
Probabilistic Programming with Gaussian Process Memoization

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous Author(s)

Affiliation
Address
email

Abstract

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

1 Introduction

Gaussian Processes (GP) are widely used tools in statistics (Barry, 1986), machine learning (Neal, 1995; Williams and Barber, 1998; Kuss and Rasmussen, 2005; Rasmussen and Williams, 2006; Damianou and Lawrence, 2013), robotics (Ferris et al., 2006), computer vision (Kemmler et al., 2013), and scientific computation (Kennedy and O’Hagan, 2001; Schneider et al., 2008; Kwan et al., 2013). They are also central to probabilistic numerics, an emerging effort to develop more computationally efficient numerical procedures, and to Bayesian optimization, a family of meta-optimization techniques that are widely used to tune parameters for deep learning algorithms (Snoek et al., 2012; Gelbart et al., 2014). They have even seen use in artificial intelligence; for example, they provide the key technology behind a project that produces qualitative natural language descriptions of time series (Duvenaud et al., 2013; Lloyd et al., 2014).

This paper describes Gaussian process memoization, a technique for integrating GPs into a probabilistic programming language, and demonstrates its utility by re-implementing and extending state-of-the-art applications of the GP. Memoization, typically implemented by a procedure called `mem()`, is a classic higher-order programming technique in which a procedure is augmented with an input-output cache that is checked each time before the function is invoked. This prevents unnecessary recomputation, potentially saving time at the cost of increased storage requirements. Gaussian process memoization, implemented by the `gpmem()` procedure, generalizes this idea to include a statistical emulator that uses previously computed values as data in a statistical model that can cheaply forecast probable outputs. The covariance function for the Gaussian process is also allowed to be an arbitrary probabilistic program.

This paper presents three applications of `gpmem`: (i) a replication of results (Neal, 1997) on outlier rejection via hyper-parameter inference; (ii) a fully Bayesian extension to the Automated Statis-

054 tician project; and (iii) an implementation of Bayesian optimization via Thompson sampling. The
 055 first application can in principle be replicated in several other probabilistic languages embedding the
 056 proposal that is described in this paper. The remaining two applications rely on distinctive capabili-
 057 ties of Venture: support for fully Bayesian structure learning and language constructs for inference
 058 programming. All applications share a single 50-line Python library and require fewer than 20 lines
 059 of probabilistic code each.

061 2 Background on Gaussian Processes

063 Let the data be pairs of real-valued scalars $\{(x_i, y_i)\}_{i=1}^n$ (complete data will be denoted by column
 064 vectors \mathbf{x}, \mathbf{y}). GPs present a non-parametric way to express prior knowledge on the space of possible
 065 functions f modeling a regression relationship. Formally, a GP is an infinite-dimensional extension
 066 of the multivariate Gaussian distribution. Often one assumes the values \mathbf{y} are noisily measured, that
 067 is, one only sees the values of $\mathbf{y}_{\text{noisy}} = \mathbf{y} + \mathbf{w}$ where \mathbf{w} is Gaussian white noise with variance σ_{noise}^2 .
 068 In that case, the log-likelihood of a GP is

$$069 \log p(\mathbf{y}_{\text{noisy}} | \mathbf{x}) = -\frac{1}{2}\mathbf{y}^\top(\Sigma + \sigma_{\text{noise}}^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log|\Sigma + \sigma_{\text{noise}}^2\mathbf{I}| - \frac{n}{2}\log 2\pi \quad (1)$$

070 where n is the number of data points. Both log-likelihood and predictive posterior can be computed
 071 efficiently in a Venture SP with an algorithm that resorts to Cholesky factorization(Rasmussen and
 072 Williams, 2006, chap. 2) resulting in a computational complexity of $\mathcal{O}(n^3)$ in the number of data
 073 points.
 074

075 The covariance function (or kernel) of a GP governs high-level properties of the observed data such
 076 as linearity, periodicity and smoothness. It comes with few free parameters that we call hyper-
 077 parameters. Adjusting these results in minor changes, for example with regards to when two data
 078 points are treated similar. More drastically different covariance functions are achieved by changing
 079 the structure of the covariance function itself. Note that covariance function structures are compo-
 080 sitional: adding or multiplying two valid covariance functions results in another valid covariance
 081 function.
 082

083 Venture includes the primitive `make_gp`, which takes as arguments a unary function `mean` and a
 084 binary (symmetric, positive-semidefinite) function `cov` and produces a function `g` distributed as a
 085 Gaussian process with the supplied mean and covariance. For example, a function $g \sim \mathcal{GP}(0, \text{SE})$,
 086 where `SE` is a squared-exponential covariance

$$087 \text{SE}(x, x') = \sigma^2 \exp\left(\frac{(x - x')^2}{2\ell}\right)$$

088 with $\sigma = 1$ and $\ell = 1$, can be instantiated as follows:
 089

```
090 assume zero = make_const_func( 0.0 )
091 assume se = make_squaredexp( 1.0, 1.0 )
092 assume g = make_gp( zero, se )
```

093 There are two ways to view `g` as a “random function.” In the first view, the `assume` directive that
 094 instantiates `g` does not use any randomness—only the subsequent calls to `g` do—and coherence
 095 constraints are upheld by the interpreter by keeping track of which evaluations of `g` exist in the current
 096 execution trace. Namely, if the current trace contains evaluations of `g` at the points x_1, \dots, x_N with
 097 return values y_1, \dots, y_N , then the next evaluation of `g` (say, jointly at the points x_{N+1}, \dots, x_{N+n})
 098 will be distributed according to the joint conditional distribution
 099

$$100 P((g x_{N+1}), \dots, (g x_{N+n}) | (g x_i) = y_i \text{ for } i = 1, \dots, N).$$

101 In the second view, `g` is a randomly chosen deterministic function, chosen from the space of all
 102 deterministic real-valued functions; in this view, the `assume` directive contains *all* the randomness,
 103 and subsequent invocations of `g` are deterministic. The first view is procedural and is faithful to the
 104 computation that occurs behind the scenes in Venture. The second view is declarative and is faithful
 105 to notations like “ $g \sim P(g)$ ” which are often used in mathematical treatments. Because a model
 106 program could make arbitrarily many calls to `g`, and the joint distribution on the return values of the
 107

108 calls could have arbitrarily high entropy, it is not computationally possible in finite time to choose
 109 the entire function g all at once as in the second view. Thus, it stands to reason that any computationally
 110 implementable notion of “nonparametric random functions” must involve incremental random
 111 choices in one way or another, and Gaussian processes in Venture are no exception.
 112

113 2.1 The `gpmem` construct

114
 115 Memoization is the practice of storing previously computed values of a function so that future calls
 116 with the same inputs can be evaluated by lookup rather than recomputation. Although memoization
 117 does not change the semantics of a deterministic program, it does change that of a stochastic
 118 program (Goodman et al., 2008). The authors provide an intuitive example: let f be a function
 119 that flips a coin and return “head” or “tails”. The probability that two calls of f are equivalent is
 120 0.5. However, if the function call is memoized, it is 1. In fact, there is an infinite range of possible
 121 caching policies (specifications of when to use a stored value and when to recompute), each potentially
 122 having a different semantics. Any particular caching policy can be understood by random
 123 world semantics (Poole, 1993; Sato, 1995) over the stochastic program: each possible world corre-
 124 sponds to a mapping from function input sequence to function output sequence (McAllester et al.,
 125 2008). In Venture, these possible worlds are first-class objects, and correspond to the *probabilistic*
 126 *execution traces* (Mansinghka et al., 2014).

127 To transfer this idea to probabilistic programming, we now introduce a language construct called a
 128 *statistical memoizer*. Suppose we have a function f which can be evaluated but we wish to learn
 129 about the behavior of f using as few evaluations as possible. The statistical memoizer, which here
 130 we give the name `gpmem`, was motivated by this purpose. It produces two outputs:
 131

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

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

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

136 Different implementations of the statistical memoizer can have different prior distributions $P(f)$; in
 137 this paper, we deploy a Gaussian process prior (implemented as `gpmem` below). Note that we require
 138 the ability to sample f_{emu} jointly at multiple inputs because the values of $f(x_1), \dots, f(x_k)$ will in
 139 general be dependent.
 140

141 In Venture, we initialize `gpmem` as follows:

```
142
143     assume K = make_squaredexp (sf, 1)
144     assume (f_compute f_emu) = gpmem( f, K))
```

145 Adding a single line to the program, such as

```
146
147     infer mh( quote( parameters), one, 50),
```

148 allows us perform Bayesian inference over the parameters sf and l . Such inference can be performed
 149 without the refactoring and elaboration needed if one would describe the above declaratively as in
 150 traditional statistics notation. In contrast to traditional statistics notation, probabilistic programs
 151 are written procedurally; reasoning declaratively about the dynamics of a fundamentally procedural
 152 inference algorithm is often unwieldy due to the absence of programming constructs such as loops
 153 and mutable state.
 154

155 We implement `gpmem` by memoizing a target procedure in a wrapper that remembers previously
 156 computed values. This comes with interesting implications: from the standpoint of computation,
 157 a data set of the form $\{(x_i, y_i)\}$ can be thought of as a function $y = f_{\text{look-up}}(x)$, where $f_{\text{look-up}}$ is
 158 restricted to only allow evaluation at a specific set of inputs x .

159 Modelling the data set with a GP then amounts to trying to learn a smooth function f_{emu} (“emu”
 160 stands for “emulator”) which extends f to its full domain. We can then incorporate observations
 161 in two different ways: we either are either told that the at a point x the value is y :

```
162     observe f_emu ( x ) = y
```

163
164 Or we express this as
165

```
166     predict f_compute ( x )
```

167
168 The second expression has at least two benefits: (i) readability (in some cases), and (ii) amenability
169 to active learning. As to (i), the statistical code of creating a Gaussian process is replaced with a
170 memoization-like idiom, which will be more familiar to programmers. As to (ii), when using gpmem,
171 it is quite easy to decide incrementally which data point to sample next: for example, the loop from
172 $x[1]$ to $x[n]$ could be replaced by a loop in which the next index i is chosen by a supplied decision
173 rule. In this way, we could use gpmem to perform online learning using only a subset of the available
174 data.

175 We illustrate the use of gpmem in this context in a tutorial in Fig. 1.
176

177 3 Applications 178

179 gpmem is an elegant linguistic framework for function learning-related tasks. The technique allows
180 language constructs from programming to help express models which would be cumbersome to
181 express in statistics notation. We will now illustrate this with three example applications.
182

183 3.1 Nonlinear regression in the presence of outliers 184

185 The probability of the hyper-parameters of a GP as defined above and given covariance function
186 structure \mathbf{K} is:

$$187 P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) = \frac{P(\mathbf{D} | \boldsymbol{\theta}, \mathbf{K})P(\boldsymbol{\theta} | \mathbf{K})}{P(\mathbf{D} | \mathbf{K})}. \quad (2)$$

188

189 Let the \mathbf{K} be the sum of a smoothing and a white noise (WN) kernel. For this case, Neal (1997)
190 suggested the problem of outliers in data as a use-case for a hierarchical Bayesian treatment of
191 Gaussian processes¹. The work suggests a hierarchical system of hyper-parameterization. Here, we
192 draw hyper-parameters from a Γ distributions:

$$193 \ell^{(t)} \sim \Gamma(\alpha_1, \beta_1), \sigma^{(t)} \sim \Gamma(\alpha_2, \beta_2) \quad (3)$$

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

$$195 \alpha_1^{(t)} \sim \Gamma(\alpha_\alpha^1, \beta_\alpha^1), \alpha_2^{(t)} \sim \Gamma(\alpha_\alpha^2, \beta_\alpha^2), \dots \quad (4)$$

196

197 One can represent this kind of model using gpmem (Fig. 2). Neal provides a custom inference
198 algorithm setting and evaluates it using the following synthetic data problem. Let f be the underlying
199 function that generates the data:

$$200 f(x) = 0.3 + 0.4x + 0.5 \sin(2.7x) + \frac{1.1}{(1+x^2)} + \eta \quad \text{with } \eta \sim \mathcal{N}(0, \sigma) \quad (5)$$

201

202 We synthetically generate outliers by setting $\sigma = 0.1$ in 95% of the cases and to $\sigma = 1$ in the
203 remaining cases. gpmem can capture the true underlying function within only 100 MH steps on
204 the hyper-parameters to get a good approximation for their posterior. Note that Neal devises an
205 additional noise model and performs a large number of Hybrid-Monte Carlo and Gibbs steps.

206 3.2 Discovering qualitative structure from time series data 207

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

211 With gpmem we can provide a fully Bayesian treatment of this, previously unavailable, using a stochastic
212 grammar (see Fig. 3).

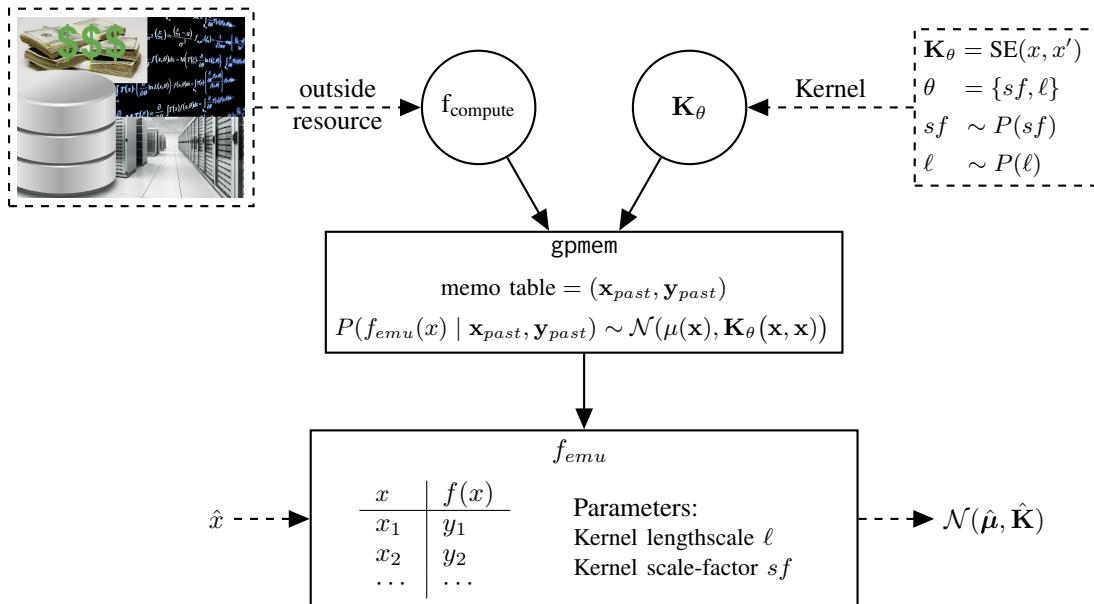
213
214 ¹In (Neal, 1997) the sum of an SE plus a constant kernel is used. We keep the WN kernel for illustrative
215 purposes.

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

```

216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269

```



```

define f = proc( x ) {
    exp(-0.1*abs(x-2))) *
    10* cos(0.4*x) + 0.2
}
assume (f_compute f_emu) = gpmem( f, K )
sample f_emu( array( -20, ..., 20))

predict f_compute( 12.6)
sample f_emu( array( -20, ..., 20))

predict f_compute( -6.4)
sample f_emu( array( -20, ..., 20))

observe f_emu( -3.1) = 2.60
observe f_emu( 7.8) = -7.60
observe f_emu( 0.0) = 10.19

sample f_emu( array( -20, ..., 20))

infer mh(quote(hyper-parameter), one, 50)

sample f_emu( array( -20, ..., 20))

```

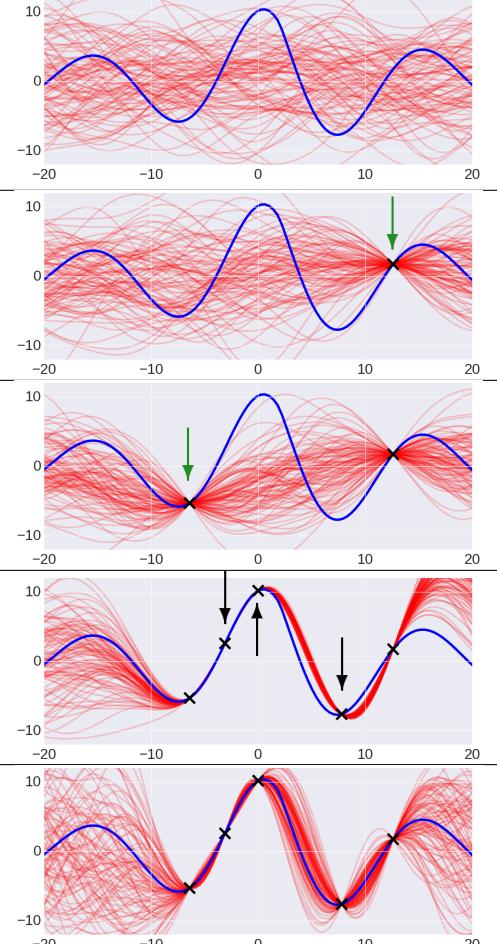


Figure 1: gpmem tutorial. The top shows a schematic of gpmem. f_{compute} probes an outside resource. This can be expensive (top left). Every probe is memoized and improves the GP-based emulator. Below the schematic we see a movie of the evolution of gpmem's state of believe of the world given certain Venture directives.

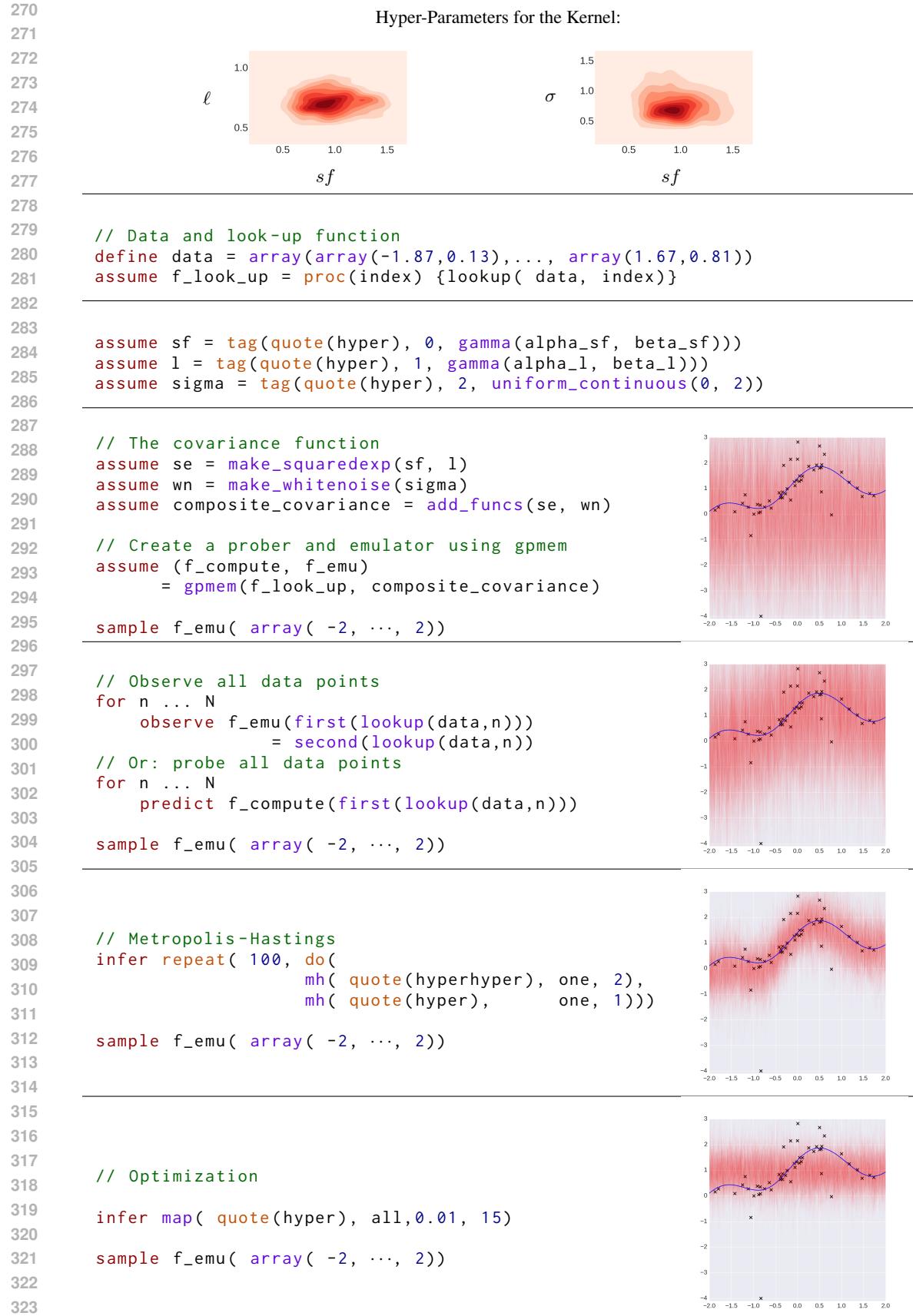
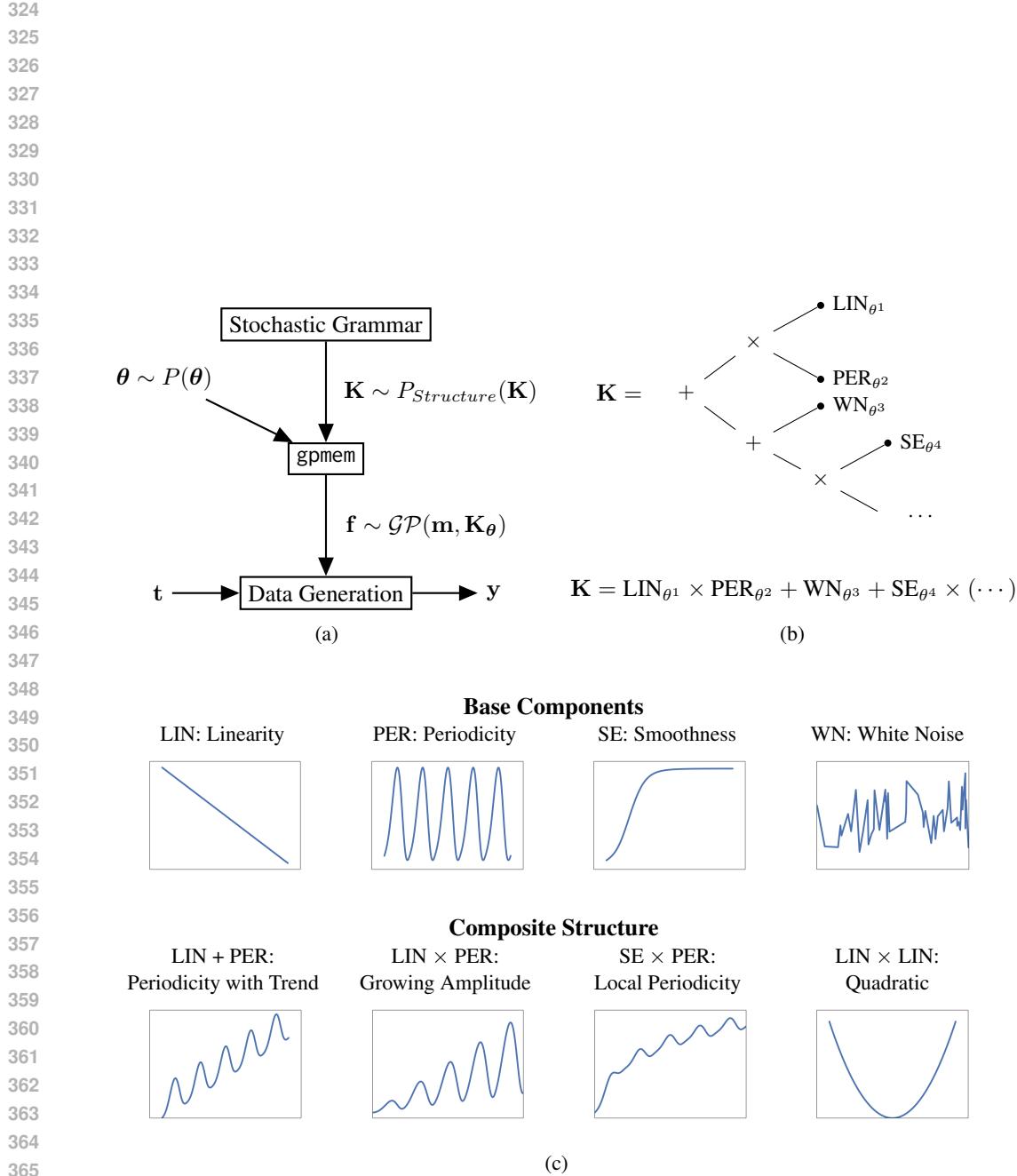


Figure 2: Regression with outliers and hierarchical prior structure.



366 Figure 3: (a) Graphical description of Bayesian GP structure learning. (b) Composite structure. (c)
 367 The natural language interpretation of the structure.
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377

378 We deploy a probabilistic context free grammar for our prior on structures. An input of non-
 379 composite kernels (base kernels) is supplied to generate a posterior distributions of composite struc-
 380 ture to express local and global aspects of the data.
 381

382 We approximate the following intractable integrals of the expectation for the prediction:

$$383 \quad \mathbb{E}[y^* | x^*, \mathbf{D}, \mathbf{K}] = \iint f(x^*, \boldsymbol{\theta}, \mathbf{K}) P(\boldsymbol{\theta} | \mathbf{D}, \mathbf{K}) P(\mathbf{K} | \boldsymbol{\Omega}, s, n) d\boldsymbol{\theta} d\mathbf{K}. \quad (6)$$

386 This is done by sampling from the posterior probability distribution of the hyper-parameters and the
 387 possible kernel:
 388

$$389 \quad y^* \approx \frac{1}{T} \sum_{t=1}^T f(x^* | \boldsymbol{\theta}^{(t)}, \mathbf{K}^{(t)}). \quad (7)$$

391 In order to provide the sampling of the kernel, we introduce a stochastic process that simulates the
 392 grammar for algebraic expressions of covariance function algebra:
 393

$$394 \quad \mathbf{K}^{(t)} \sim P(\mathbf{K} | \boldsymbol{\Omega}, s, n) \quad (8)$$

395 Here, we start with the set of given base kernels and draw a random subset. For this subset of size
 396 n , we sample a set of possible operators $\boldsymbol{\Omega}$ combining base kernels. The marginal probability of a
 397 composite structure

$$398 \quad P(\mathbf{K} | \boldsymbol{\Omega}, s, n) = P(\boldsymbol{\Omega} | s, n) \times P(s | n) \times P(n), \quad (9)$$

400 is characterized by the prior $P(n)$ on the number of base kernels used, the probability of a uniformly
 401 chosen subset of the set of n possible covariance functions

$$403 \quad P(s | n) = \frac{n!}{|s|!}, \quad (10)$$

405 and the probability of sampling a global or a local structure, which is given by a binomial distribu-
 406 tion:
 407

$$408 \quad P(\boldsymbol{\Omega} | s, n) = \binom{n}{r} p_{+ \times}^k (1 - p_{+ \times})^{n-k}. \quad (11)$$

409 Many equivalent covariance structures can be sampled due to covariance function algebra and equiv-
 410 alent representations with different parameterization (Lloyd et al., 2014). To inspect the posterior
 411 of these equivalent structures we convert each kernel expression into a sum of products and subse-
 412 quently simplify. All base kernels can be found in Appendix A, rules for this simplification can be
 413 found in appendix B. The code for learning of kernel structure is as follows:
 414

415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431

```

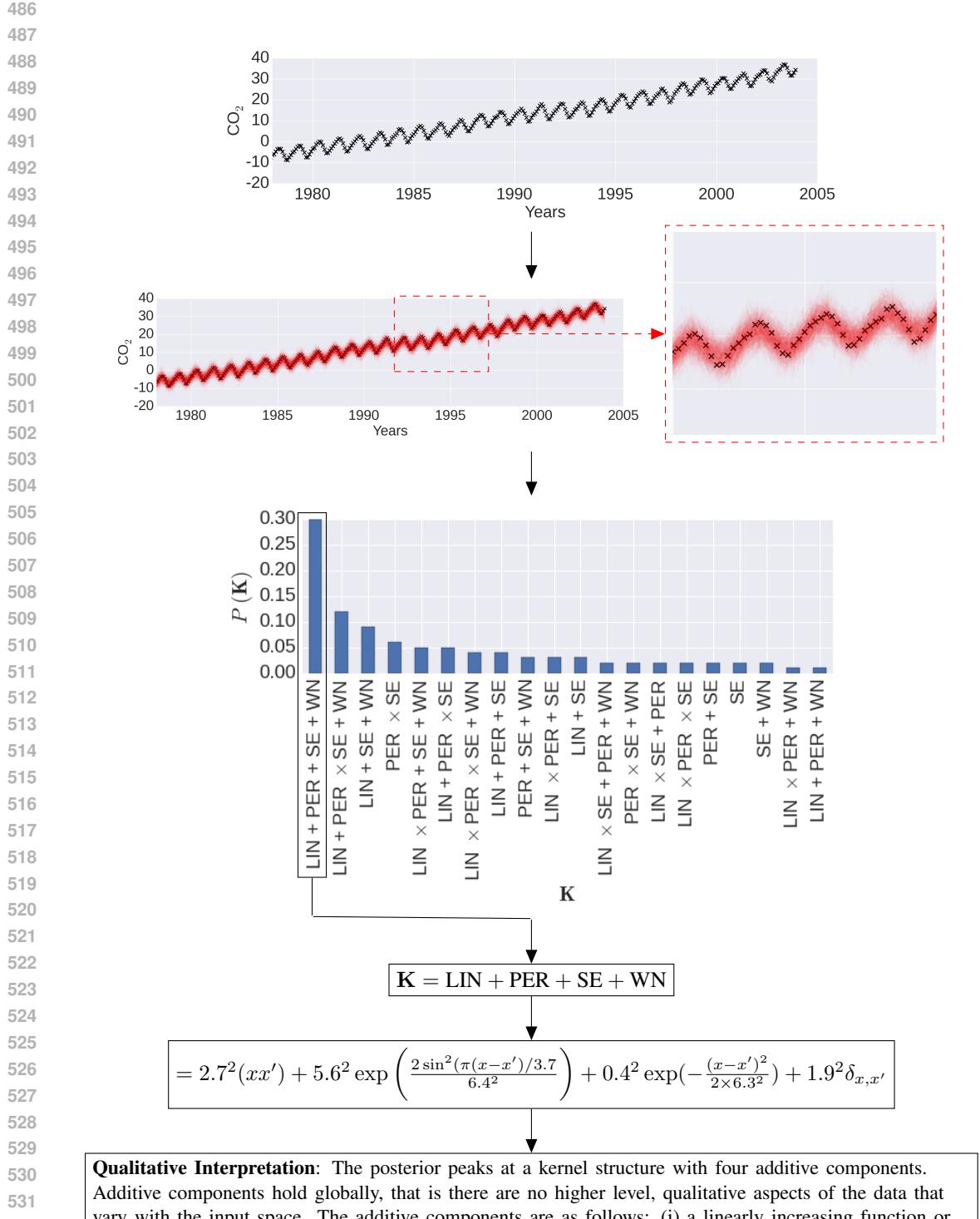
432
433 // GRAMMAR FOR KERNEL STRUCTURE
434 1 assume kernels = list(se, wn, lin, per, rq) // defined as above
435
436 // prior on the number of kernels
437 2 assume p_number_k = uniform_structure(n)
438 3 assume subset_kernels = tag(quote(grammar), 0,
439                                subset(kernels, p_number_k))
440
441 // kernel composition
442 5 assume composition = proc(l) {
443 6   if (size(l) <= 1)
444 7     { first(l) }
445 8   else { if (bernoulli())
446 9     { add_funcs(first(l), composition(rest(l))) }
44710    else { mult_funcs(first(l), composition(rest(l))) }
44811  }
44912  }
450
45113 assume K = tag(quote(grammar), 1, composition(subset_kernels))
452
45314 assume (f_compute f_emu) = gpmem(f_look_up, K)
454
455 // Probe all data points
45615 for n ... N
45716   predict f_compute(get_data_xs(n))
458
459 // PERFORMING INFERENCE
46017 infer repeat(2000, do(
46118   mh(quote(grammar), one, 1),
46219     for kernel in K:
46320       mh(quote(parameters_kernel), one, 1)))
464
465
466
467
468
469
470
```

We defined the space of covariance structures in a way that allows us to produce results coherent with work presented in Automatic Statistician. For example, for the airline data set describing monthly totals of international airline passengers (Box et al., 1997, according to Duvenaud et al., 2013). Our most frequent sample is identical with the highest scoring result reported in previous work using a search-and-score method (Duvenaud et al., 2013) for the CO₂ data set (see Rasmussen and Williams, 2006 for a description) and the predictive capability is comparable. However, the components factor in a different way due to different parameterization of the individual base kernels. We see that the most probable alternatives for a structural description both recover the data dynamics (Fig. ?? for the airline data set).

Confident about our results, we can now query the data for certain structures being present. We illustrate this using the Mauna Loa data used in previous work on automated kernel discovery (Duvenaud et al., 2013). We assume a relatively simple hypothesis space consisting of only four kernels, a linear, a smoothing, a periodic and a white noise kernel. In this experiment, we resort to the white noise kernel instead RQ (similar to (Lloyd et al., 2014)). We can now run the algorithm, compute a posterior of structures (see Fig. 4 and 5). We can also query this posterior distribution for the marginal of certain simple structures to occur. We demonstrate this in Fig. 6

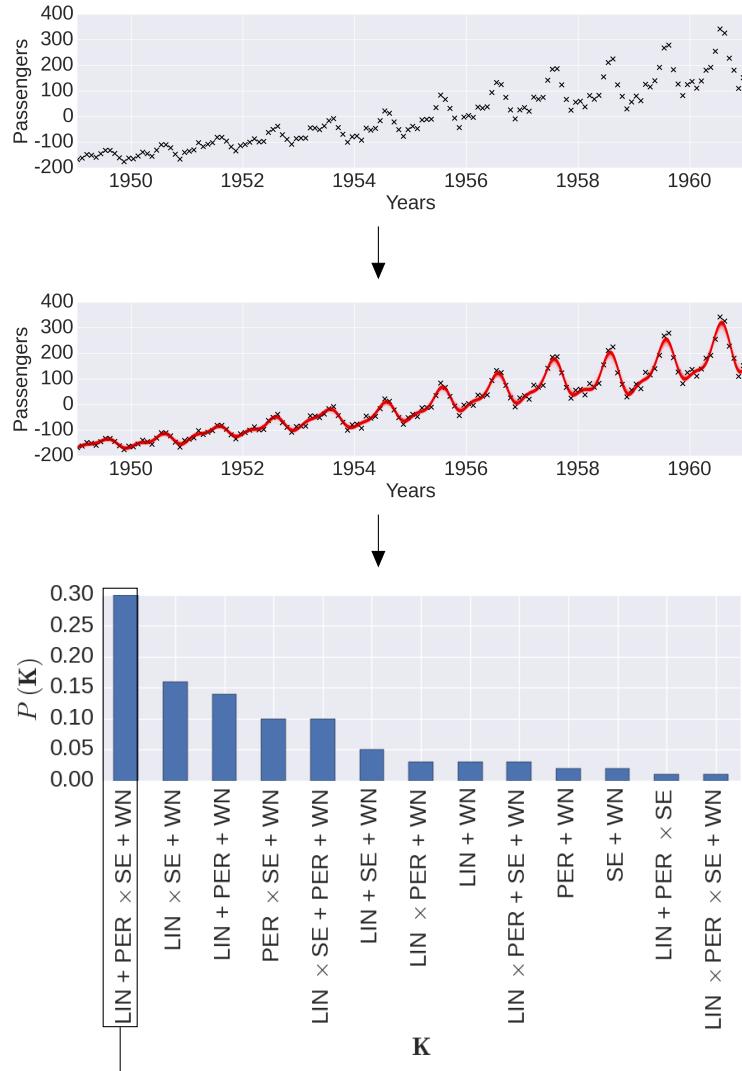
3.3 Bayesian optimization

We introduce Thompson sampling, the basic solution strategy underlying the Bayesian optimization with gpmem. Thompson sampling (Thompson 1933) is a widely used Bayesian framework for addressing the trade-off between exploration and exploitation in multi-armed (or continuum-armed) bandit problems. We cast the multi-armed bandit problem as a one-state Markov decision process, and describe how Thompson sampling can be used to choose actions for that Markov Decision Processes (MDP).



534
535
536
537
538
539

Figure 4: Posterior of structure and qualitative, human interpretable reading. We take the raw data (top), compute a posterior distribution on structures (red samples and bar plot). We take the peak of this distribution ($\text{LIN} + \text{PER} + \text{SE} + \text{WN}$) with the sampled parameters used to generate the samples for the second plot from the top and write it in functional form with parameters. We depict the human readable interpretation of the equation on the bottom.



$$K = \text{LIN} + \text{SE} \times \text{PER} + \text{WN}$$

$$= 7.47^2(xx') + \left(0.27^2 \exp\left(-\frac{(x-x')^2}{2 \times 4.63^2}\right) \times 7.34^2 \exp\left(\frac{2 \sin^2(\pi(x-x')/4.4)}{4.55^2}\right) \right) + 2.93^2 \delta_{x,x'}$$

Qualitative Interpretation: The posterior peaks at a kernel structure with three additive components. Additive components hold globally, that is there are no higher level, qualitative aspects of the data that vary with the input space. The additive components are as follows: (i) a linearly increasing function or trend; (ii) a approximate periodic function; and (iv) white noise.

Figure 5: Posterior of structure and qualitative, human interpretable reading. We take the raw data (top), compute a posterior distribution on structures (red samples and bar plot). We take the peak of this distribution ($\text{LIN} + \text{PER} \times \text{SE} + \text{WN}$) with the sampled parameters used to generate the samples for the second plot from the top and write it in functional form with parameters. We depict the human readable interpretation of the equation on the bottom.

594
595
596
597
598
599

600 $P(\mathbf{K}) \approx \frac{1}{N} \sum_{n=1}^N f(\mathbf{K}_n)$ where $f(\mathbf{K}_n) = \begin{cases} 1, & \text{if } \mathbf{K}_{global} \in \mathbf{K}_n, \\ 0, & \text{otherwise.} \end{cases}$



601 $P(\mathbf{K}_a \wedge \mathbf{K}_b) \approx \frac{1}{N} \sum_{n=1}^N f(\mathbf{K}_n)$ where $f(\mathbf{K}_n) = \begin{cases} 1, & \text{if } \mathbf{K}_a \text{ and } \mathbf{K}_b \in \mathbf{K}_n, \\ 0, & \text{otherwise.} \end{cases}$

602 $P(\mathbf{K}_a \vee \mathbf{K}_b) = P(\mathbf{K}_a) + P(\mathbf{K}_b) - P(\mathbf{K}_a \wedge \mathbf{K}_b)$

603
604
605
606
607
608

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

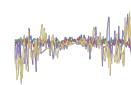
610 $P((\text{LIN} \vee \text{LIN} \times \text{SE}) \wedge (\text{PER} \vee \text{PER} \times \text{SE} \vee \text{PER} \times \text{LIN}) \wedge (\text{WN} \vee \text{LIN} \times \text{WN})) = 0.36$

611
612
613

614 Is there a trend?
615 $P(\text{LIN} \vee \text{LIN} \times \text{SE}) = 0.65$



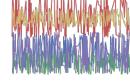
616 Is there noise?
617 $P(\text{WN} \vee \text{LIN} \times \text{WN}) = 0.75$



618 A linear trend?
619 $P(\text{LIN}) = 0.63$
620 A smooth trend?
621 $P(\text{LIN} \times \text{SE}) = 0.02$

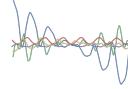


622 Heteroskedastic noise?
623 $P(\text{LIN} \times \text{WN}) = 0$
624 White noise?
625 $P(\text{WN}) = 0.75$



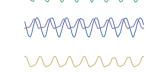
626
627
628

629 Is there repeating structure?
630 $P(\text{PER} \vee \text{PER} \times \text{SE} \vee \text{PER} \times \text{LIN}) = 0.73$



631
632
633

634 PER = 0.32



635 PER × SE = 0.34



636 PER × LIN = 0.07



637
638
639

640 Figure 6: We can query the data if some logical statements are probable to be true, for example, is it
641 true that there is a trend?

642
643
644
645
646
647

648
649 Here, an agent is to take a sequence of actions a_1, a_2, \dots from a (possibly infinite) set of possible
650 actions \mathcal{A} . After each action, a reward $r \in \mathbb{R}$ is received, according to an unknown conditional
651 distribution $P_{\text{true}}(r | a)$. The agent's goal is to maximize the total reward received for all actions in
652 an online manner. In Thompson sampling, the agent accomplishes this by placing a prior distribution
653 $P(\vartheta)$ on the possible "contexts" $\vartheta \in \Theta$. Here a context is a believed model of the conditional
654 distributions $\{P(r | a)\}_{a \in \mathcal{A}}$, or at least, a believed statistic of these conditional distributions which
655 is sufficient for deciding an action a . If actions are chosen so as to maximize expected reward, then
656 one such sufficient statistic is the believed conditional mean $V(a | \vartheta) = \mathbb{E}[r | a; \vartheta]$, which can be
657 viewed as a believed value function. For consistency with what follows, we will assume our context
658 ϑ takes the form $(\theta, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$ where \mathbf{a}_{past} is the vector of past actions, \mathbf{r}_{past} is the vector of their
659 rewards, and θ (the "semicontext") contains any other information that is included in the context.
660

In this setup, Thompson sampling has the following steps:

661 **Algorithm 1** Thompson sampling.

663 Repeat as long as desired:

- 664 1. **Sample.** Sample a semicontext $\theta \sim P(\theta)$.
 - 665 2. **Search (and act).** Choose an action $a \in \mathcal{A}$ which (approximately) maximizes $V(a | \vartheta) =$
666 $\mathbb{E}[r | a; \vartheta] = \mathbb{E}[r | a; \theta, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}]$.
 - 667 3. **Update.** Let r_{true} be the reward received for action a . Update the believed distribution on
668 θ , i.e., $P(\theta) \leftarrow P_{\text{new}}(\theta)$ where $P_{\text{new}}(\theta) = P(\theta | a \mapsto r_{\text{true}})$.
-

672 Note that when $\mathbb{E}[r | a; \vartheta]$ (under the sampled value of θ for some points a) is far from the true value
673 $\mathbb{E}_{P_{\text{true}}}[r | a]$, the chosen action a may be far from optimal, but the information gained by probing
674 action a will improve the belief ϑ . This amounts to "exploration." When $\mathbb{E}[r | a; \vartheta]$ is close to the
675 true value except at points a for which $\mathbb{E}[r | a; \vartheta]$ is low, exploration will be less likely to occur, but
676 the chosen actions a will tend to receive high rewards. This amounts to "exploitation." The trade-off
677 between exploration and exploitation is illustrated in Figure 7. Roughly speaking, exploration will
678 happen until the context ϑ is reasonably sure that the unexplored actions are probably not optimal,
679 at which time the Thompson sampler will exploit by choosing actions in regions it knows to have
680 high value.

681 Typically, when Thompson sampling is implemented, the search over contexts $\vartheta \in \Theta$ is limited
682 by the choice of representation. In traditional programming environments, θ often consists of a few
683 numerical parameters for a family of distributions of a fixed functional form. With work, a mixture of
684 a few functional forms is possible; but without probabilistic programming machinery, implementing
685 a rich context space Θ would be an unworkably large technical burden. In a probabilistic programming
686 language, however, the representation of heterogeneously structured or infinite-dimensional context
687 spaces is quite natural. Any computable model of the conditional distributions $\{P(r | a)\}_{a \in \mathcal{A}}$ can
688 be represented as a stochastic procedure $(\lambda(a) \dots)$. Thus, for computational Thompson sampling,
689 the most general context space $\hat{\Theta}$ is the space of program texts. Any other context space Θ has a
690 natural embedding as a subset of $\hat{\Theta}$.

691 **A Mathematical Specification**

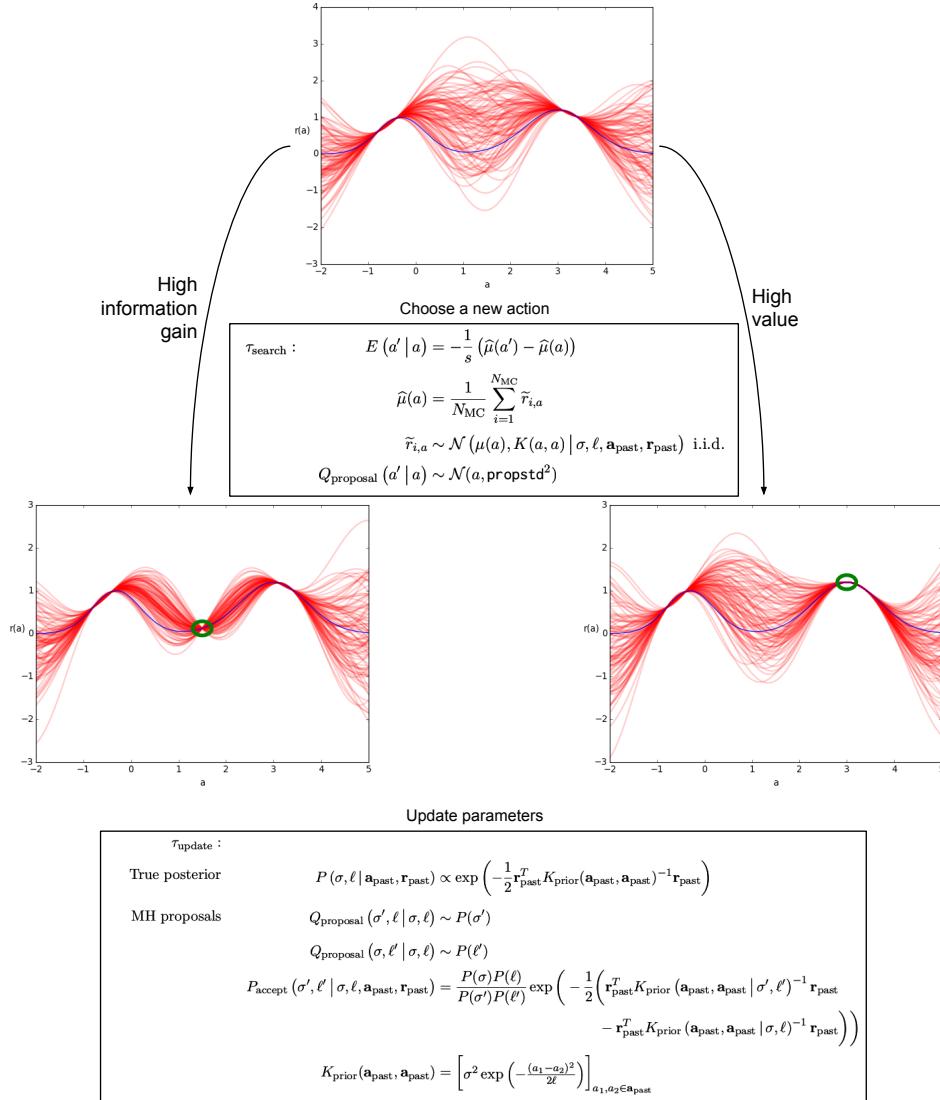
692 Our mathematical specification assert the following properties:

- 693 • The regression function has a Gaussian process prior.
- 694 • The actions $a_1, a_2, \dots \in \mathcal{A}$ are chosen by a Metropolis-like search strategy with Gaussian
695 drift proposals.
- 696 • The hyperparameters of the Gaussian process are inferred using Metropolis–Hastings sam-
697 pling after each action.

698 In this version of Thompson sampling, the contexts ϑ are Gaussian processes over the action space
699 $\mathcal{A} = [-20, 20] \subseteq \mathbb{R}$. That is,

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

702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724



745
746
747
748
749
750
751
752
753
754
755

Figure 7: 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 3.3.

756 where the mean μ is a computable function $\mathcal{A} \rightarrow \mathbb{R}$ and the covariance K is a computable (symmetric, positive-semidefinite) function $\mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$. This represents a Gaussian process $\{R_a\}_{a \in \mathcal{A}}$,
 757 where R_a represents the reward for action a . Computationally, we represent a context as a data
 758 structure
 759

$$760 \quad \vartheta = (\theta, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}) = (\mu_{\text{prior}}, K_{\text{prior}}, \eta, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}),$$

761 where μ_{prior} is a procedure to be used as the prior mean function. w.l.o.g. we set $\mu_{\text{prior}} \equiv 0$. K_{prior} is
 762 a procedure to be used as the prior covariance function, parameterized by η .
 763

764 The posterior mean and covariance for such a context ϑ are gotten by the usual conditioning formulas
 765 (assuming, for ease of exposition as above, that the prior mean is zero):³
 766

$$\begin{aligned} 767 \quad \mu(\mathbf{a}) &= \mu(\mathbf{a} | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}) \\ 768 \quad &= K_{\text{prior}}(\mathbf{a}, \mathbf{a}_{\text{past}}) K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}})^{-1} \mathbf{r}_{\text{past}} \\ 769 \quad K(\mathbf{a}, \mathbf{a}) &= K(\mathbf{a}, \mathbf{a} | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}}) \\ 770 \quad &= K_{\text{prior}}(\mathbf{a}, \mathbf{a}) - K_{\text{prior}}(\mathbf{a}, \mathbf{a}_{\text{past}}) K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}})^{-1} K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}). \end{aligned}$$

771 Note that the context space Θ is not a finite-dimensional parametric family, since the vectors
 772 \mathbf{a}_{past} and \mathbf{r}_{past} grow as more samples are taken. Θ is, however, representable as a computational
 773 procedure together with parameters and past samples, as we do in the representation $\vartheta =$
 774 $(\mu_{\text{prior}}, K_{\text{prior}}, \eta, \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$.
 775

776 We combine the Update and Sample steps of Algorithm 1 by running a Metropolis–Hastings (MH)
 777 sampler whose stationary distribution is the posterior $P(\theta | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$. The functional forms of
 778 μ_{prior} and K_{prior} are fixed in our case, so inference is only done over the parameters $\eta = \{\sigma, \ell\}$; hence
 779 we equivalently write $P(\sigma, \ell | \mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}})$ for the stationary distribution. We make MH proposals to
 780 one variable at a time, using the prior as proposal distribution:
 781

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

782 and
 783

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

784 The MH acceptance probability for such a proposal is
 785

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

786 Because the priors on σ and ℓ are uniform in our case, the term involving Q_{proposal} equals 1 and we
 787 have simply
 788

$$\begin{aligned} 789 \quad P_{\text{accept}}(\sigma', \ell' | \sigma, \ell) &= \min \left\{ 1, \frac{P(\mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}} | \sigma', \ell')}{P(\mathbf{a}_{\text{past}}, \mathbf{r}_{\text{past}} | \sigma, \ell)} \right\} \\ 790 \quad &= \min \left\{ 1, \exp \left(-\frac{1}{2} \left(\mathbf{r}_{\text{past}}^T K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}} | \sigma', \ell')^{-1} \mathbf{r}_{\text{past}} \right. \right. \right. \\ 791 \quad &\quad \left. \left. \left. - \mathbf{r}_{\text{past}}^T K_{\text{prior}}(\mathbf{a}_{\text{past}}, \mathbf{a}_{\text{past}} | \sigma, \ell)^{-1} \mathbf{r}_{\text{past}} \right) \right) \right\}. \end{aligned}$$

792 The proposal and acceptance/rejection process described above define a transition operator τ_{update}
 793 which is iterated a specified number of times; the resulting state of the MH Markov chain is taken
 794 as the sampled semicontext θ in Step 1 of Algorithm 1.
 795

796 For Step 2 (Search) of Thompson sampling, we explore the action space using an MH-like transition
 797 operator τ_{search} . As in MH, each iteration of τ_{search} produces a proposal which is either accepted or
 798 rejected, and the state of this Markov chain after a specified number of steps is the new action a .
 799 The Markov chain's initial state is the most recent action, and the proposal distribution is Gaussian
 800 drift:
 801

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

802 ³Here, for vectors $\mathbf{a} = (a_i)_{i=1}^n$ and $\mathbf{a}' = (a'_i)_{i=1}^{n'}$, $\mu(\mathbf{a})$ denotes the vector $(\mu(a_i))_{i=1}^n$ and $K(\mathbf{a}, \mathbf{a}')$ denotes
 803 the matrix $[K(a_i, a'_j)]_{1 \leq i \leq n, 1 \leq j \leq n'}$.

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

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

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

$$E(a' | a) = -\frac{1}{s} (\hat{\mu}(a') - \hat{\mu}(a))$$

815 where

$$\hat{\mu}(a) = \frac{1}{N_{\text{avg}}} \sum_{i=1}^{N_{\text{avg}}} \tilde{r}_{i,a}$$

816 and

$$\tilde{r}_{i,a} \sim \mathcal{N}(\mu(a), K(a, a))$$

817 and $\{\tilde{r}_{i,a}\}_{i=1}^{N_{\text{avg}}}$ are i.i.d. for a fixed a . Here the temperature parameter $s \geq 0$ and the population size
 818 N_{avg} are specified ahead of time. Proposals of estimated value higher than that of the current action
 819 are always accepted, while proposals of estimated value lower than that of the current action are
 820 accepted with a probability that decays exponentially with respect to the difference in value. The
 821 rate of the decay is determined by the temperature parameter s , where high temperature corresponds
 822 to generous acceptance probabilities. For $s = 0$, all proposals of lower value are rejected; for
 823 $s = \infty$, all proposals are accepted. For points a at which the posterior mean $\mu(a)$ is low but the
 824 posterior variance $K(a, a)$ is high, it is possible (especially when N_{avg} is small) to draw a “wild”
 825 value of $\hat{\mu}(a)$, resulting in a favorable acceptance probability.

826 Indeed, taking an action a with low estimated value but high uncertainty serves the useful function
 827 of improving the accuracy of the estimated value function at points near a (see Figure 7).^{4,5} We see
 828 a complete probabilistic program with `gpmem` implementing Bayesian optimization with Thompson
 829 Sampling below (Listing 1).

830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861

⁴At least, this is true when we use a smoothing prior covariance function such as the squared exponential.

⁵For 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 .

```

864
865
866
867
868 1 assume sf = tag(quote(hyper), 0, uniform_continuous(0, 10))
869 2 assume l = tag(quote(hyper), 1, uniform_continuous(0, 10))
870 3 assume se = make_squaredexp(sf, 1)
871 4 assume blackbox_f = get_bayesopt_blackbox()
872 5 assume (f_compute, f_emulate) = gpmem(blackbox_f, se)

873 // A naive estimate of the argmax of the given function
874 define mc_argmax = proc(func) {
875   candidate_xs = mapv(proc(i) {uniform_continuous(-20, 20)},
876                        arange(20));
877   candidate_ys = mapv(func, candidate_xs);
878   lookup(candidate_xs, argmax_of_array(candidate_ys))
879 };
880
881 // Shortcut to sample the emulator at a single point without packing
882 // and unpacking arrays
883 define emulate_pointwise = proc(x) {
884   run(sample(lookup(f_emulate(array(unquote(x))), 0)))
885 };
886
887 // Main inference loop
888 infer repeat(15, do(pass,
889   // Probe V at the point mc_argmax(emulate_pointwise)
890   predict(f_compute(unquote(mc_argmax(emulate_pointwise)))),
891   // Infer hyperparameters
892   mh(quote(hyper), one, 50)));
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

```

Listing 1: Bayesian optimization using gpmem

```

1 assume sf = tag(quote(hyper), 0, uniform_continuous(0, 10))
2 assume l = tag(quote(hyper), 1, uniform_continuous(0, 10))
3 assume se = make_squaredexp(sf, 1)
4 assume blackbox_f = get_bayesopt_blackbox()
5 assume (f_compute, f_emulate) = gpmem(blackbox_f, se)

6 // A naive estimate of the argmax of the given function
7 define mc_argmax = proc(func) {
8   candidate_xs = mapv(proc(i) {uniform_continuous(-20, 20)},
9                        arange(20));
10  candidate_ys = mapv(func, candidate_xs);
11  lookup(candidate_xs, argmax_of_array(candidate_ys))
12};

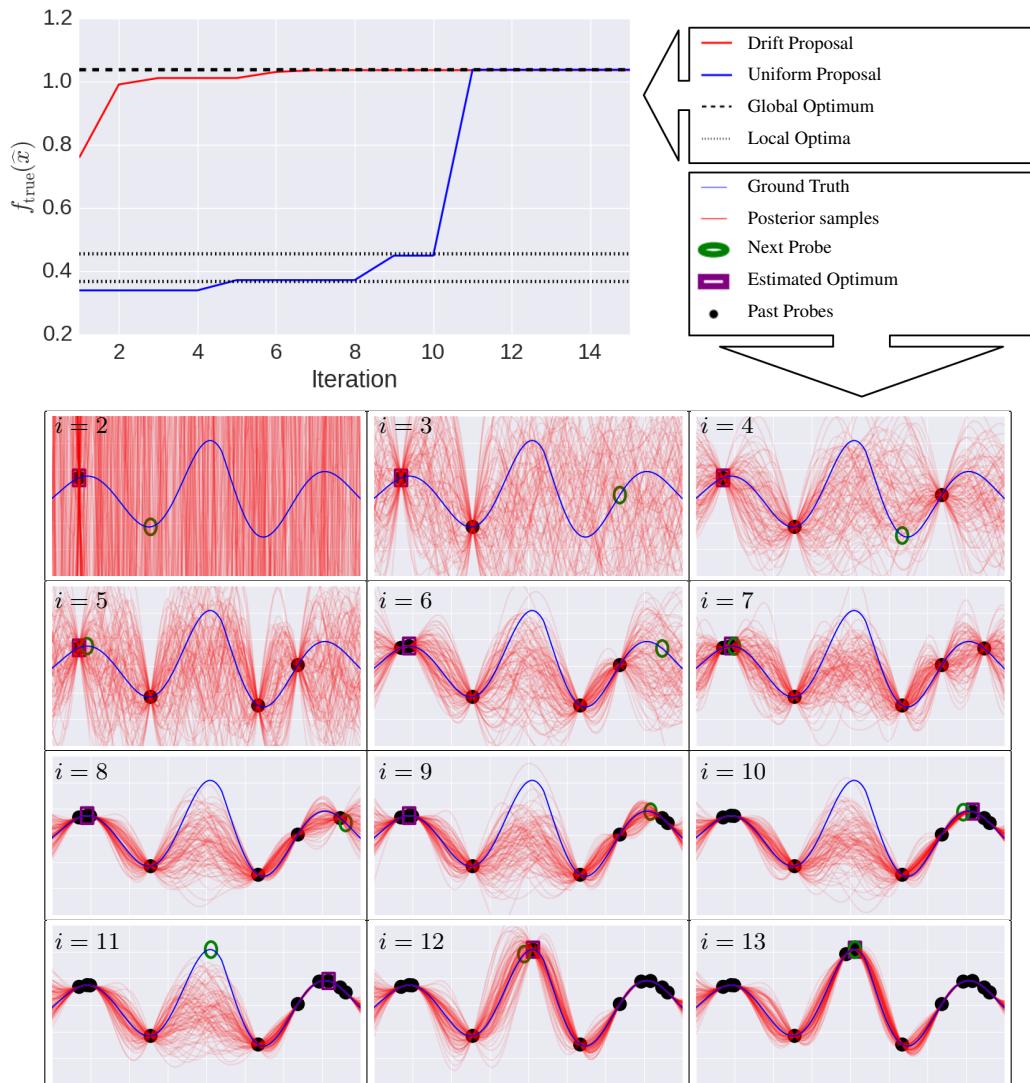
13 // Shortcut to sample the emulator at a single point without packing
14 // and unpacking arrays
15 define emulate_pointwise = proc(x) {
16   run(sample(lookup(f_emulate(array(unquote(x))), 0)))
17};

18 // Main inference loop
19 infer repeat(15, do(pass,
20   // Probe V at the point mc_argmax(emulate_pointwise)
21   predict(f_compute(unquote(mc_argmax(emulate_pointwise)))),
22   // Infer hyperparameters
23   mh(quote(hyper), one, 50)));
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
605
606
607
607
608
609
609
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
```

```

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

```



Appendix

A Covariance Functions

$$\text{SE} = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (12)$$

$$\text{LIN} = \sigma^2(xx') \quad (13)$$

$$C = \sigma^2 \quad (14)$$

$$\text{WN} = \sigma^2 \delta_{x,x'} \quad (15)$$

$$RQ = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2} \right)^{-\alpha} \quad (16)$$

$$\text{PER} = \sigma^2 \exp\left(\frac{2 \sin^2(\pi(x - x')/p)}{\ell^2}\right). \quad (17)$$

B Covariance Simplification

$$\begin{array}{ll}
 \text{SE} \times \text{SE} & \rightarrow \text{SE} \\
 \{\text{SE}, \text{PER}, \text{C}, \text{WN}\} \times \text{WN} & \rightarrow \text{WN} \\
 \text{LIN} + \text{LIN} & \rightarrow \text{LIN} \\
 \{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\} \times \text{C} & \rightarrow \{\text{SE}, \text{PER}, \text{C}, \text{WN}, \text{LIN}\}
 \end{array}$$

Rule 1 is derived as follows:

$$\begin{aligned}
\sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_c^2}\right) &= \sigma_a^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2}\right) \times \sigma_b^2 \exp\left(-\frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2}\right) \times \exp\left(-\frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2} - \frac{(x-x')^2}{2\ell_b^2}\right) \\
&= \sigma_c^2 \exp\left(-\frac{(x-x')^2}{2\ell_c^2}\right)
\end{aligned} \tag{18}$$

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

$$\sigma_a^2 \exp\left(-\frac{(x-x')^2}{2\ell_a^2}\right) \times \sigma_b \delta_{x,x'} = \sigma_a \sigma_b \delta_{x,x'} \quad (19)$$

Rule 3 is derived as follows:

$$\theta_c(x \times x') = \theta_a(x \times x') + \theta_b(x \times x') \quad (20)$$

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

1026 **References**
1027

- 1028 Barry, D. (1986). Nonparametric bayesian regression. *The Annals of Statistics*, 14(3):934–953.
1029 Box, G. E. P., Jenkins, G. M., and Reinsel, G. C. (1997). *Time series analysis: forecasting and*
1030 *control*.
1031 Damianou, A. and Lawrence, N. (2013). Deep gaussian processes. In *Proceedings of the Interna-*
1032 *tional Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 207–215.
1033 Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2013). Structure
1034 discovery in nonparametric regression through compositional kernel search. In *Proceedings of*
1035 *the International Conference on Machine Learning (ICML)*, pages 1166–1174.
1036 Ferris, B., Haehnel, D., and Fox, D. (2006). Gaussian processes for signal strength-based location
1037 estimation. In *Proceedings of the Conference on Robotics Science and Systems*. Citeseer.
1038 Gelbart, M. A., Snoek, J., and Adams, R. P. (2014). Bayesian optimization with unknown con-
1039 straints. *arXiv preprint arXiv:1403.5607*.
1040 Goodman, N. D .and Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. (2008). Church:
1041 A language for generative models. In *Proceedings of the Conference on Uncertainty in Artificial*
1042 *Intelligence (UAI)*, pages 220–229.
1043 Kemmler, M., Rodner, E., Wacker, E., and Denzler, J. (2013). One-class classification with gaussian
1044 processes. *Pattern Recognition*, 46(12):3507–3518.
1045 Kennedy, M. C. and O’Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the*
1046 *Royal Statistical Society. Series B, Statistical Methodology*, pages 425–464.
1047 Kuss, M. and Rasmussen, C. E. (2005). Assessing approximate inference for binary gaussian process
1048 classification. *The Journal of Machine Learning Research*, 6:1679–1704.
1049 Kwan, J., Bhattacharya, S., Heitmann, K., and Habib, S. (2013). Cosmic emulation: The
1050 concentration-mass relation for wcdm universes. *The Astrophysical Journal*, 768(2):123.
1051 Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2014). Automatic con-
1052 struction and natural-language description of nonparametric regression models. In *Proceedings*
1053 *of the Conference on Artificial Intelligence (AAAI)*.
1054 Mansinghka, V. K., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic pro-
1055 gramming platform with programmable inference. *arXiv preprint arXiv:1404.0099*.
1056 McAllester, D., Milch, B., and Goodman, N. D. (2008). Random-world semantics and syntactic
1057 independence for expressive languages. Technical report.
1058 Neal, R. M. (1995). *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto.
1059 Neal, R. M. (1997). Monte carlo implementation of gaussian process models for bayesian regression
1060 and classification. *arXiv preprint physics/9701026*.
1061 Poole, D. (1993). Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*,
1062 64(1):81–129.
1063 Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adap-*
1064 *tive Computation and Machine Learning)*. The MIT Press.
1065 Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *In*
1066 *Proceedings of the International Conference on Logic Programming*. Citeseer.
1067 Schneider, M. D., Knox, L., Habib, S., Heitmann, K., Higdon, D., and Nakhleh, C. (2008). Simula-
1068 tions and cosmological inference: A statistical model for power spectra means and covariances.
1069 *Physical Review D*, 78(6):063529.
1070 Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine
1071 learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2951–
1072 2959.
1073 Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view
1074 of the evidence of two samples. *Biometrika*, pages 285–294.
1075 Williams, C. K. I. and Barber, D. (1998). Bayesian classification with gaussian processes. *IEEE*
1076 *Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.