(0, K(\mathbf{x}^*, \mathbf{x}^* | \boldsymbol{\theta}))$; where the symbol * indicates that no data has been
 73 observed yet.

74 We now show how to compute the predictive posterior distribution of test output
 75 $\mathbf{y}' := f(\mathbf{x}')$ conditioned on training data $\mathbf{y} := f(\mathbf{x})$. (Here \mathbf{x} and \mathbf{x}' are known constant
 76 vectors, and we are conditioning on an observed value of \mathbf{y} .) The predictive posterior can
 77 be computed by first forming the joint density when both training and test data are treated
 78 as randomly chosen from the prior, then fixing the value of \mathbf{y} to a constant. To start, let

$$\Sigma := \begin{bmatrix} K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta}) & K(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}) \\ K(\mathbf{x}', \mathbf{x} | \boldsymbol{\theta}) & K(\mathbf{x}', \mathbf{x}' | \boldsymbol{\theta}) \end{bmatrix} \text{ and } \Sigma^{-1} =: \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{bmatrix}.$$

79 We then have

$$P(\mathbf{y}, \mathbf{y}' | \boldsymbol{\theta}) \propto \exp \left\{ -\frac{1}{2} [\mathbf{y}^\top \quad \mathbf{y}'^\top] \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} \\ \mathbf{M}_{21} & \mathbf{M}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{y}' \end{bmatrix} \right\}.$$

80 Treating \mathbf{y} as a fixed constant, we obtain

$$P(\mathbf{y}' | \mathbf{y}, \boldsymbol{\theta}) \propto P(\mathbf{y}, \mathbf{y}' | \boldsymbol{\theta}) \propto \exp \left\{ -\frac{1}{2} \mathbf{y}'^\top \mathbf{M}_{22} \mathbf{y}' - \mathbf{h}^\top \mathbf{y}' \right\},$$

81 where $\mathbf{h} = \mathbf{M}_{21}\mathbf{y}$ is a constant vector. Thus $P(\mathbf{y}' | \mathbf{y}, \boldsymbol{\theta})$ is Gaussian,

$$P(\mathbf{y}' | \mathbf{y}, \boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}, \mathbf{K}_{\boldsymbol{\theta}}^{\text{post}}), \quad (1)$$

with covariance matrix $\mathbf{K}_{\boldsymbol{\theta}}^{\text{post}} = \mathbf{M}_{22}^{-1}$. To find its mean $\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}$, we note that $P_{\mathbf{y}'|\mathbf{y},\boldsymbol{\theta}}(\mathbf{y}' + \boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}})$ is Gaussian with the same covariance as $P(\mathbf{y}' | \mathbf{y}, \boldsymbol{\theta})$, but its exponent has no linear term:

$$\begin{aligned} P_{\mathbf{y}'|\mathbf{y},\boldsymbol{\theta}}(\mathbf{y}' + \boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}} | \mathbf{y}, \boldsymbol{\theta}) &\propto \exp \left\{ -\frac{1}{2} (\mathbf{y}' + \boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}})^\top \mathbf{M}_{22} (\mathbf{y}' + \boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}) - \mathbf{h}^\top (\mathbf{y}' + \boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}) \right\} \\ &\propto \exp \left\{ -\frac{1}{2} \mathbf{y}'^\top \mathbf{M}_{22} \mathbf{y}' - \underbrace{(\mathbf{h} + \mathbf{M}_{22}\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}})^\top \mathbf{y}'}_{\text{must be 0}} \right\}. \end{aligned}$$

82 Thus $\mathbf{h} = -\mathbf{M}_{22}\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}$ and $\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}} = -\mathbf{M}_{22}^{-1}\mathbf{h} = -\mathbf{M}_{22}^{-1}\mathbf{M}_{21}\mathbf{y}$.

The partitioned inverse equations (Barnett and Barnett, 1979 following MacKay, 1998) give

$$\mathbf{M}_{22} = (K(\mathbf{x}', \mathbf{x}' | \boldsymbol{\theta}) - K(\mathbf{x}', \mathbf{x} | \boldsymbol{\theta})K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta})^{-1}K(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}))^{-1},$$

$$\mathbf{M}_{21} = -\mathbf{M}_{22}K(\mathbf{x}', \mathbf{x} | \boldsymbol{\theta})K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta})^{-1}.$$

Substituting these in the above gives

$$\mathbf{K}_{\boldsymbol{\theta}}^{\text{post}} = K(\mathbf{x}', \mathbf{x}' | \boldsymbol{\theta}) - K(\mathbf{x}', \mathbf{x} | \boldsymbol{\theta})K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta})^{-1}K(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}), \quad (2)$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}} = K(\mathbf{x}', \mathbf{x} | \boldsymbol{\theta})K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta})^{-1}\mathbf{y}. \quad (3)$$

⁸³ Together, $\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}$ and $\mathbf{K}_{\boldsymbol{\theta}}^{\text{post}}$ determine the computation of the predictive posterior with unseen
⁸⁴ input data (1).

⁸⁵ Often one assumes the observed regression output is noisily measured, that is, one only
⁸⁶ sees the values of $\mathbf{y}_{\text{noisy}} = \mathbf{y} + \mathbf{w}$ where \mathbf{w} is Gaussian white noise with variance σ_{noise}^2 . This
⁸⁷ noise term can be absorbed into the covariance function $K(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta})$. The log-likelihood of
⁸⁸ a GP can then be written as:

$$\log P(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top \mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\log |\mathbf{K}_{\boldsymbol{\theta}}| - \frac{n}{2}\log 2\pi \quad (4)$$

⁸⁹ where n is the number of data points. Both log-likelihood and predictive posterior can be
⁹⁰ computed efficiently using a Stochastic Process (SP) in Venture (Mansinghka et al., 2014)
⁹¹ with an algorithm that resorts to Cholesky factorization(Rasmussen and Williams, 2006,
⁹² chap. 2). We write the Cholesky factorization as $\mathbf{L} := \text{chol}(\mathbf{K}_{\boldsymbol{\theta}})$ when :

$$\mathbf{K}_{\boldsymbol{\theta}} = \mathbf{L}\mathbf{L}^\top \quad (5)$$

⁹³ where \mathbf{L} is a lower triangular matrix. This allows us to compute the inverse of a covariance
⁹⁴ matrix as

$$\mathbf{K}_{\boldsymbol{\theta}}^{-1} = (\mathbf{L}^{-1})^\top (\mathbf{L}^{-1}) \quad (6)$$

⁹⁵ and its determinant as

$$\det(\mathbf{K}_{\boldsymbol{\theta}}) = \det(\mathbf{L})^2 \quad (7)$$

⁹⁶ We compute (4) as

$$\log(P(\mathbf{y} | \mathbf{x}, \boldsymbol{\theta})) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log \mathbf{L}_{ii} - \frac{n}{2}\log 2\pi \quad (8)$$

⁹⁷ where

$$\mathbf{L} := \text{chol}(\mathbf{K}_{\boldsymbol{\theta}}) \quad (9)$$

⁹⁸ and

$$\boldsymbol{\alpha} := \mathbf{L}^\top \setminus (\mathbf{L} \setminus \mathbf{y}). \quad (10)$$

⁹⁹ This results in a computational complexity for sampling in the number of data points of
¹⁰⁰ $O(n^3/6)$ for (9) an $O(n^2/2)$ for (10).

¹⁰¹ Above, we defined the GP prior as $\mathbf{y}^* \sim \mathcal{N}(0, K(\mathbf{x}_*, \mathbf{x}_* | \boldsymbol{\theta}))$. We see that this prior is
¹⁰² fully determined by its covariance function.

¹⁰³ **2.1 Covariance Functions**

¹⁰⁴ The covariance function (or kernel) of a GP governs high-level properties of the observed
¹⁰⁵ data such as smoothness or linearity. The high-level properties are indicated with super-
¹⁰⁶ script on functions. A linear covariance can be written as:

$$k^{\text{linear}} = \sigma_1^2(xx'). \quad (11)$$

¹⁰⁷ We can also express periodicity:

$$k^{\text{periodic}} = \sigma_2^2 \exp\left(\frac{2 \sin^2(\pi(x - x')/p)}{\ell^2}\right). \quad (12)$$

¹⁰⁸ By changing these properties we get completely different prior behavior for sampling \mathbf{y}^*
¹⁰⁹ from a GP with a linear kernel

$$\mathbf{y}^* \sim \mathcal{N}(0, K^{\text{linear}}(\mathbf{x}^*, \mathbf{x}^* | \sigma_1))$$

¹¹⁰ as compared to sampling from the prior predictive with a periodic kernel (as depicted in
¹¹¹ Fig. 1 (c) and (d))

$$\mathbf{y}^* \sim \mathcal{N}(0, K^{\text{periodic}}(\mathbf{x}^*, \mathbf{x}^* | \sigma_2, \ell)).$$

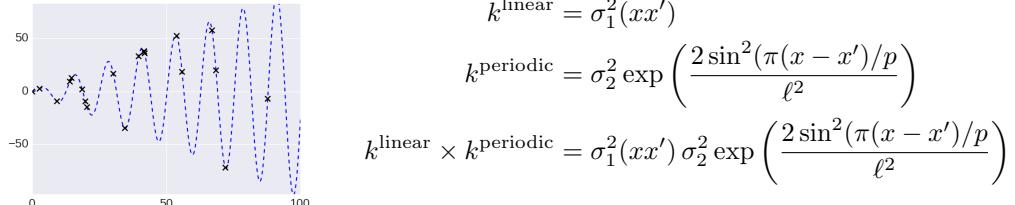
¹¹² These high-level properties are compositional via addition and multiplication of different
¹¹³ covariance functions. That means that we can also combine these properties. By using
¹¹⁴ multiplication of kernels we can model a local interaction of two components, for example

$$k^{\text{linear}} \times k^{\text{periodic}} = \sigma_1^2(xx') \sigma_2^2 \exp\left(\frac{2 \sin^2(\pi(x - x')/p)}{\ell^2}\right) \quad (13)$$

¹¹⁵ This results in a combination of the higher level properties of linearity and periodicity. In
¹¹⁶ Fig 1 (e) we depict samples for \mathbf{y}^* that are periodic with linearly increasing amplitude. We
¹¹⁷ consider this a local interaction because the actual interaction depends on the similarity of
¹¹⁸ two data points. An addition of covariance functions models a global interaction, that is an
¹¹⁹ interaction of two high-level components that is qualitatively not dependent on the input
¹²⁰ space. An example for this a periodic function with a linear trend.

¹²¹ For each kernel type, each $\boldsymbol{\theta}$ is different, that is, in (11) we have $\boldsymbol{\theta} = \{\sigma_1\}$, in (12) we
¹²² have $\boldsymbol{\theta} = \{\sigma_2, p, \ell\}$ and in (13) we have $\boldsymbol{\theta} = \{\sigma_1, \sigma_2, p, \ell\}$. Adjusting these hyper-parameters
¹²³ changes lower level qualitative attributes such as length scales (ℓ) while preserving the higher
¹²⁴ level qualitative properties of the distribution such as linearity.

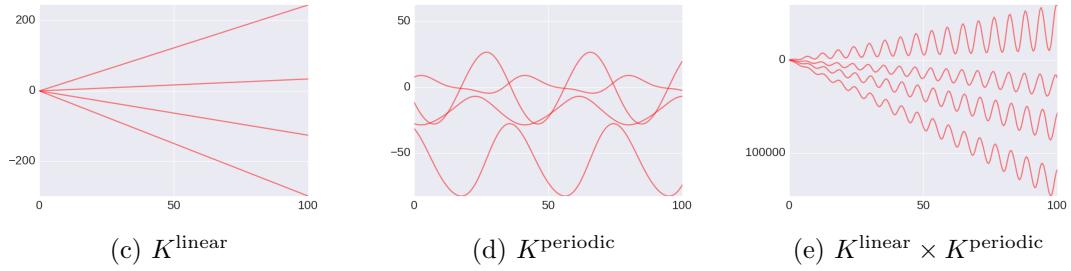
¹²⁵ If we choose suitable hyper-parameters, for example by performing inference, we can
¹²⁶ capture the underlying dynamics of the data well (see Fig. 1 (f-h)) while sampling \mathbf{y}' .
¹²⁷ Note that goodness of fit is not only limited to the parameters. A too simple qualitative
¹²⁸ structure implies unsuitable behaviour, as for example in (Fig. 1 (g)) where additional
¹²⁹ recurring spikes are introduced to account for the changing amplitude of the true function
¹³⁰ that generated the data.



(a) Raw Data

(b) Kernels

Prior predictive, $\mathbf{y}_* \sim \mathcal{N}(\mathbf{0}, K(\mathbf{x}_*, \mathbf{x}_* | \boldsymbol{\theta}))$:



Posterior predictive $\mathbf{y}' \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}, \mathbf{K}_{\boldsymbol{\theta}}^{\text{post}})$:

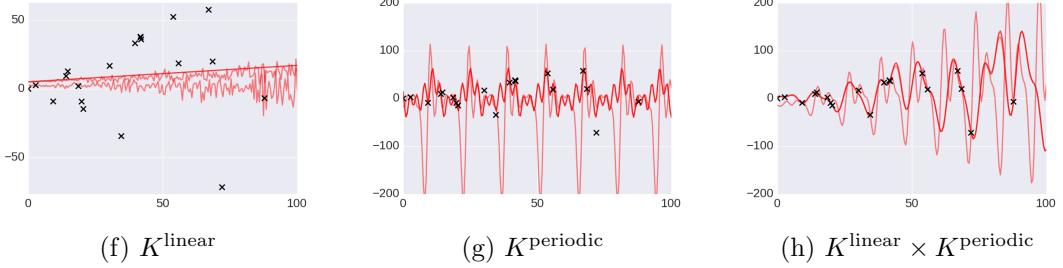


Figure 1: We depict kernel composition. (a) shows raw data (black) generated with a sine function with linearly growing amplitude (blue). This data is used for all the plots (c-h). (b) shows the linear and the periodic base kernel in functional form as well as a composition of both. The multiplication of the two kernels indicates local interaction. The local interaction we account for in this case is the growing amplitude (a). For each column (c-h) $\boldsymbol{\theta}$ is different. (c-e) show samples from the prior prior predictive \mathbf{y}^* where random parameters are used, that is, we sample before any data points are observed. (f-h) show samples from the predictive posterior \mathbf{y}' , after the data has been observed.

131 **3. Gaussian Process Memoization in Venture**

132 Memoization is the practice of storing previously computed values of a function so that
133 future calls with the same inputs can be evaluated by lookup rather than re-computation.
134 To transfer this idea to probabilistic programming, we now introduce a language construct
135 called a *statistical memoizer*. Suppose we have a function f which can be evaluated but we
136 wish to learn about the behavior of f using as few evaluations as possible. The statistical
137 memoizer, which here we give the name `gpmem`, was motivated by this purpose. It produces
138 two outputs:

$$f \xrightarrow{\text{gpmem}} (f_{\text{compute}}, f_{\text{emu}}).$$

139 The function f_{compute} calls f and stores the output in a memo table, just as traditional
140 memoization does. The function f_{emu} is an online statistical emulator which uses the memo
141 table as its training data. A fully Bayesian emulator, modelling the true function f as a
142 random function $f \sim P(f)$, would satisfy

$$(f_{\text{emu}}(x_1 \dots x_k) \sim P(f(x_1), \dots, f(x_k) \mid f(x) = (f \ x) \text{ for each } x \text{ in memo table}).$$

143 Different implementations of the statistical memoizer can have different prior distribu-
144 tions $P(f)$; in this paper, we deploy a GP prior (implemented as `gpmem` below). Note
145 that we require the ability to sample f_{emu} jointly at multiple inputs because the values of
146 $f(x_1), \dots, f(x_k)$ will in general be dependent.

147 We explain how `gpmem`, the statistical memoizer with GP-prior, works using a simple
148 tutorial (Fig. 2). The top panel (Fig. 2, (a)) of this figure sketches the schematic of
149 `gpmem`. f is the external process that we memoize. It can be evaluated using resources that
150 potentially come from outside of Venture. We feed this function into `gpmem` alongside a
151 parameterised kernel k . In this example, we make the qualitative assumption of f being
152 smooth, and define k to be a squared-exponential covariance function:

$$k = k^{\text{se}} = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right).$$

153 The hyper-parameters θ for this kernel are sampled from a prior distribution which is
154 depicted in the top right box. Note that we annotate $\theta = \{\text{sf}, 1\}$ for subsequent inference
155 as belonging to (i) the scope "hyper-parameter" and (ii) blocks 0 and 1 respectively.

156 `gpmem` implements a memoization table, where all previously computed function evalua-
157 tions ($\{\mathbf{x}, \mathbf{y}\}$) are stored. We also initialize a GP-prior that will serve as our statistical
158 emulator:

$$P(f_{\text{emu}}(x) \mid \mathbf{x}, \mathbf{y}, \theta) \sim \mathcal{N}(\mu_{\theta}^{\text{post}}, \mathbf{K}_{\theta}^{\text{post}}))$$

159 where

$$P(f_{\text{emu}}(x) \mid \mathbf{x}, \mathbf{y}, \theta) = \mathbf{y}'$$

160 under the traditional GP perspective. All value pairs stored in the memoization table
161 (memo table = (\mathbf{x}, \mathbf{y})) are incorporated as observations of the GP. We simply feed the
162 regression input into the emulator and output a predictive posterior Gaussian distribution
163 determined by the GP and the memoization table.

164 We can either define the function f that serves as as input for `gpmem` natively in Venture
165 (as shown in the Fig. 2 (b)) or we interleave Venture with foreign code. This can be useful

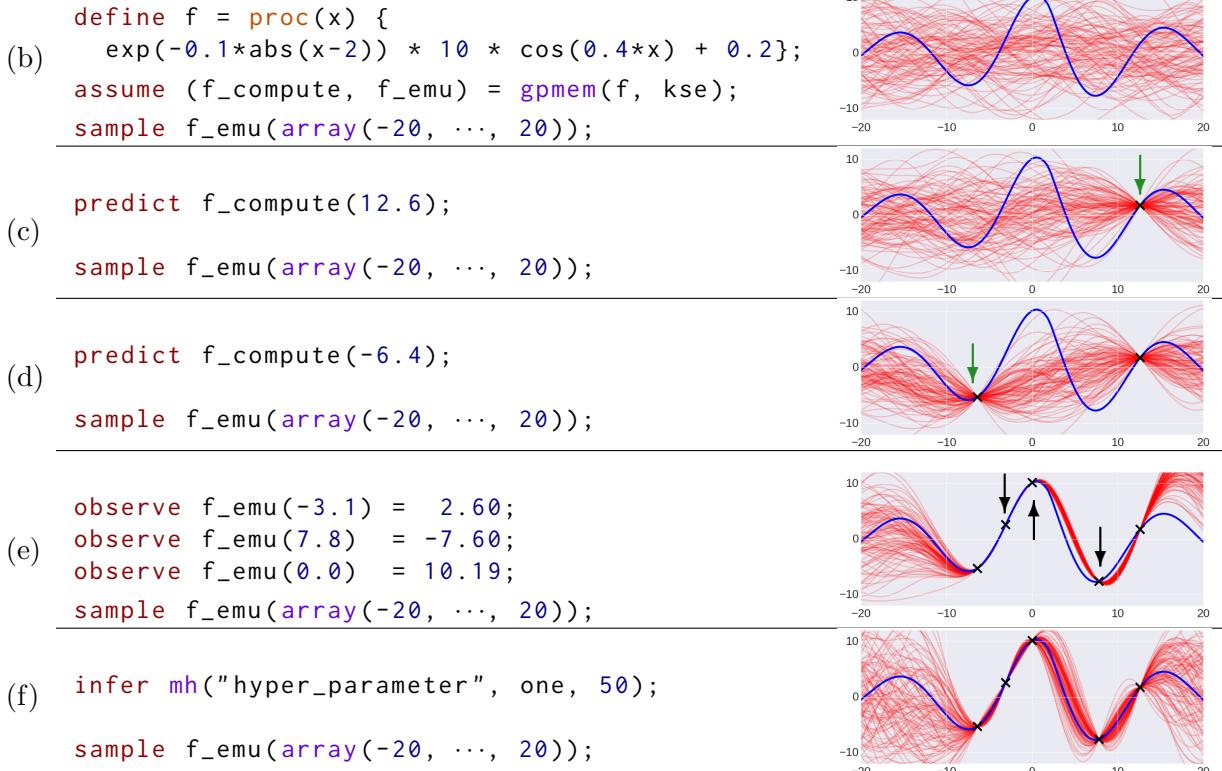
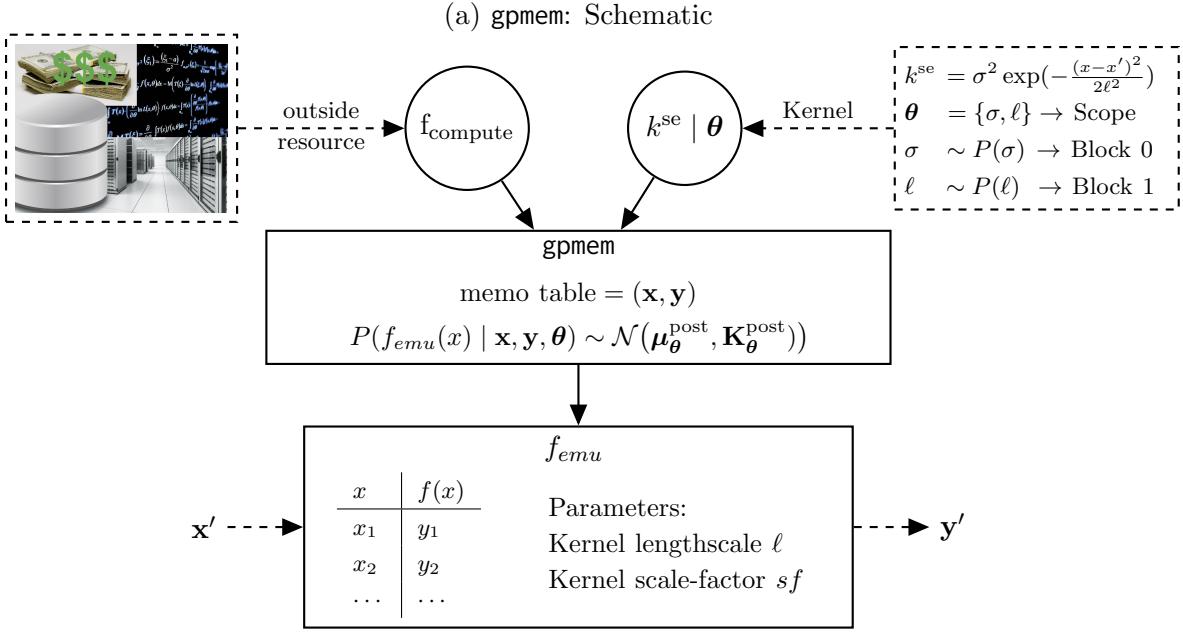


Figure 2: gpmem tutorial. The top shows a schematic of gpmem. f_{compute} probes an outside resource. This can be expensive (top left). Every probe is memoized and improves the emulator. Below the schematic we see the evolution of gpmem's state of belief of the world given certain Venture directives. On the right, we depict the true function (blue), samples from the emulator (red) and incorporated observations (black). We omit code for the hyper-parameters and kse , it can be found in Appendix D.

166 when f is computed with the help of outside resources. Before making any observations or
 167 calls to f we can sample from the prior at the inputs from -20 to 20 using the emulator:

```
168     assume (f_compute, f_emu) = gpmem(f, kse));
169
170     sample f_emu(array(-20, ..., 20));
```

171 where the second line corresponds to:

$$\mathbf{y}^* \sim \mathcal{N}\left(0, K^{\text{se}}\left(\begin{bmatrix}-20 \\ \dots \\ 20\end{bmatrix}, \begin{bmatrix}-20 \\ \dots \\ 20\end{bmatrix} \mid \boldsymbol{\theta} = \{\sigma, \ell\}\right)\right).$$

172 In Fig. 2 (c), we probe the external function f at point 12.6 and memoize it's result by
 173 calling

```
174     predict f_compute (12.6);
```

175 When we subsequently sample from the emulator, that is compute the \mathbf{y}' at the input
 176 $\mathbf{x}' = [-20, \dots, 20]^\top$. We see how the posterior which shifts from uncertainty to near
 177 certainty close to the input 12.6.

178 We can repeat the process at a different point (probing point -6.4 in Fig. 2 (d)) to see
 179 that we gain certainty about another part of the curve.

180 We can add information to f_{emu} about presumable value pairs of f without calling
 181 f_{compute} (Fig. 2 (e)). If a friend tells us the value of f we can call observe to store this
 182 information in the incorporated observations for f_{emu} only:

```
183     observe f_emu( -3.1 ) = 2.60;
```

184 We have this value pair now available for the computation \mathbf{y}' . For sampling with the
 185 emulator, the effect is the same as calling predict with the f_{compute} . However, we can
 186 imagine at least one scenario where such as distinction in the treatment of observations is
 187 beneficial. Let us say we do not only have the real function available but also a domain
 188 expert with knowledge about this function. This expert could tell us what the value is at
 189 a given input. Potentially, the value provided by the expert could disagree with the value
 190 computed with f for example due to different levels of observation noise.

191 Finally, we can update our posterior by inferring the posterior over hyper-parameter
 192 values $\boldsymbol{\theta}$. We can take 50 Metropolis-Hastings (MH) steps by calling that where we take an
 193 MH steps for either of the two blocks for the scope "hyper-parameter":

```
194     infer mh( "hyper-parameter", one, 50);
```

195 The newly inferred hyper-parameters allow us now adequately reflect uncertainty about the
 196 curve given all incorporated observations (compare Fig. 2, bottom panel (f) on the right
 197 with the samples before inference, one panel above (e)).

198 **4. Applications**

199 This paper illustrates the flexibility of `gpmem` by showing how it can concisely encode three
 200 different applications of GPs. The first is a standard example from hierarchical Bayesian
 201 statistics, where Bayesian inference over a hierarchical hyper-prior is used to provide a curve-
 202 fitting methodology that is robust to outliers. The second is a structure learning application
 203 from probabilistic artificial intelligence, where GPs are used to discover qualitative structure
 204 in time series data. The third is a reinforcement learning application, where GPs are used as
 205 part of a Thompson sampling formulation of Bayesian optimization for general real-valued
 206 objective functions with real inputs.

207 **4.1 Nonlinear regression in the presence of outliers**

208 We can apply `gpmem` for regression in a hierarchical Bayesian setting (Fig. 3). In a Bayesian
 209 treatment of hyper-parameter learning for GPs, we can write the posterior probability of
 210 the hyper-parameters of a GP (Fig. 3, (a)) given covariance function $\mathcal{K} = k$ as:

$$P(\boldsymbol{\theta} = \{sf, \ell, \sigma\} | \mathbf{D}, \mathcal{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathcal{K})P(\boldsymbol{\theta} | \mathcal{K})}{P(\mathbf{D} | \mathcal{K})} \quad (14)$$

211 where $\mathbf{D} = \{\mathbf{x}, \mathbf{y}\}$ is a training data set and \mathcal{K} is treated as a random variable over covariance
 212 functions. Since we can apply `gpmem` to any process or procedure, it can be used in situations
 213 where only a data set is available via a look-up function `f_look_up`. In fact, we demonstrate
 214 `gpmem`'s application to regression using an example where the data was generated by a
 215 function which is not available (Fig. 3 (b)). This function, f_{true} , is taken from a paper on
 216 the treatment of outliers with hierarchical Bayesian hyper-priors for GPs (Neal, 1997):

$$f_{true}(x) = 0.3 + 0.4x + 0.5 \sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad \text{with } \eta \sim \mathcal{N}(0, \sigma). \quad (15)$$

217 We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in
 218 the remaining cases. Instead of accessing the f_{true} directly, we are accessing the data in
 219 form of a two dimensional array with `f_look_up`.

220 We set $\mathcal{K} = k^{se+wn}$ and parameterize it with $\boldsymbol{\theta} = \{sf, \ell, \sigma\}$. For these hyper-parameters,
 221 Neals work suggests a hierarchical system for hyper-parameterization. Here, we draw
 222 hyper-parameters from Γ distributions:

$$\ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (16)$$

223 and in turn sample the α and β from Γ distributions as well:

$$\alpha_1^{(t)} \sim \Gamma(\alpha_\alpha^1, \beta_\alpha^1), \alpha_2^{(t)} \sim \Gamma(\alpha_\alpha^2, \beta_\alpha^2), \dots \quad (17)$$

224 We model this in Venture as illustrated in Fig. 3 (c), using the build-in SP `gamma`. Note
 225 that we omit the prior distributions of (17) due limited space in the tutorial.

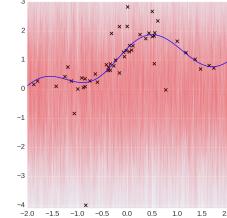
226 In Fig. 3, panel (d), we see that k^{se+wn} is defined as a composite covariance func-
 227 tion. It is the sum (`add_funcs`) of an squared exponential kernel (`make_squaredexp`) and a

(a) $P(\boldsymbol{\theta} = \{\ell, sf, \sigma\} | \mathbf{D}, \mathcal{K})$ 

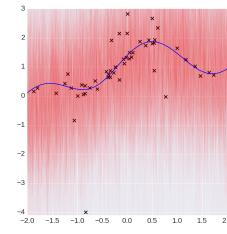
```
(b) // Data and look-up function
define data = array(array(-1.87, 0.13), ..., array(1.67, 0.81));
assume f_look_up = proc(index) {lookup(data, index)};
```

```
(c) assume sf = tag("hyper", 0, gamma(alpha_sf, beta_sf));
assume l = tag("hyper", 1, gamma(alpha_l, beta_l));
assume sigma = tag("hyper", 2, uniform_continuous(0, 2));
```

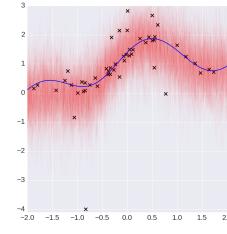
```
(d) // The covariance function
assume se = make_squaredexp(sf, l);
assume wn = make_whitenoise(sigma);
assume composite_covariance = add_funcs(se, wn);
// Create a prober and emulator using gpmem
assume (f_compute, f_emu) =
    gpmem(f_look_up, composite_covariance);
sample f_emu(array(-2, ..., 2));
```



```
(e) // Observe all data points
for (n = 0; n < size(data); n++){
    observe f_emu(first(lookup(data,n))) =
        second(lookup(data,n)));
// Or: probe all data points
for (n = 0; n < size(data); n++){
    predict f_compute(first(lookup(data,n)));
sample f_emu(array(-2, ..., 2));
```



```
(f) // Metropolis-Hastings
infer repeat(100, do(
    mh("hyperhyper", one, 2),
    mh("hyper", one, 1)));
sample f_emu(array(-2, ..., 2));
```



```
(g) // Optimization
infer map("hyper", all, 0.01, 15);
sample f_emu(array(-2, ..., 2));
```

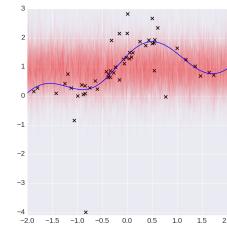


Figure 3: Regression with outliers and hierarchical prior structure

228 white noise (k^{wn} , Appendix A) kernel which is implemented with `make_whitenoise`¹. We
 229 then initialize `gpmem` feeding it with `composite_covariance` and the data look-up function
 230 `f_look_up`. We sample from the prior \mathbf{y}^* with random parameters `sf,1` and `sigma` and
 231 without any observations available. We depict those samples on the right (red), alongside
 232 the true function that generated the data (blue) and the data points we have available in
 233 the data set (black).

234 We can incorporate observations using both `observe` and `predict` (Fig. 3 (e)). When
 235 we subsequently sample \mathbf{y}' from the emulator with $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}^{\text{post}}, \mathbf{K}_{\boldsymbol{\theta}}^{\text{post}})$, we can see that the
 236 GP posterior incorporates knowledge about the `data`. Yet, the hyper-parameters `sf,1` and
 237 `sigma` are still random, so the emulator does not capture the true underlying dynamics
 238 (f_{true}) of the `data` correctly.

239 Next, we demonstrate how we can capture these underlying dynamics within only 100
 240 nested MH steps on the hyper-parameters to get a good approximation for their posterior \mathbf{y}'
 241 (Fig. 3 (f)). We say nested because we first take two sweeps in the scope `hyperhyper` which
 242 characterizes (17) and then one sweep on the scope `hyper` which characterizes (16). This
 243 is repeated 100 using `repeat(100, do(···)`. Note that Neal devises an additional noise
 244 model and performs a large number of Hybrid-Monte Carlo and Gibbs steps to achieve this,
 245 whereas inference in Venture with `gpmem` is merely one line of code.

246 Finally, we can change our inference strategy altogether. If we decide that instead of
 247 following a Bayesian sampling approach, we would like to perform empirical optimization,
 248 we do this by only changing one line of code, deploying `map` instead of `mh` (Fig. 3 (g)).

249 4.2 Discovering qualitative structure from time series data

250 Inductive learning of symbolic expressions for continuous-valued time series data is a hard
 251 task which has recently been tackled using a greedy search over the approximate posterior
 252 of the possible kernel compositions for GPs (Duvenaud et al., 2013; Lloyd et al., 2014)².

253 With `gpmem` we can provide a fully Bayesian treatment of this, previously unavailable,
 254 using a stochastic grammar (see Fig. 4). This allows us to read an unstructured time
 255 series and automatically output a high-level, qualitative description of it. The stochastic
 256 grammar takes a set of primitive base kernels $\text{BK} = \{k_{\boldsymbol{\theta}1}^1, \dots, k_{\boldsymbol{\theta}m}^m\}$ of size m , including
 257 their corresponding parametrization $\boldsymbol{\theta}^* = \{\boldsymbol{\theta}^1, \dots, \boldsymbol{\theta}^m\}$ (Fig. 4 (a) and (b)) We depict the
 258 input for the stochastic grammar in Listing 1.

1. Note that in Neal's work (1997) the sum of an SE plus a constant kernel is used. We use a WN kernel for illustrative purposes instead.

2. <http://www.automaticstatistician.com/>

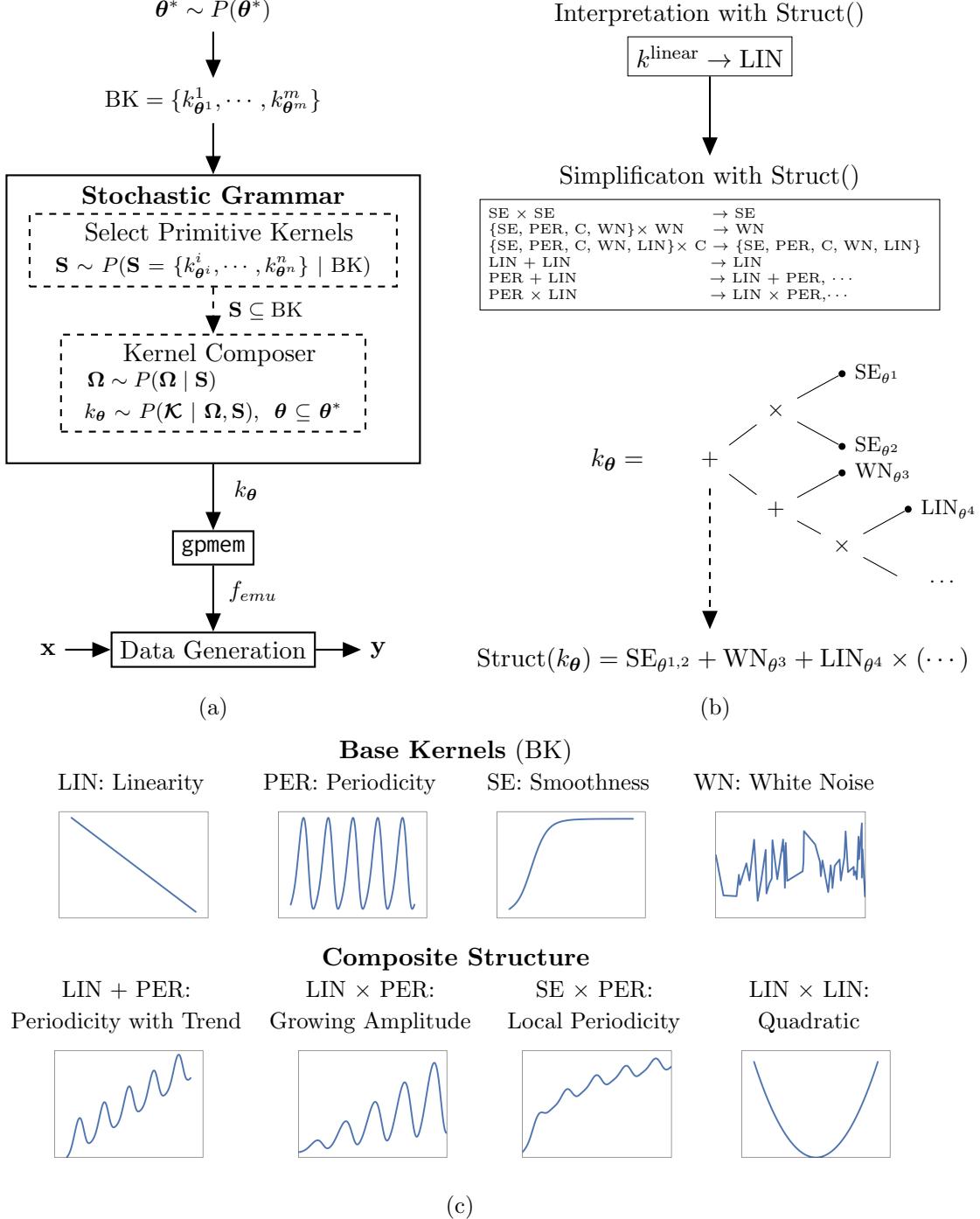


Figure 4: (a) Bayesian GP structure learning. A set of base kernels (BK) with priors on their hyper-parameters serve as hypothesis space and is supplied as input to the stochastic grammar. The stochastic grammar has two parts: (i) a sampler that selects a random set \mathbf{S} of primitive kernels from BK and (ii) a kernel composer that combines the individual base kernels and generates a composite kernel function k_{θ} . This serves as input for gpmem. We observe value pairs \mathbf{x}, \mathbf{y} of unstructured time series data on the bottom of the schematic. (b) An example of composite kernel structure. We use the $\text{Struct}(k_{\theta})$ to compute a symbolic interpretation for k_{θ} . We use addition and multiplication to combine base kernels. Base kernels and compositional kernels are shown in (c) alongside their interpretation with $\text{Struct}()$.

Listing 1: Initialize Base Kernels BK and $P(n)$

```

1 // Initialize hyper-parameters
2 assume theta_1 = tag("hyper-parameters", 1, gamma(5,1));
3 assume theta_2 = tag("hyper-parameters", 2, gamma(5,1));
4 assume theta_3 = tag("hyper-parameters", 3, gamma(5,1));
5 assume theta_4 = tag("hyper-parameters", 4, gamma(5,1));
6 assume theta_5 = tag("hyper-parameters", 5, gamma(5,1));
7 assume theta_6 = tag("hyper-parameters", 6, gamma(5,1));
8 assume theta_7 = tag("hyper-parameters", 7, gamma(5,1));
9
11 // Make kernels
12 assume lin = apply_function(make_linear, theta_1);
13 assume per = apply_function(make_periodic, theta_2, theta_3, theta_4);
14 assume se = apply_function(make_squaredexp, theta_5, theta_6);
15 assume wn = apply_function(make_noise, theta_7);
16
17 // Initialize the set of primitive base kernels BK
18 assume BK = list(lin, per, se, wn);

```

259 We sample a random subset S of the set of supplied base kernels. S is of size $n \leq m$.
260 We write

$$S = \{K_{\theta^i}^i, \dots, K_{\theta^n}^n\} \sim P(S = \{K_{\theta^i}^i, \dots, K_{\theta^n}^n\} | BK)$$

261 with

$$P(S = \{K_{\theta^i}^i, \dots, K_{\theta^n}^n\} | BK) = \frac{n!}{|S = \{K_{\theta^i}^i, \dots, K_{\theta^n}^n\}|!}.$$

262 BK is assumed to be fixed as the most general margin of our hypothesis space. In the
263 following, we will drop it in the notation. The only building block that we are now missing is
264 how to combine the sampled base kernels into a compositional covariance function (see Fig.
265 4 (b)). For each interaction i , we have to infer whether the data supports a local interaction
266 or a global interaction, choosing between one out of two algebraic operators $\Omega_i = \{+, \times\}$.
267 The probability for all such decisions is given by a binomial distribution:

$$P(\Omega | S) = \binom{n}{r} p_{+\times}^r (1 - p_{+\times})^{n-r}. \quad (18)$$

268 We can write the marginal probability of a kernel function as

$$P(K | x, y, \theta) = \iint_{\Omega, S} P(K | x, y, \theta, \Omega, S) \times P(\Omega | S) \times P(S) d\Omega dS \quad (19)$$

269 with $\theta \subseteq \theta^*$ as implied by S . For structure learning with GP kernels, a composite kernel
270 is sampled from $P(K)$ and fed into `gpmem`. The emulator generated by `gpmem` observes
271 unstructured time series data. Venture code for the probabilistic grammar is shown in
272 Listing 2, code for inference with `gpmem` in Listing 3.

Listing 2: Stochastic Grammar

```

1 // Select a random subset of the set possible primitive kernels (BK)
2 assume primitive_kernel_selection = tag("grammar", 0,
3                                         select_primitive_kernels(BK));
4 // Construct kernel composition with a composer procedure
5 assume kernel_composer = proc(l) {
6     if (size(l) <= 1) {
7         first(l)
8     } else {
9         if (bernoulli()) {
10            add_funcs(first(l), kernel_composer(rest(l)))
11        } else {
12            mult_funcs(first(l), kernel_composer(rest(l)))
13        }
14    }
15 };
16
17 assume K = tag("grammar", 1,
18                 kernel_composer(primitive_kernel_selection));

```

Listing 3: gpmem inference for structure learning:

```

1 // Apply gpmem
2 assume (f_compute, f_emu) = gpmem(f_look_up, K);
3 // Probe all data points
4 for (n = 0; n < size(data); n++) {
5     predict f_compute(first(lookup(data, n)));
6 // Perform inference
7 infer repeat(200, do(
8     mh("grammar", one, 1),
9     mh("hyper-parameters", one, 2)));

```

273 Many equivalent covariance structures can be sampled due to covariance function algebra
 274 and equivalent representations with different parameterization (Lloyd et al., 2014). To
 275 inspect the posterior of these equivalent structures we convert each kernel expression into
 276 a sum of products and subsequently simplify. All base kernels can be found in Appendix
 277 A, rules for this simplification can be found in Appendix B. Kernel structure comes with
 278 symbolic interpretations. To read kernel function k and apply the simplifications described
 279 above, we apply, Struct(k). For example, we write

$$\text{Struct}(k^{\text{linear}}) = \text{LIN},$$

280 which translates a function into a symbolic expression, see Appendix C. We defined a simple
 281 space of covariance structures in a way that allows us to produce results coherent with work
 282 presented in Automatic Statistician. The results are illustrated with two data sets.

283 **Mauna Loa CO₂ data**

284 We illustrate results in Fig 5. In Fig 5 (a) we depict the raw data. We see mean centered
 285 CO₂ measurements of the Mauna Loa Observatory, an atmospheric baseline station on
 286 Mauna Loa, on the island of Hawaii. A description of the data set can be found in Rasmussen
 287 and Williams, 2006, chapter 5. We use those raw data to compute a posterior on structure,
 288 parameters and GP samples. The latter are shown in Fig 5 (b) where we zoom in to show
 289 how the posterior captures the error bars adequately. This posterior of the GP is generated
 290 with a random sample from the parameters of the peak of the distribution on structure
 291 (Fig 5 (c)). We differentiate between a posterior distribution on kernel functions and on
 292 distribution on symbolic expressions describing different kernel structures. This allows us
 293 to compute the posterior of symbolically equivalent structures, such as $\text{Struct}(k^{\text{linear}} +$
 294 $k^{\text{periodic}}) = \text{Struct}(k^{\text{periodic}} + k^{\text{linear}})$. Both structures yield and addition of a linear kernel
 295 and a periodic kernel, that is LIN + PER. We coin the random variable over symbolic
 296 expressions for kernels as $\mathcal{K}_{\text{Struct}}$. The distribution peaks at:

$$\mathcal{K}_{\text{Struct}} = \text{LIN} + \text{PER} + \text{SE} + \text{WN}. \quad (20)$$

297 We write this Kernel equation out in Fig 5 (d). This kernel structure has a natural language
 298 interpretation that we spell out in Fig 5 (e), explaining that the posterior peaks at a kernel
 299 structure with four additive components. Each of which holds globally, that is there are no
 300 higher level, qualitative aspects of the data that vary with the input space. The additive
 301 components for this result are as follows:

- 302 • a linearly increasing function or trend;
- 303 • a periodic function;
- 304 • a smooth function; and
- 305 • white noise.

306 Previous work on automated kernel discovery (Duvenaud et al., 2013) illustrated the
 307 Mauna Loa data using an RQ kernel. We resort to the white noise kernel instead of RQ
 308 (similar to (Lloyd et al., 2014)).

309 **Airline Data**

310 The second data set (Fig. 6) we depict results for is the airline data set describing monthly
 311 totals of international airline passengers (Box et al., 1997, according to Duvenaud et al.,
 312 2013).

313 We illustrate results for this data set in Fig 6. In Fig 6 (a) we depict the raw data. Again,
 314 the data is mean centered and we use it to compute a posterior on structure, parameters
 315 and GP samples. The latter are shown in Fig 6 (b). This posterior of the GP is generated
 316 with a random sample from the parameters of the peak of the distribution on structure (Fig
 317 6 (c)). The posterior over symbolic kernel expressions peaks at:

$$\mathcal{K}_{\text{Struct}} = \text{LIN} + \text{SE} \times \text{PER} + \text{WN}. \quad (21)$$

318 We write this Kernel equation out in Fig 6 (d). This kernel structure has a natural language
 319 interpretation that we spell out in Fig 6 (e), explaining that the posterior peaks at a kernel

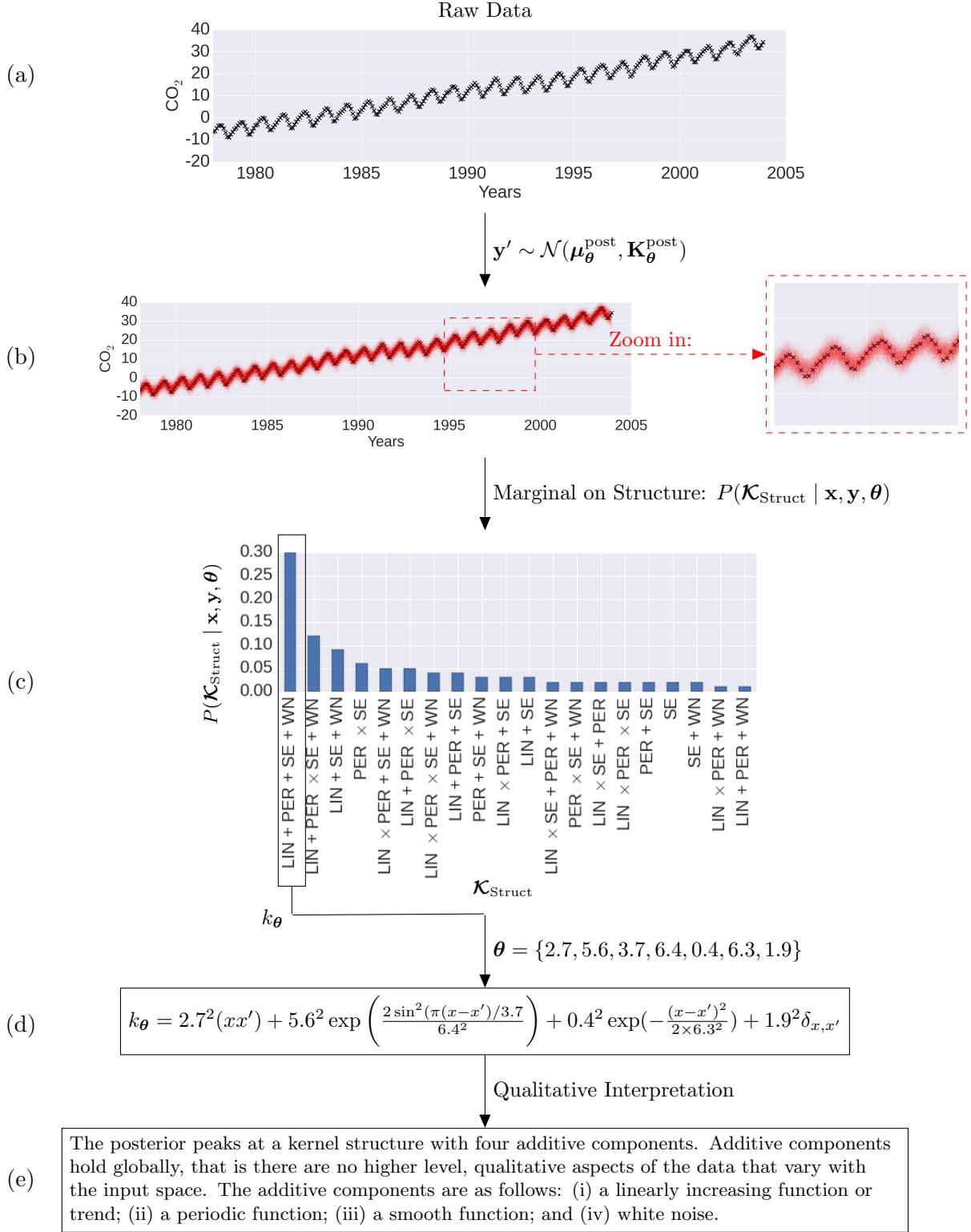


Figure 5: Structure Learning. Starting with raw data (a), we fit a GP (b) and compute the posterior distribution on structures (c). We take a sample of the peak of this distribution (LIN + PER + SE + WN) including its parameters and write it in functional form (d). We depict the human readable interpretation (e). We used (d) to plot (b).

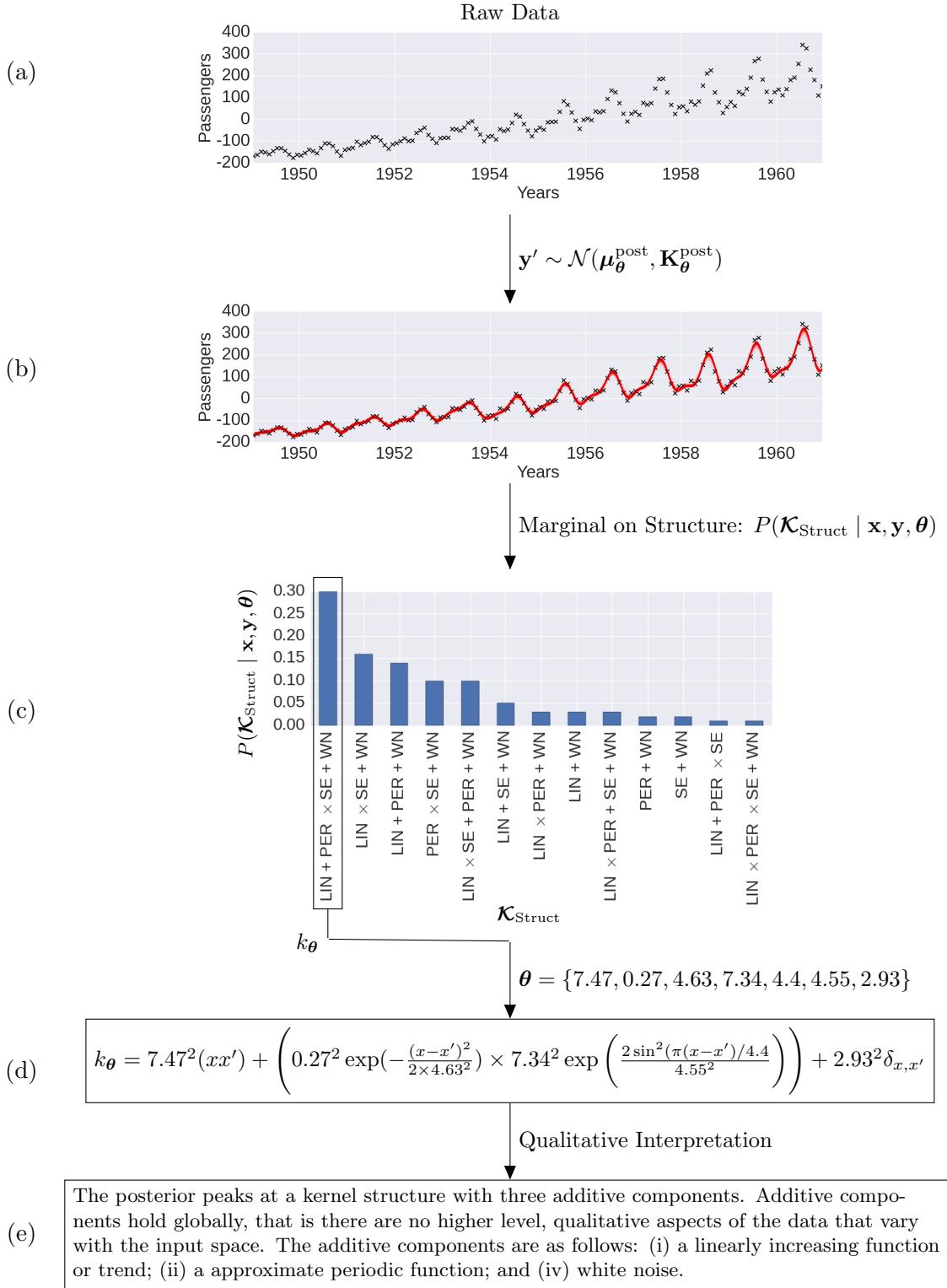


Figure 6: Structure Learning. Starting with raw data (a), we fit a GP (b) and compute the posterior distribution on structures (c). We take a sample of the peak of this distribution ($\text{LIN} + \text{PER} \times \text{SE} + \text{WN}$) including its parameters and write it in functional form (d). We depict the human readable interpretation (e). We used (d) to plot (b).

320 structure with three additive components. Additive components hold globally, that is there
 321 are no higher level, qualitative aspects of the data that vary with the input space. The
 322 additive components are as follows:

- 323 • a linearly increasing function or trend;
- 324 • a approximate periodic function; and
- 325 • white noise.

326 Both datasets served as illustrations in the Automatic Statistician project.

327 **Querying time series**

328 With our Bayesian approach to structure learning we can gain valuable insights into time
 329 series data that were previously unavailable. This is due to our ability to estimate posterior
 330 marginal probabilities over the kernel structure. Over this marginal, we define boolean
 331 search operations that allow us to query the data for the probability of certain structures
 332 to hold true globally.

$$P(\mathcal{K}_{\text{Struct}} \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{T} \sum_{t=1}^T f(\mathcal{K}_{\text{Struct}}^t) \quad \text{where } f(\mathcal{K}_{\text{Struct}}^t) = \begin{cases} 1, & \text{if } \mathcal{K}_{\text{Struct}}^t \in \mathcal{K}_{\text{Struct}}^t \\ 0, & \text{otherwise.} \end{cases} \quad (22)$$

333 to ask whether it is true that a global structure $\mathcal{K}_{\text{Struct}}$ is present. T is the number of
 334 all posterior samples for $\mathcal{K}_{\text{Struct}}$ and $\mathcal{K}_{\text{Struct}}^t$ is one such sample. We can now ask simple
 335 questions, for example:

336 Is there white noise in the data?

337 where we set $\mathcal{K}_{\text{Struct}} = \text{WN}$ in (22). We can also formulate more sophisticated search
 338 operations using Boolean operators such as AND (\wedge) and OR (\vee). The AND operator is
 339 defined as follows:

$$P(\mathcal{K}_{\text{Struct}}^a \wedge \mathcal{K}_{\text{Struct}}^b \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N f(\mathcal{K}_{\text{Struct}}^n)$$

340 where

$$f(\mathcal{K}_{\text{Struct}}^n) = \begin{cases} 1, & \text{if } \mathcal{K}_{\text{Struct}}^n \text{ and } \mathcal{K}_{\text{Struct}}^n \in \mathcal{K}_{\text{Struct}}^n \\ 0, & \text{otherwise} \end{cases}.$$

341 By estimating $P(\text{LIN} \wedge \text{WN} \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$ we can use this operator to ask questions such as

342 Is there a Linear component AND a white noise in the data?

Finally, we define the logical OR as

$$\begin{aligned} P(\mathcal{K}_{\text{Struct}}^a \vee \mathcal{K}_{\text{Struct}}^b \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) &= P(\mathcal{K}_{\text{Struct}}^a \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) + P(\mathcal{K}_{\text{Struct}}^b \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) \\ &\quad - P(\mathcal{K}_{\text{Struct}}^a \wedge \mathcal{K}_{\text{Struct}}^b \mid \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) \end{aligned}$$

343 which allows us to ask questions about structures that are logically connected with OR,
 344 such as:

345 Is there white noise or heteroskedastic noise?

346 by estimating $P(\text{LIN} \times \text{WN} \vee \text{WN} | \mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$. We know that noise can either be het-
347 eroskedastic or white, and we also know due to simple manipulations using kernel algebra
348 that $\text{LIN} \times \text{WN}$ and WN are the only possible ways to construct noise with kernel compo-
349 sition, we see that we can generalize the question above to:

350 Is there noise in the data?

351 where we write the marginal posterior on qualitative structure for noise:

$$P(\mathcal{K}_{\text{Struct}}^{\text{noise}} | \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = P(\text{LIN} \times \text{WN} \vee \text{WN} | \mathbf{x}, \mathbf{y}, \boldsymbol{\theta}). \quad (23)$$

352 Note that this allows us to start with general queries and subsequently formulate follow up
353 queries that go into more detail. For example, we could start with a general query, such as:

354 What is the probability of a trend, a recurring pattern **and** noise in the data?

355 and then follow up with more detailed questions (Fig 7). This way of querying your data
356 for their statistical implications is in stark contrast to what previous research in automatic
357 kernel construction was able to provide. We could view our approach as a time series search
358 engine which allows us to test whether or not certain structures can be found in an available
359 time series. Another way to view this approach is as a new language to interact with the
360 world. Real-world observations often come with time-stamps and in form of continuous
361 valued sensor measurements. We provide the toolbox to query such observations in a
362 similar manner as one would query a knowledge base in a logic programming language.

363 4.3 Bayesian optimization

364 The final application demonstrating the power of `gpmem` illustrates its use in Bayesian op-
365 timization. We introduce Thompson sampling, the basic solution strategy underlying the
366 Bayesian optimization with `gpmem`. Thompson sampling Thompson (1933) is a widely used
367 Bayesian framework for addressing the trade-off between exploration and exploitation in
368 multi-armed (or continuum-armed) bandit problems. We cast the multi-armed bandit prob-
369 lem as a one-state Markov decision process (MDP), and describe how Thompson sampling
370 can be used to choose actions for that MDP.

371 The MDP is as follows: An agent is to take a sequence of actions x_1, x_2, \dots from a
372 (possibly infinite) set of possible actions \mathcal{X} . After each action, a reward $y \in \mathbb{R}$ is received,
373 according to an unknown conditional distribution $P_{\text{true}}(y | x)$. The agent's goal is to max-
374 imize the total reward received for all actions in an online manner. In Thompson sampling,
375 the agent accomplishes this by placing a prior distribution $P(\vartheta)$ on the possible "contexts"
376 $\vartheta \in \Theta$. Here a context is a believed model of the conditional distributions $\{P(x | y)\}_{x \in \mathcal{X}}$,
377 or at least, a believed statistic of these conditional distributions which is sufficient for de-
378 ciding an action a . If actions are chosen so as to maximize expected reward, then one such
379 sufficient statistic is the believed conditional mean $V(x | \vartheta) = \mathbb{E}[y | x; \vartheta]$, which can be
380 viewed as a believed value function. For consistency with what follows, we will assume our
381 context ϑ takes the form $(\boldsymbol{\theta}, \mathbf{x}, \mathbf{y})$ where \mathbf{x} is the vector of past actions, \mathbf{y} is the vector of
382 their rewards, and $\boldsymbol{\theta}$ (the "semicontext") contains any other information that is included
383 in the context.

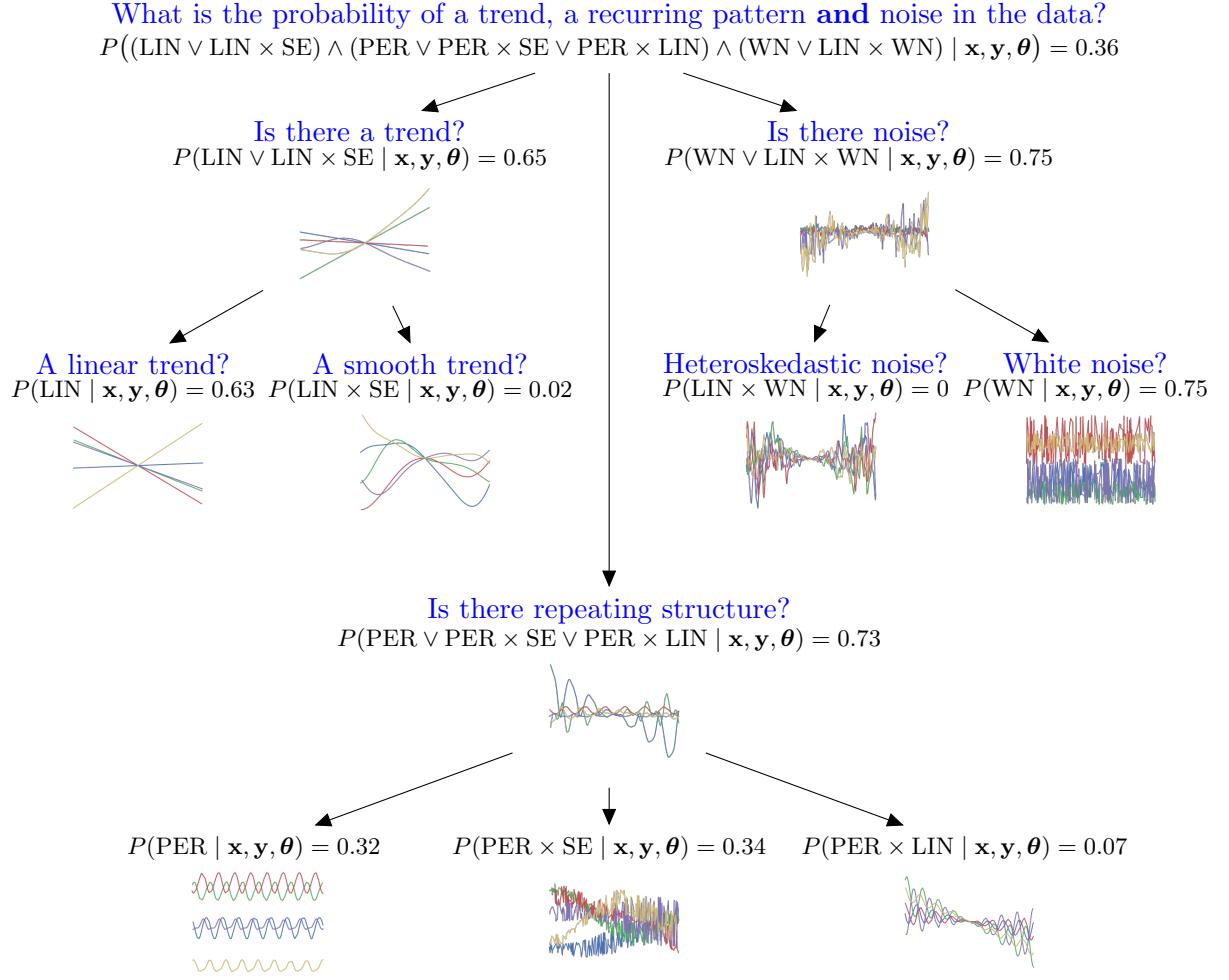


Figure 7: **Querying structural motifs in time series using posterior inference over kernel structure.** The kernel structure serves as a way to formulate natural language questions about the data (blue). The initial question of interest (top) is a fairly general one: "What is the probability of a trend, a recurring pattern and noise in the data?" Below the natural language version of this question, the same question is formulated as an inference problem (black) over the marginal probability on kernels with Boolean operators AND (\wedge) and OR (\vee). To gain a deeper understanding of specific motifs in the time series more specific queries can be written. On the right, a query asks whether there is noise in the data (blue) by computing the disjunction of the marginal of a global white noise kernel and a multiplication between a linear and a white noise kernel (black). Samples from the predictive prior \mathbf{y}_* of such kernels give an indication of the qualitative aspects that a kernel structure implies (coloured curves below the marginal). If the probability that there is noise in the data is high then it makes sense to drill even deeper asking more detailed questions. With regards to noise, this translates to querying whether or not the data supports the hypothesis that there is heteroskedastic noise or white noise. Queries for motifs of repeating structure are shown in the middle of the tree, queries related to trends on the left.

384 In this setup, Thompson sampling has the following steps:

Algorithm 1 Thompson sampling.

Repeat as long as desired:

1. **Sample.** Sample a semicontext $\boldsymbol{\theta} \sim P(\boldsymbol{\theta})$.
 2. **Search (and act).** Choose an action $x \in \mathcal{X}$ which (approximately) maximizes $V(x | \vartheta) = \mathbb{E}[y | x; \vartheta] = \mathbb{E}[y | x; \boldsymbol{\theta}, \mathbf{x}, \mathbf{y}]$.
 3. **Update.** Let y_{true} be the reward received for action x . Update the believed distribution on $\boldsymbol{\theta}$, i.e., $P(\boldsymbol{\theta}) \leftarrow P_{\text{new}}(\boldsymbol{\theta})$ where $P_{\text{new}}(\boldsymbol{\theta}) = P(\boldsymbol{\theta} | x \mapsto y_{\text{true}})$.
-

385 Note that when $\mathbb{E}[y | x; \vartheta]$ (under the sampled value of $\boldsymbol{\theta}$ for some points x) is far from
386 the true value $\mathbb{E}_{P_{\text{true}}}[y | x]$, the chosen action a may be far from optimal, but the informa-
387 tion gained by probing action x will improve the belief ϑ . This amounts to “exploration.”
388 When $\mathbb{E}[y | x; \vartheta]$ is close to the true value except at points x for which $\mathbb{E}[y | x; \vartheta]$ is low,
389 exploration will be less likely to occur, but the chosen actions x will tend to receive high re-
390 wards. This amounts to “exploitation.” The trade-off between exploration and exploitation
391 is illustrated in Figure 8. Roughly speaking, exploration will happen until the context ϑ is
392 reasonably sure that the unexplored actions are probably not optimal, at which time the
393 Thompson sampler will exploit by choosing actions in regions it knows to have high value.

394 Typically, when Thompson sampling is implemented, the search over contexts $\vartheta \in \Theta$ is
395 limited by the choice of representation. In traditional programming environments, $\boldsymbol{\theta}$ often
396 consists of a few numerical parameters for a family of distributions of a fixed functional
397 form. With work, a mixture of a few functional forms is possible; but without probabilistic
398 programming machinery, implementing a rich context space Θ would be an unworkably
399 large technical burden. In a probabilistic programming language, however, the representation
400 of heterogeneously structured or infinite-dimensional context spaces is quite natural. Any
401 computable model of the conditional distributions $\{P(y | x)\}_{x \in \mathcal{X}}$ can be represented as a
402 stochastic procedure $(\lambda(x) \dots)$. Thus, for computational Thompson sampling, the most
403 general context space $\widehat{\Theta}$ is the space of program texts. Any other context space Θ has a
404 natural embedding as a subset of $\widehat{\Theta}$.

405 **A Mathematical Specification**

406 We now describe a particular case of Thompson sampling with the following properties:

- 407
- The regression function has a Gaussian process prior.

408

 - The actions $x_1, x_2, \dots \in \mathcal{X}$ are chosen by a Metropolis-like search strategy with Gaus-
409 sian drift proposals.

410

 - The hyperparameters of the Gaussian process are inferred using Metropolis–Hastings
411 sampling after each action.

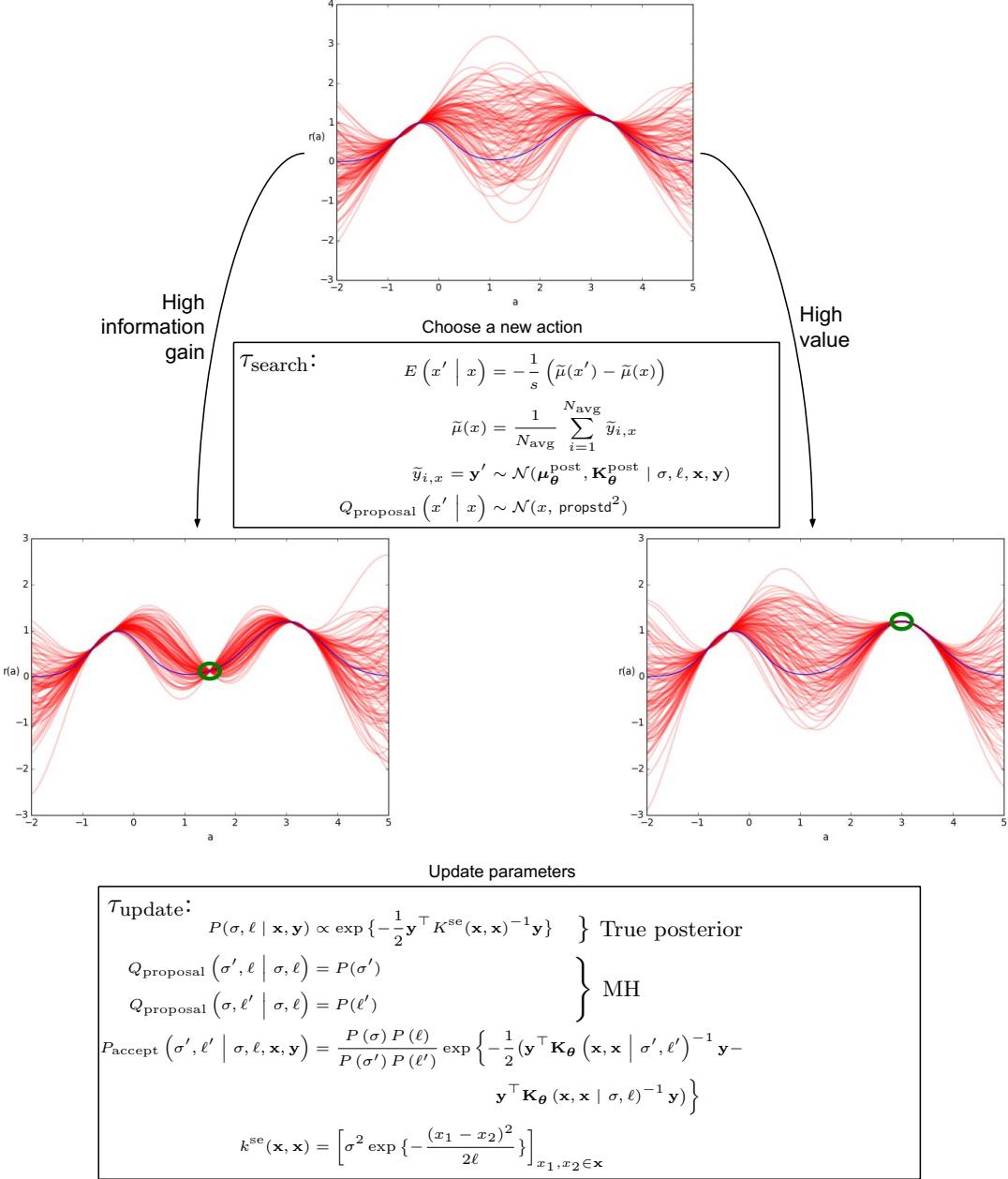


Figure 8: Two possible actions (in green) for an iteration of Thompson sampling. The believed distribution on the value function V is depicted in red. In this example, the true reward function is deterministic, and is drawn in blue. The action on the right receives a high reward, while the action on the left receives a low reward but greatly improves the accuracy of the believed distribution on V . The transition operators τ_{search} and τ_{update} are described in Section 4.3.

412 In this version of Thompson sampling, the contexts ϑ are Gaussian processes over the
413 action space $\mathcal{X} = [-20, 20] \subseteq \mathbb{R}$. That is,

$$V \sim \mathcal{GP}(\mu, k),$$

414 where the mean μ is a computable function $\mathcal{X} \rightarrow \mathbb{R}$ and the covariance k is a computable
415 (symmetric, positive-semidefinite) function $\mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. This represents a Gaussian process
416 $\{R_a\}_{a \in \mathcal{X}}$, where R_x represents the reward for action x . We write past actions as \mathbf{x} and past
417 rewards as \mathbf{y} . Computationally, we represent a context as a data structure

$$\vartheta = (\boldsymbol{\theta}, \mathbf{x}, \mathbf{y}) = (\mu, k, \boldsymbol{\theta}, \mathbf{x}, \mathbf{y}),$$

418 where $\boldsymbol{\mu}$ is a procedure to be used as the prior mean function and k is a procedure to be
419 used as the prior covariance function, parameterized by $\boldsymbol{\theta}$. As above set $\mu \equiv 0$.

420 Note that the context space Θ is not a finite-dimensional parametric family, since the
421 vectors \mathbf{x} and \mathbf{y} grow as more samples are taken. Θ is, however, representable as a computa-
422 tional procedure together with parameters and past samples, as we do in the representation
423 $\vartheta = (\mu, k, \boldsymbol{\theta}, \mathbf{x}, \mathbf{y})$.

424 We combine the Update and Sample steps of Algorithm 1 by running a Metropolis–
425 Hastings (MH) sampler whose stationary distribution is the posterior $P(\boldsymbol{\theta} | \mathbf{x}, \mathbf{y})$. The
426 functional forms of $\boldsymbol{\mu}$ and $\mathbf{K}_{\boldsymbol{\theta}}$ are fixed in our case, so inference is only done over the
427 parameters $\boldsymbol{\theta} = \{\sigma, \ell\}$; hence we equivalently write $P(\sigma, \ell | \mathbf{x}, \mathbf{y})$ for the stationary dis-
428 tribution. We make MH proposals to one variable at a time, using the prior as proposal
429 distribution:

$$Q_{\text{proposal}}(\sigma' | \sigma, \ell) = P(\sigma')$$

430 and

$$Q_{\text{proposal}}(\ell' | \sigma, \ell) = P(\ell').$$

431 The MH acceptance probability for such a proposal is

$$P_{\text{accept}}(\sigma', \ell' | \sigma, \ell) = \min \left\{ 1, \frac{Q_{\text{proposal}}(\sigma, \ell | \sigma', \ell')}{Q_{\text{proposal}}(\sigma', \ell' | \sigma, \ell)} \cdot \frac{P(\mathbf{x}, \mathbf{y} | \sigma', \ell')}{P(\mathbf{x}, \mathbf{y} | \sigma, \ell)} \right\}$$

Because the priors on σ and ℓ are uniform in our case, the term involving Q_{proposal} equals 1 and we have simply

$$\begin{aligned} P_{\text{accept}}(\sigma', \ell' | \sigma, \ell) &= \min \left\{ 1, \frac{P(\mathbf{x}, \mathbf{y} | \sigma', \ell')}{P(\mathbf{x}, \mathbf{y} | \sigma, \ell)} \right\} \\ &= \min \left\{ 1, \exp \left(-\frac{1}{2} \left(\mathbf{y}^T K(\mathbf{x}, \mathbf{x} | \sigma', \ell')^{-1} \mathbf{y} \right. \right. \right. \\ &\quad \left. \left. \left. - \mathbf{y}^T K(\mathbf{x}, \mathbf{x} | \sigma, \ell)^{-1} \mathbf{y} \right) \right) \right\}. \end{aligned}$$

432 The proposal and acceptance/rejection process described above define a transition operator
433 τ_{update} which is iterated a specified number of times; the resulting state of the MH Markov
434 chain is taken as the sampled semicontext $\boldsymbol{\theta}$ in Step 1 of Algorithm 1.

435 For Step 2 (Search) of Thompson sampling, we explore the action space using an MH-
 436 like transition operator τ_{search} . As in MH, each iteration of τ_{search} produces a proposal which
 437 is either accepted or rejected, and the state of this Markov chain after a specified number
 438 of steps is the new action a . The Markov chain's initial state is the most recent action, and
 439 the proposal distribution is Gaussian drift:

$$Q_{\text{proposal}}(x' | x) \sim \mathcal{N}(x, \text{propstd}^2),$$

440 where the drift width `propstd` is specified ahead of time. The acceptance probability of
 441 such a proposal is

$$P_{\text{accept}}(x' | x) = \min \{1, \exp(-E(x' | x))\},$$

442 where the energy function $E(\bullet | a)$ is given by a Monte Carlo estimate of the difference in
 443 value from the current action:

$$E(x' | x) = -\frac{1}{s} (\tilde{\mu}(x') - \tilde{\mu}(x))$$

444 where

$$\tilde{\mu}(x) = \frac{1}{N_{\text{avg}}} \sum_{i=1}^{N_{\text{avg}}} \tilde{y}_{i,x}$$

445 and

$$\tilde{y}_{i,x} = \mathbf{y}' \sim \mathcal{N}(\boldsymbol{\mu}_{\theta}^{\text{post}}, \mathbf{K}_{\theta}^{\text{post}})$$

446 and $\{\tilde{y}_{i,a}\}_{i=1}^{N_{\text{avg}}}$ are i.i.d. for a fixed a . (In the above, $\boldsymbol{\mu}_{\theta}^{\text{post}}$ and $\mathbf{K}_{\theta}^{\text{post}}$ are the mean and
 447 variance of a posterior sample at the single point $\mathbf{x}' = (x')$.) Here the temperature parameter
 448 $s \geq 0$ and the population size N_{avg} are specified ahead of time. Proposals of estimated value
 449 higher than that of the current action are always accepted, while proposals of estimated
 450 value lower than that of the current action are accepted with a probability that decays
 451 exponentially with respect to the difference in value. The rate of the decay is determined by
 452 the temperature parameter s , where high temperature corresponds to generous acceptance
 453 probabilities. For $s = 0$, all proposals of lower value are rejected; for $s = \infty$, all proposals
 454 are accepted. For points a at which the posterior mean $\boldsymbol{\mu}_{\theta}^{\text{post}}$ is low but the posterior
 455 variance $\mathbf{K}_{\theta}^{\text{post}}$ is high, it is possible (especially when N_{avg} is small) to draw a “wild” value
 456 of $\tilde{\mu}(a)$, resulting in a favorable acceptance probability.

457 Indeed, taking an action a with low estimated value but high uncertainty serves the
 458 useful function of improving the accuracy of the estimated value function at points near a
 459 (see Figure 8).^{3,4} We see a complete probabilistic program with `gpmem` implementing Bayesian
 460 optimization with Thompson Sampling below (Listing 4).

3. At least, this is true when we use a smoothing prior covariance function such as the squared exponential.
 4. For this reason, we consider the sensitivity of $\hat{\mu}$ to uncertainty to be a desirable property; indeed, this is
 why we use $\hat{\mu}$ rather than the exact posterior mean μ .

Listing 4: Bayesian optimization using gpmem

```

1  assume sf = tag("hyper", 0, uniform_continuous(0, 10));
2  assume l = tag("hyper", 1, uniform_continuous(0, 10));
3  assume se = make_squaredexp(sf, l);
4  assume blackbox_f = get_bayesopt_blackbox();
5  assume (f_compute, f_emulate) = gpmem(blackbox_f, se);
6
7  // A naive estimate of the argmax of the given function
8  define mc_argmax = proc(func) {
9    candidate_xs = mapv(proc(i) {uniform_continuous(-20, 20)},
10                        arange(20));
11   candidate_ys = mapv(func, candidate_xs);
12   lookup(candidate_xs, argmax_of_array(candidate_ys))
13 };
14
15 // Shortcut to sample the emulator at a single point without packing
16 // and unpacking arrays
17 define emulate_pointwise = proc(x) {
18   run(sample(lookup(f_emulate(array(unquote(x))), 0)))
19 };
20
21 // Main inference loop
22 infer repeat(15, do(pass,
23   // Probe V at the point mc_argmax(emulate_pointwise)
24   predict(f_compute(unquote(mc_argmax(emulate_pointwise))),
25   // Infer hyperparameters
26   mh("hyper", one, 50)));

```

461 In Fig. 9 we show results for our implementation of Bayesian Optimization with Thomp-
462 son sampling. We compare two different proposal distributions, namely uniform proposals
463 and Gaussian drift proposals. We see that in this experiment, Gaussian drift is starting near
464 the global optimum and drifts quickly towards it. (red curve, top panel of Fig. 9). Uniform
465 proposals take longer to find the global optimum (blue curve, top panel of Fig. 9) but we
466 see that it can surpass the local optima of the curve⁵. The bottom panel of Fig. 9 depicts
467 a sequence of actions using uniform proposals. The sequence illustrates the exploitation
468 exploration trade-off that the implementation overcomes. We start with complete uncer-
469 tainty ($i = 2$). The Bayesian agent performs exploration until it gets a (wrong!) idea of
470 where the optimum could be (exploiting the local optima $i = 5$ to $i = 10$). $i = 11$ shows a
471 change in tactic. The Bayesian agent, having exploited the local optima in previous steps,
472 is now reducing uncertainty in area it knows nothing about, eventually finding the global
473 optimum.

5. In fact, repeated experiments have shown that when the Gaussian drift proposals starts near a local optimum, it gets stuck there. Uniform proposals do not.

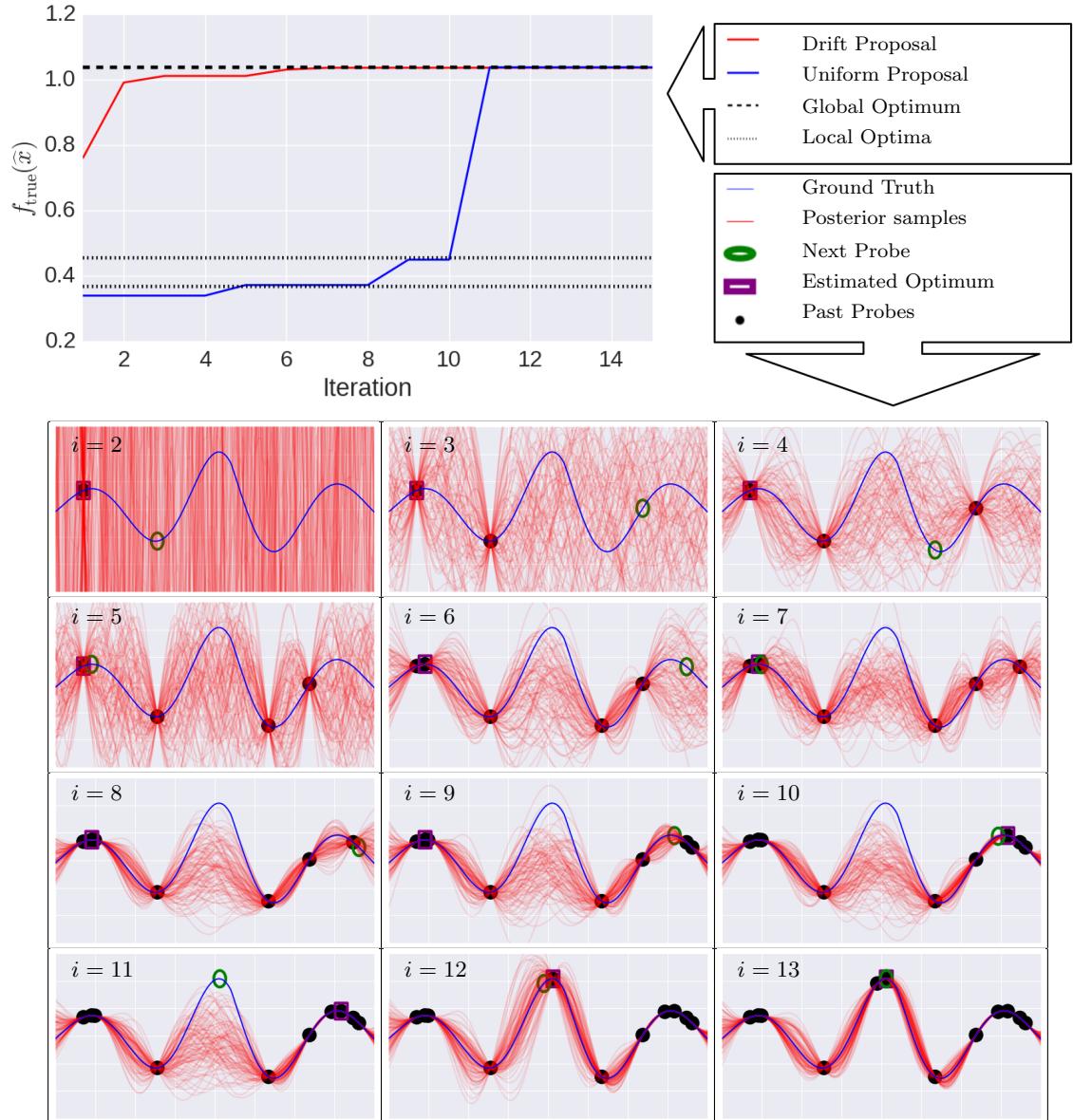


Figure 9: Top: the estimated optimum over time. Blue and Red represent optimization with uniform and Gaussian drift proposals. Black lines indicate the local optima of the true functions. Bottom: a sequence of actions. Depicted are iterations 7-12 with uniform proposals.

474 **5. Discussion**

475 We provided `gpmem`, an elegant linguistic framework for function learning-related tasks such
476 as Bayesian optimization and GP kernel structure learning. We highlighted how `gpmem`
477 overcomes shortcomings of the notations currently used in statistics, and how language
478 constructs from programming allow the expression of models which would be cumbersome
479 (prohibitively so, in some cases) to express in statistics notation. We evaluated our contri-
480 bution on a range of hard problems for state-of-the-art Bayesian nonparametrics.

481 **Appendix**

482 **A Covariance Functions**

$$\text{SE} = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (24)$$

$$\text{LIN} = \sigma^2(xx') \quad (25)$$

$$\text{C} = \sigma^2 \quad (26)$$

$$\text{WN} = \sigma^2 \delta_{x,x'} \quad (27)$$

$$\text{RQ} = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha} \quad (28)$$

$$\text{PER} = \sigma^2 \exp\left(\frac{2\sin^2(\pi(x - x')/p)}{\ell^2}\right). \quad (29)$$

483 **B Covariance Simplification**

$\text{SE} \times \text{SE}$	$\rightarrow \text{SE}$
$\{\text{SE}, \text{PER}, \text{C}, \text{WN}\} \times \text{WN}$	$\rightarrow \text{WN}$
$\text{LIN} + \text{LIN}$	$\rightarrow \text{LIN}$
$\{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\} \times \text{C}$	$\rightarrow \{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\}$

485 Rule 1 is derived as follows:

$$\begin{aligned} \sigma_c^2 \exp\left(-\frac{(x - x')^2}{2\ell_c^2}\right) &= \sigma_a^2 \exp\left(-\frac{(x - x')^2}{2\ell_a^2}\right) \times \sigma_b^2 \exp\left(-\frac{(x - x')^2}{2\ell_b^2}\right) \\ &= \sigma_c^2 \exp\left(-\frac{(x - x')^2}{2\ell_a^2}\right) \times \exp\left(-\frac{(x - x')^2}{2\ell_b^2}\right) \\ &= \sigma_c^2 \exp\left(-\frac{(x - x')^2}{2\ell_a^2} - \frac{(x - x')^2}{2\ell_b^2}\right) \\ &= \sigma_c^2 \exp\left(-\frac{(x - x')^2}{2\ell_c^2}\right) \end{aligned} \quad (30)$$

486 For stationary kernels that only depend on the lag vector between x and x' it holds that
 487 multiplying such a kernel with a WN kernel we get another WN kernel (Rule 2). Take for
 488 example the SE kernel:

$$\sigma_a^2 \exp\left(-\frac{(x - x')^2}{2\ell_c^2}\right) \times \sigma_b \delta_{x,x'} = \sigma_a \sigma_b \delta_{x,x'} \quad (31)$$

489 Rule 3 is derived as follows:

$$\theta_c(x \times x') = \theta_a(x \times x') + \theta_b(x \times x') \quad (32)$$

490 Multiplying any kernel with a constant obviously changes only the scale parameter of a
 491 kernel (Rule 4).

492 **C The Struct-Operator**

$$\begin{aligned}\text{Struct}(k^{\text{linear}}) &= \text{LIN} \\ \text{Struct}(k^{\text{periodic}}) &= \text{PER} \\ \text{Struct}(k^{\text{se}}) &= \text{SE} \\ \text{Struct}(k^{\text{wn}}) &= \text{WN} \\ \text{Struct}(k^{\text{linear+periodic}}) &= \text{LIN} + \text{PER} \\ \text{Struct}(k^{\text{linear}\times\text{periodic}}) &= \text{LIN} \times \text{PER}\end{aligned}$$

493 **D Squared Exponential Kernel in Venture**

```
494  
495 assume sf = tag("hyper-parameters", 1, gamma(5,1));  
496 assume l = tag("hyper-parameters", 2, gamma(5,1));  
497 assume kse = apply_function(make_squaredexp, sf, l);
```

498 References

- 499 Stephen Barnett and Stephen Barnett. *Matrix methods for engineers and scientists*.
500 McGraw-Hill, 1979.
- 501 D. Barry. Nonparametric bayesian regression. *The Annals of Statistics*, 14(3):934–953,
502 1986.
- 503 G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis: forecasting and
504 control*. 1997.
- 505 A. Damianou and N. Lawrence. Deep gaussian processes. In *Proceedings of the International
506 Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 207–215, 2013.
- 507 D. Duvenaud, J. R. Lloyd, R. Grosse, J. Tenenbaum, and Z. Ghahramani. Structure dis-
508 covery in nonparametric regression through compositional kernel search. In *Proceedings
509 of the International Conference on Machine Learning (ICML)*, pages 1166–1174, 2013.
- 510 B. Ferris, D. Haehnel, and D. Fox. Gaussian processes for signal strength-based location
511 estimation. In *Proceedings of the Conference on Robotics Science and Systems*. Citeseer,
512 2006.
- 513 M. A. Gelbart, J. Snoek, and R. P. Adams. Bayesian optimization with unknown constraints.
514 *arXiv preprint arXiv:1403.5607*, 2014.
- 515 M. Kemmler, E. Rodner, E. Wacker, and J. Denzler. One-class classification with gaussian
516 processes. *Pattern Recognition*, 46(12):3507–3518, 2013.
- 517 M. C. Kennedy and A. O’Hagan. Bayesian calibration of computer models. *Journal of the
518 Royal Statistical Society. Series B, Statistical Methodology*, pages 425–464, 2001.
- 519 M. Kuss and C. E. Rasmussen. Assessing approximate inference for binary gaussian process
520 classification. *The Journal of Machine Learning Research*, 6:1679–1704, 2005.
- 521 J. Kwan, S. Bhattacharya, K. Heitmann, and S. Habib. Cosmic emulation: The
522 concentration-mass relation for wcdm universes. *The Astrophysical Journal*, 768(2):123,
523 2013.
- 524 J. R. Lloyd, D. Duvenaud, R. Grosse, J. Tenenbaum, and Z. Ghahramani. Automatic
525 construction and natural-language description of nonparametric regression models. In
526 *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2014.
- 527 David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and
528 Systems Sciences*, 168:133–166, 1998.
- 529 V. K. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic pro-
530 gramming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.
- 531 R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto,
532 1995.

- 533 R. M. Neal. Monte carlo implementation of gaussian process models for bayesian regression
534 and classification. *arXiv preprint physics/9701026*, 1997.
- 535 C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive
536 Computation and Machine Learning)*. The MIT Press, 2006.
- 537 M. D. Schneider, L. Knox, S. Habib, K. Heitmann, D. Higdon, and C. Nakhleh. Simulations
538 and cosmological inference: A statistical model for power spectra means and covariances.
539 *Physical Review D*, 78(6):063529, 2008.
- 540 J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine
541 learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*,
542 pages 2951–2959, 2012.
- 543 W. R. Thompson. On the likelihood that one unknown probability exceeds another in view
544 of the evidence of two samples. *Biometrika*, pages 285–294, 1933.
- 545 C. K. I. Williams and D. Barber. Bayesian classification with gaussian processes. *IEEE
546 Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.