

Probabilistic Programming with Gaussian Process Memoization

Ulrich Schaechtle

Department of Computer Science
Royal Holloway, Univ. of London

Ben Zinberg

Computer Science & AI Lab
Massachusetts Institute of Technology

Alexey Radul

Computer Science & AI Lab
Massachusetts Institute of Technology

Kostas Stathis

Department of Computer Science
Royal Holloway, Univ. of London

Vikash K. Mansinghka

Computer Science & AI Lab
Massachusetts Institute of Technology

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 85-line Python library and require fewer than 20 lines of probabilistic code each.

GP Memoization: *gpmem*

Gaussian Processes

$f(\mathbf{x})$ is the multivariate Gaussian $f(\mathbf{x}) \sim \mathcal{N}(0, k(\mathbf{x}, \mathbf{x}))$, where $k(\mathbf{x}, \mathbf{x}') = \text{Cov}_f(f(\mathbf{x}), f(\mathbf{x}'))$ is the covariance function, a.k.a. kernel. The marginal likelihood can be expressed as:

$$p(f(\mathbf{x}) = \mathbf{y} | \mathbf{x}) = \int p(f(\mathbf{x}) = \mathbf{y} | f, \mathbf{x}) p(f|\mathbf{x}) df$$

A Variation on Memoization

GP memoization produces two components:

$$f \rightarrow (f_{\text{compute}}, f_{\text{emu}})$$

- is a generalization of traditional memoization;
- changes semantics of a probabilistic program;
- f_{emu} is a statistical emulator with GP prior;
- each time f_{compute} is called, an observation is incorporated into f_{emu} and predictions improve.

Bayesian GP

We show how we can use *gpmem* to reproduce Neal’s Hierarchical Bayesian GP regression [2] for data with outliers.

Regression on data set D

\Downarrow

Statistical emulation of $f_{\text{restr}}(x) = \begin{cases} D[x], & \text{if } x \text{ is a data point} \\ \text{Error}, & \text{otherwise} \end{cases}$

Listing 1: Hierarchical GP Smoothing

```
/// SETTING UP THE MODEL
assume alpha_sf = tag('hyperhyper, gamma(7, 1))
assume beta_sf = tag('hyperhyper, gamma(7, 1))
assume alpha_l = tag('hyperhyper, gamma(7, 1))
assume beta_l = tag('hyperhyper, gamma(7, 1))
// Parameters of the covariance function
assume log_sf = tag('hyper, log(gamma(alpha_sf, beta_sf)))
assume log_l = tag('hyper, log(gamma(alpha_l, beta_l)))
assume log_sigma = tag('hyper, log(uniform_continuous(0, 2)))

// The covariance function
assume se = make_squaredexp(sf, l)
assume wn = make_whitenoise(sigma)
assume composite.covariance = add_funcs(se, wn)

// PERFORMING INFERENCE
// Create a prober and emulator using gpmem
assume f_restr = get_neal_prober()
assume compute_and_emo = gpmem(f_restr, composite.covariance)

// Probe all data points
predict mapv(first(compute_and_emo), get_neal_data_xs())

// Infer hypers and hyperhypers
infer repeat(100, do(mh('hyperhyper, one, 2), mh('hyper, one, 1)))
```

Observed data is generated with the following model:

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

where $\sigma = 0.1$ with probability 0.95, $\sigma = 1$ otherwise.

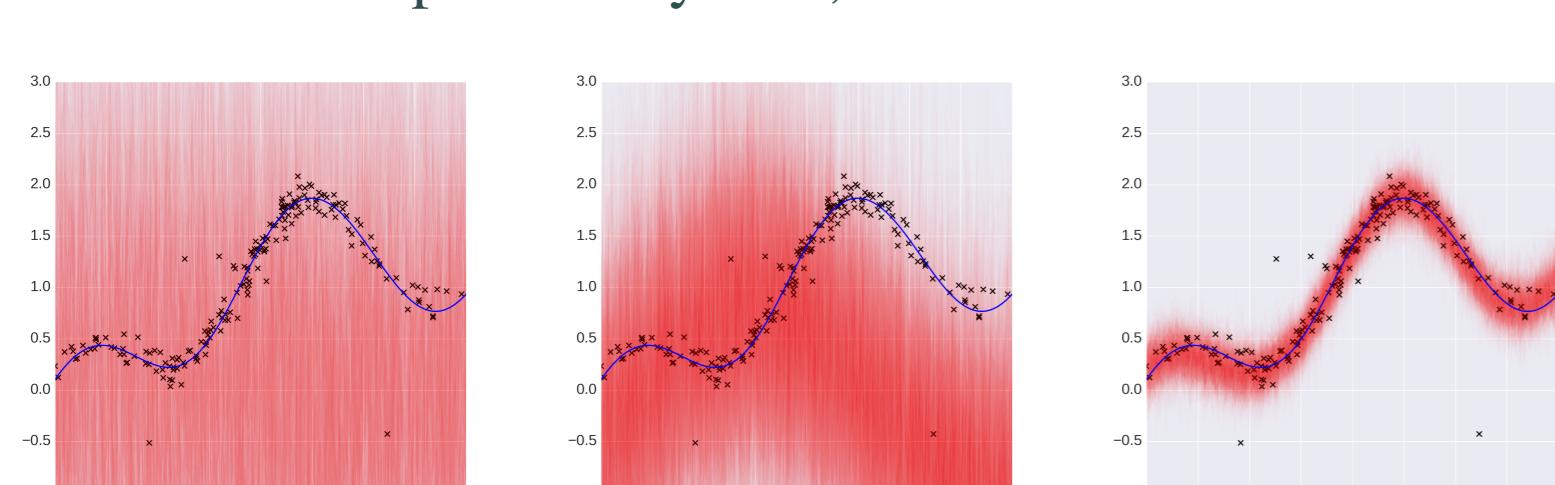


Figure 1: (left to right) predictions before data has been observed, after data has been observed, and after we inferred the hyperparameters

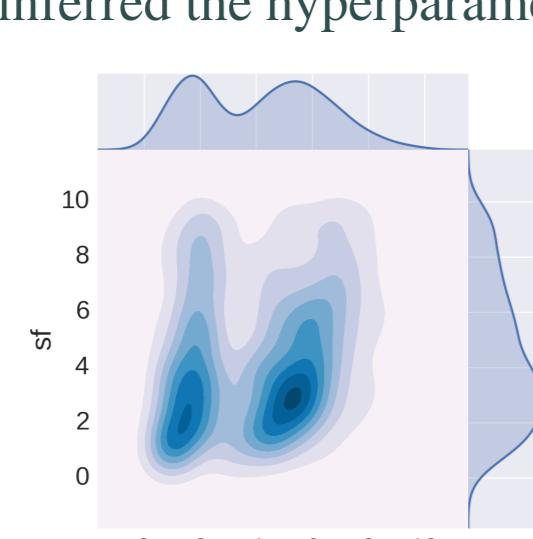


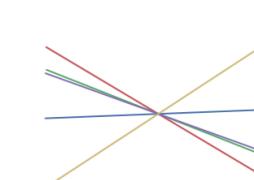
Figure 2: Contour plot and marginals of the the l and sf hyper-parameters of the above program after inference

GP Structure Learning

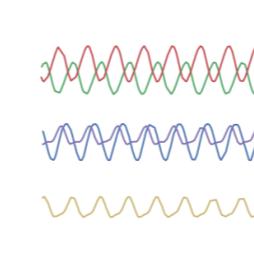
$$SE = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$



$$LIN = \theta(x x')$$



$$PER = \theta \exp\left(\frac{2 \sin^2(\pi(x-x')/\ell)}{\ell^2}\right)$$



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



Kernel Composition

$$K_{\text{new}} = K_1 + K_2 \quad \text{or} \quad K_{\text{new}} = K_1 \times K_2$$

Listing 2: Venture Code for Bayesian GP Structure Learning

```
/// GRAMMAR FOR KERNEL STRUCTURE
assume base_kernels = list(se, wn, lin, per, rq) // defined as above

// prior on the number of kernels
assume p_number_k = tag('number_kernels, uniform_structure(n))
assume s = tag('choice_subset(subset(base_kernels, p_number_k))

assume composition = tag('composition,lambda(l, // kernel composition
    if lte(size(l), 1) {
        first(l)
    } else {
        if (flip()) {
            add_funcs(first(l), cov_compo(rest(l)))
        } else {
            mult_funcs(first(l), cov_compo(rest(l)))
        }
    }
))

assume K = composition(s)

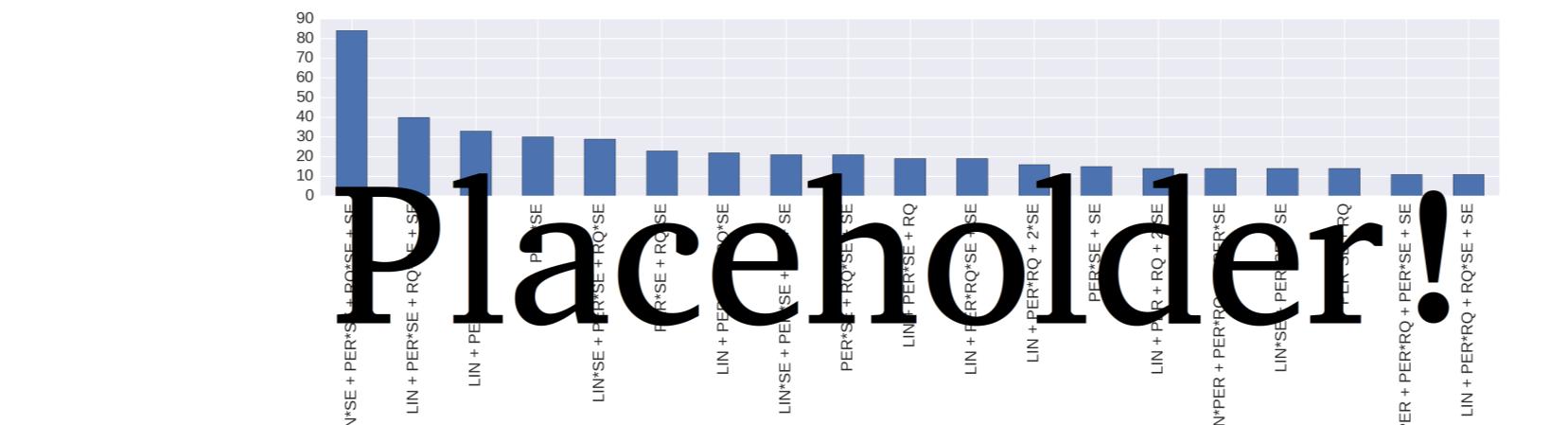
assume (f_compute f_emo) = gpmem(f_restr, K)

predict mapv(f_compute(get_data_xs)) // probe all data points

/// PERFORMING INFERENCE
infer repeat(2000, do(
    mh('number_kernel, one, 1)
    mh('choice_subset, one, 1)
    mh('composit, one, 1)
    mh('hyper, one, 10)))
```

Results

500 samples drawn from the posterior on structure for the CO2 dataset used in the Automated Statistician Project [1]:



Posterior samples for curves for the airline dataset where $SE \times LIN + PER \times LIN + RQ \times LIN$ is sampled from the structure grammar:

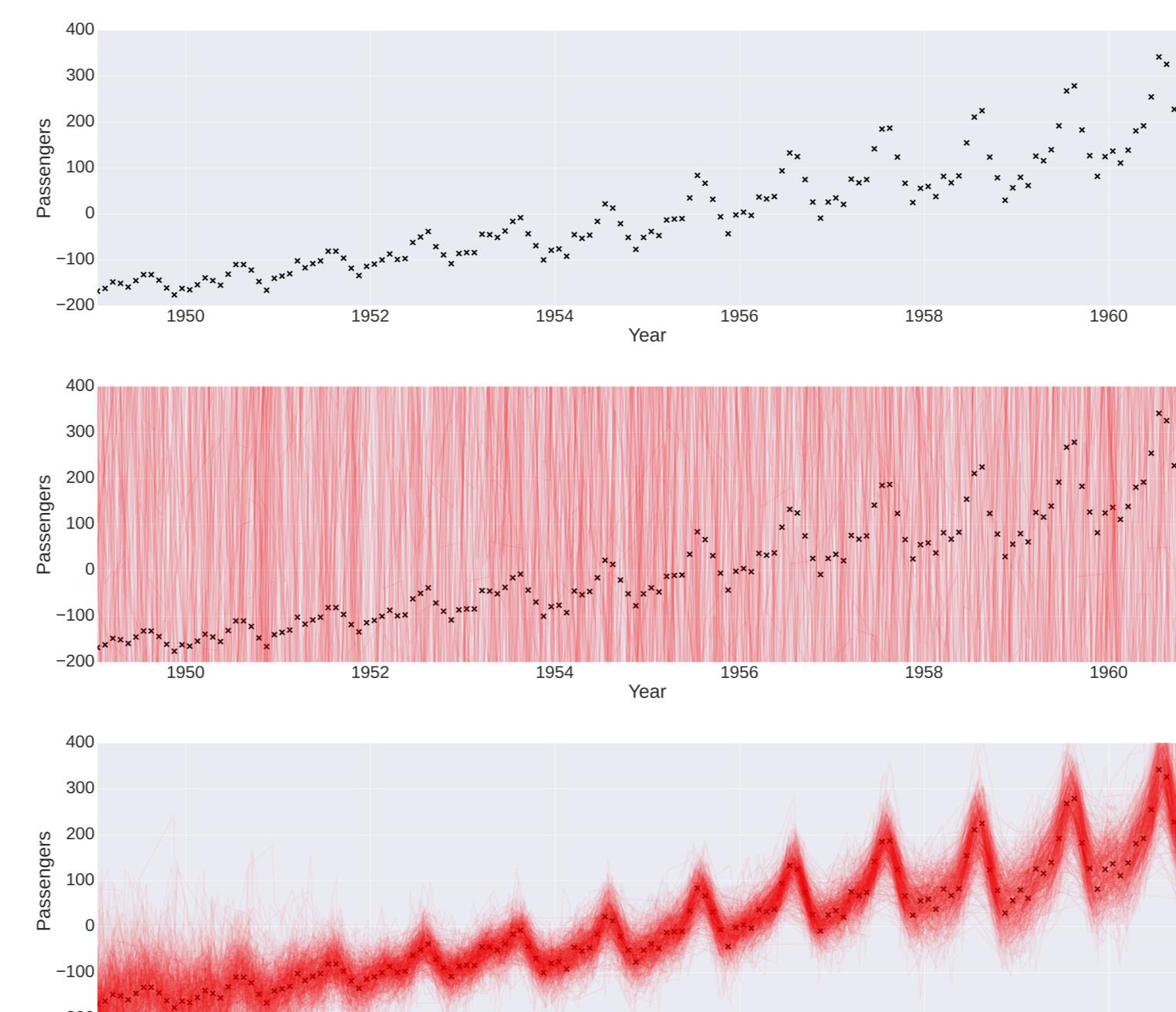
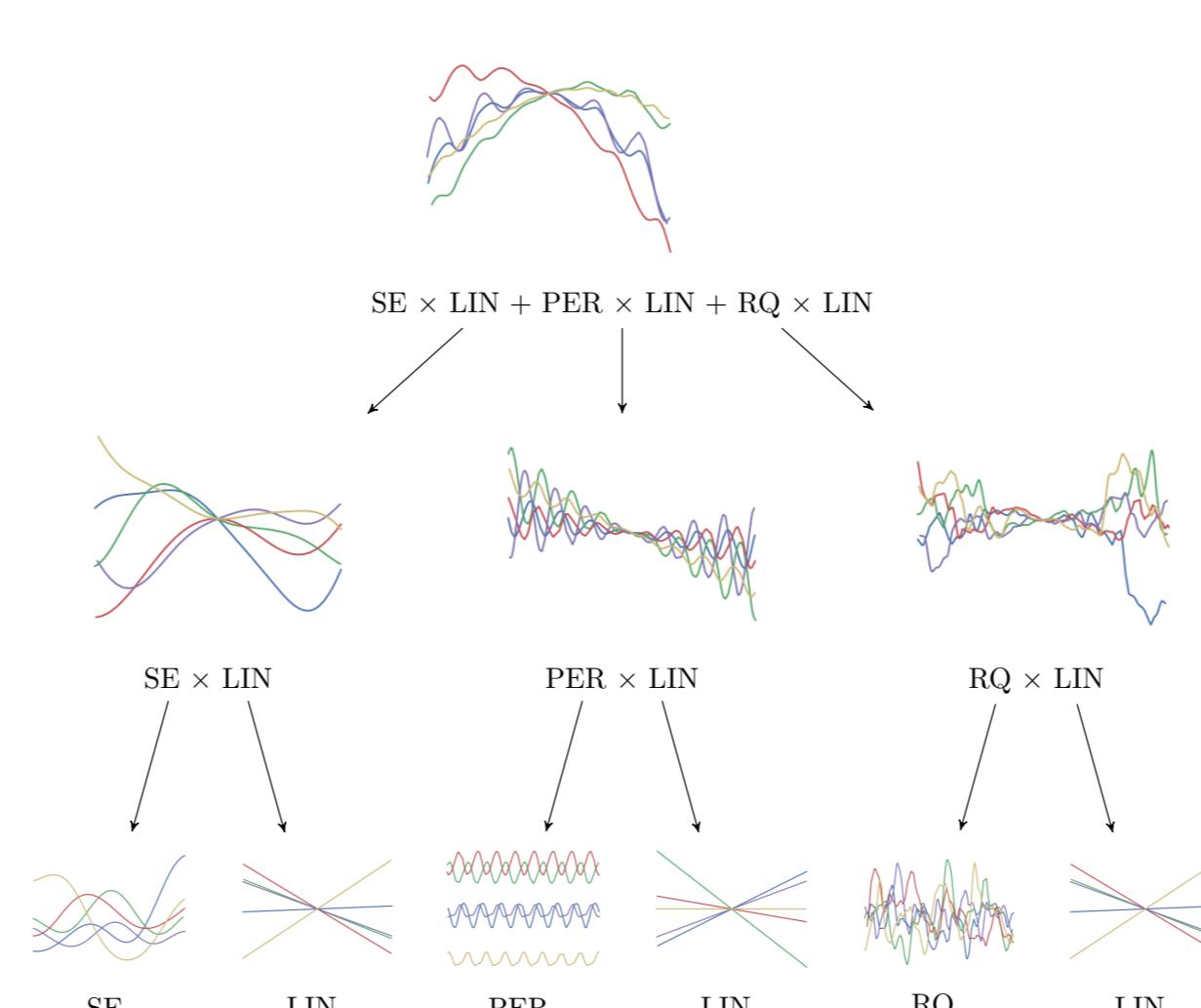


Figure 3: Top: raw data, monthly airline passengers. Middle, prior on functions. Bottom, posterior.

Decomposing a candidate Structure



Bayesian Optimization

Listing 3: Venture Code for Bayesian Optimization

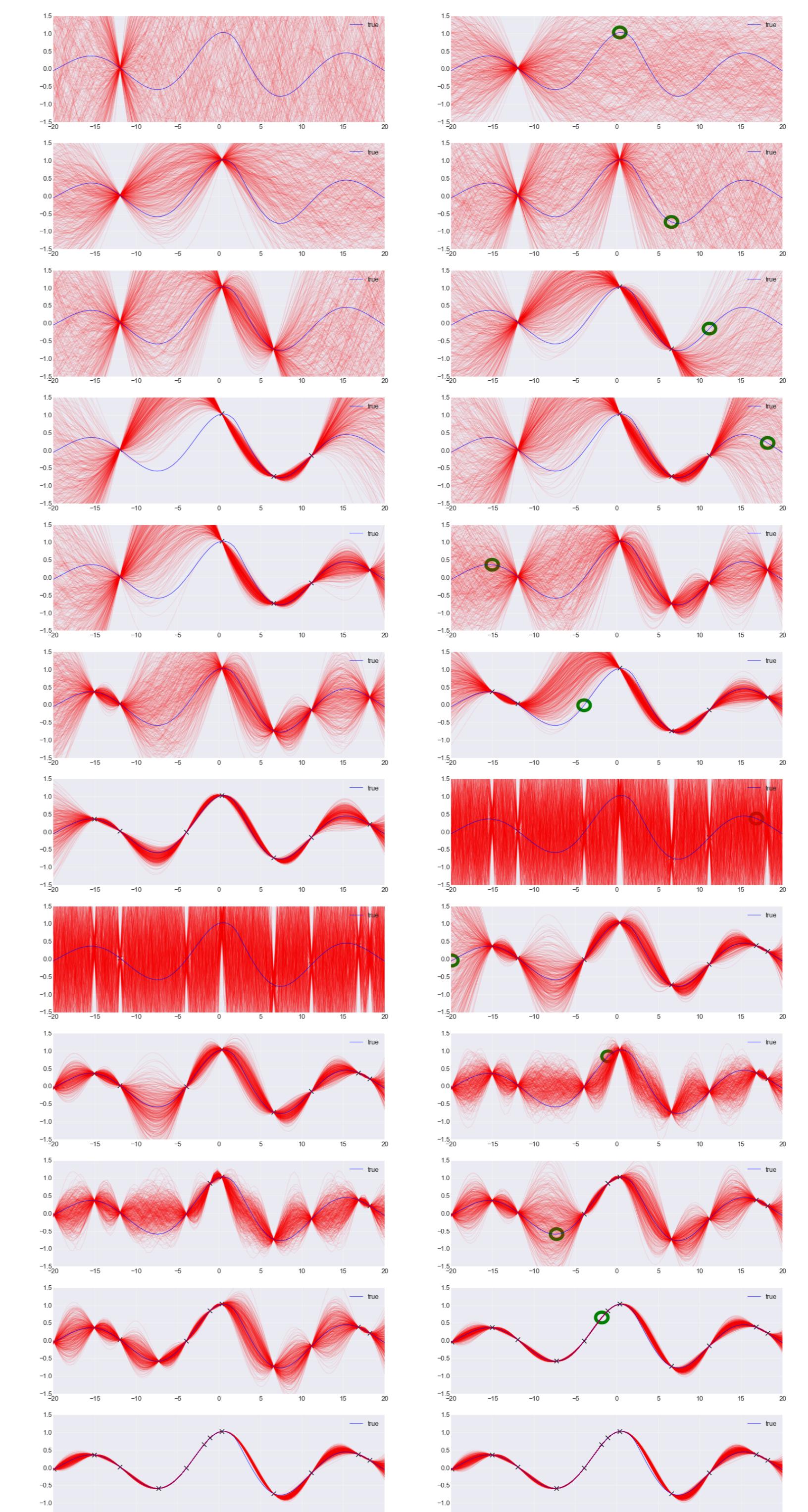
```
assume sf = tag('hyper, log(uniform_continuous(0, 10)))
assume l = tag('hyper, log(uniform_continuous(0, 10)))
assume se = make_se(sf, l)
assume blackbox_f = get_bayeso_blackbox()
assume compute_and_emo = gpmem(blackbox_f, se)

define get_uniform_candidate =
    lambda(prev_xs,uniform_continuous(-20, 20))
define mc_argmax =
    lambda(emulator, prev_xs),
        lambda(candidate_xs,
            lookup(candidate_xs,
                argmax_of_array(mapv(emulator, candidate_xs))))
        mapv(lambda(i, get_uniform_candidate(prev_xs)),
            linspace(0, 19, 20)))
define emulator_point_sample =
    lambda(x,
        run(sample
            lookup(second compute_and_emo() array(x))
            0)))
define infer
repeat(15,do(pass

    // Call f_compute on the next probe point
    predict first(compute_and_emo) = mc_argmax(emulator_point_sample, '-')

    // Hyperparameter inference
    mh('hyper, one, 50)))
```

Thompson Sampling for Homogeneous Sequential Markov Decision Processes



Above: Dynamics of Thompson sampling

- the blue curve is the true function V ;
- the red region is a blending of 100 samples of the curve generated (jointly) by a GP-based emulator V_{emu} ;
- in the left and right columns, the next chosen probe point is circled in green;
- each successive probe point a is the (stochastic) maximum of V_{emu} , sampled pointwise and conditioned on the values of the previously probed points; and
- probes tend to happen at points either where the value of V_{emu} is high, or where V_{emu} has high uncertainty.

References

- [1] D. Duvenaud, J. R. Lloyd, R. Grosse, J. Tenenbaum, and Z. Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1166–1174, 2013.
- [2] R. M. Neal. Monte carlo implementation of gaussian process models for bayesian regression and classification. *arXiv preprint physics/9701026*, 1997.