Supplementary material

Erik Bodin [1] **Markus Kaiser** [2 3] **Ieva Kazlauskaite** [4] **Zhenwen Dai** [5] **Neill D. F. Campbell** [4] **Carl Henrik Ek** [1]

1 $\mathrm{p}(\boldsymbol{h}_*)$ augmentation

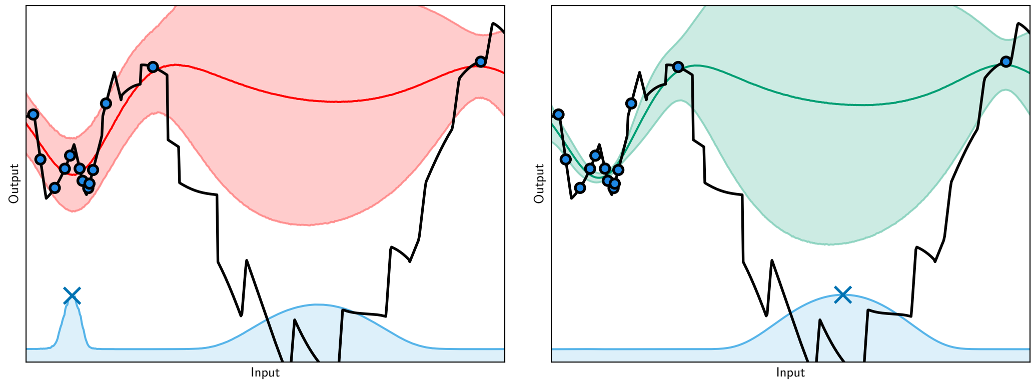

Figure 1. Ignoring irreducible uncertainty from $p(\boldsymbol{h}_*)$ in the acquisition. The effect of marginalizing $p(\boldsymbol{h}_*)$ as per standard versus collapsing the irreducible uncertainty of $h(\boldsymbol{x})$ as $p(\boldsymbol{h}_*) = \delta(\boldsymbol{h}_*)$ is shown on the left and the right, respectively. Note that $p(\boldsymbol{H})$, associated with the observations, is marginalized in both cases. The acquisition (EI) and the next sample location is shown in blue. The posteriors are shown with mean and two standard deviations for display purposes, i.e. with estimated moments and approximated as Gaussian. Note that by incorporating variance induced from $p(\boldsymbol{h}_*)$, which cannot be reduced by acquiring data, the search can end up in a failure mode and get stuck by repeatedly evaluating in a region explained by high irreducible uncertainty in the objective function. By only considering model uncertainty in the function which is reducible by active sampling, i.e. collapsing the predictive $p(\boldsymbol{h}_*)$ associated with new evaluations, the exploration of the function continues.

The predictive distribution of the latent variable for new evaluations $\mathrm{p}(\boldsymbol{h}_*)$, even at observed inputs locations, will always by the i.i.d. assumption be equal to the prior. As such, this source of uncertainty cannot be reduced by active sampling, nor does it reflect observational stochasticity which is made clear by the noiseless experiments. Including it would simply include unhelpful artifacts in the decision upon where to collect data and in the worst case the search would be stuck, see Figure 1. The importance of special treatment of similar uncertainty within active sampling has been noted in (González et al., 2017).

Noting the risk of ill-effects of simply marginalizing $\mathrm{p}(\boldsymbol{h}_*)$, one might be tempted to introduce statistical dependencies in the model such that the belief about $\boldsymbol{h}_*$ associated with a new evaluation is updated from e.g. neighbouring observations. However, such dependencies does not come for free, as they would by necessity limit the model's capability to explain away via $\boldsymbol{H}$. For example, by constraining nuisance parameters to be similar within neighbourhoods of $\mathcal{X}$ the ability to be robust to discontinuities such as step functions would be reduced.

We suggest to remove the effect $\boldsymbol{h}_*$ has on the predictive distribution of $\boldsymbol{f}_*$. In the case of additive models this is easily achieved. For example, in the GP regression context with noisy observations the predictive distribution of the noise-free latent function is easily derived by removal of the noise variable after the noise has been considered in the data. The reason why it is easy in the additive case is because the expectation is trivially separable, as $\mathbb{E}[g+h] = \mathbb{E}[g] + \mathbb{E}[h]$. In this paper we consider the introduction of nuisance parameters in surrogates generally, including those with *non-linear* interactions. In

[1]University of Bristol, United Kingdom [2]Siemens AG, Germany [3]Technical University of Munich, Germany [4]University of Bath, United Kingdom [5]Spotify Research, United Kingdom. Correspondence to: Erik Bodin <erik.bodin@bristol.ac.uk>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

the general non-linear case how to remove the interaction is an open question. For these cases, when there is no available model-specific treatment, we propose the heuristic of collapsing the prior distribution of $\boldsymbol{h}_*$ into a Dirac delta distribution centered at its mode $\boldsymbol{h}_* = \mathbf{0}$, which is consistent with the additive case.

2 Further details on the objective function definition

$$f(\boldsymbol{x}) = g(\boldsymbol{x}, \boldsymbol{h}), \quad \boldsymbol{h} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}), \tag{1}$$

where g is a well-behaved function that can be nicely modeled by a surrogate model of choice, which is a Gaussian process (GP) in this paper, and the vector-valued function $h(\boldsymbol{x})$ encodes the structures which the surrogate model struggles to capture. Let's assume an uninformative prior distribution of $\boldsymbol{x}$ over $\mathcal{X}$, e.g. a uniform distribution $\boldsymbol{x} \sim \mathcal{U}(\mathcal{X})$. We denote the mean and variance of $h(\boldsymbol{x})$ under the prior distribution as $\mu_{\boldsymbol{h}} = \mathbb{E}_{\mathrm{p}(\boldsymbol{x})}[h(\boldsymbol{x})]$ and $\Sigma_{\boldsymbol{h}} = \mathbb{E}_{\mathrm{p}(\boldsymbol{x})}[(h(\boldsymbol{x}) - \mu_{\boldsymbol{h}})(h(\boldsymbol{x}) - \mu_{\boldsymbol{h}})^\top]$, respectively. An important step to convert $\boldsymbol{h}$ into being part of a noise distribution is to treat it as random and independent of $\boldsymbol{x}$, i.e. $\boldsymbol{h}$ becomes an independent random variable with respect to each data point, just like a standard noise term. In this paper, we use a normal distribution for $\boldsymbol{h}$, $\boldsymbol{h} \sim \mathcal{N}(\mu_{\boldsymbol{h}}, \Sigma_{\boldsymbol{h}})$. In this way, the objective function becomes a function of two variables $g(\boldsymbol{x}, \boldsymbol{h})$, in which $\boldsymbol{h}$ is a random variable independent of $\boldsymbol{x}$ and which explain the data variance that cannot be explained by $\boldsymbol{x}$. Note that the random variable $\boldsymbol{h}$ can be equivalently rewritten as $\boldsymbol{h} = \mu_{\boldsymbol{h}} + \boldsymbol{L}\bar{\boldsymbol{h}}$, $\Sigma_{\boldsymbol{h}} = \boldsymbol{L}\boldsymbol{L}^\top$, $\bar{\boldsymbol{h}} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ and then the objective function becomes $g(\boldsymbol{x}, \mu_{\boldsymbol{h}} + \boldsymbol{L}\bar{\boldsymbol{h}})$. As g is a non-linear function inferred during BO, the linear transform can be absorbed into the formulation.

3 Additional examples

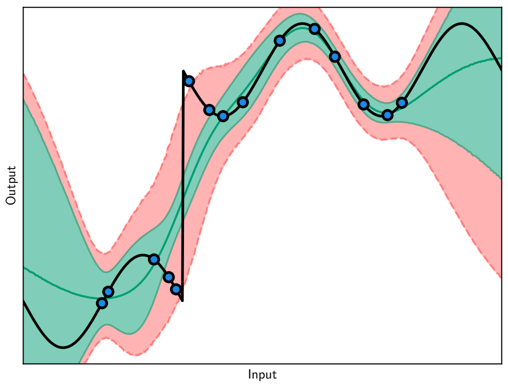

Figure 2. Robustness to nonsmooth structures. The modulated function posterior (of the LGP) is shown in green with mean and two standard deviations. The posterior of f (with $p(\boldsymbol{h}_*)$ marginalized) is shown with two standard deviations from its mean in red. The true function is shown in black. Both posteriors are for display purposes approximated as Gaussian as in Figure 1. Note how some structure of f is ignored and treated as non-additive, heteroscedastic noise.

4 Model setup, inference and auxiliary optimization

We use the Warped GP (Snoek et al., 2014) surrogate model with default settings as provided by the Spearmint library (spe). We use the Matérn 5/2 kernel for all surrogates and the model-marginalised expected improvement acquisition function as in (Snoek et al., 2012b). For all baselines we make us of hyper-parameter marginalisation via Markov Chain Monte Carlo (MCMC) (Shahriari et al., 2016).

For the noise-free, homoscedastic GP and Warped GP (all with few parameters) we use slice sampling, as recommended for BO in (Snoek et al., 2012a) due to its automatic adjustment of the step size to match the local shape of the density function. The heteroscedastic GP has a latent noise variance per observation. Similarly, LGP has the latent inputs $\boldsymbol{H}$ associated with the observations, making inference impractical for both of these models using slice sampling as of comparably large dimensionalites. For these models, in all BO experiments, we use Hamiltonian Monte Carlo with step size adaptation. A burn-in of $30,000$ steps and a thinning rate of $x50$ (select every 50th value) were used, and 100 posterior samples collected. Step-size adaptation were made during 80% of the burn-in phase, with a target acceptance probabilty of 0.75 which is the center of asymptotically optimal rate for HMC (Betancourt et al., 2014).

Observed outputs are normalized to have zero mean and unit variance (standard normalization) at each iteration. For the GP surrogate with and without the noise models as well using the latent input extension, we use a LogNormal(0, 1) prior for the lengthscale and noise variance parameters and use unit signal variance. We use one dimension for the latent variables $\boldsymbol{h}_n$ in the comparisons with the baselines. For the maximization of the expected utility with respect to input location, we use δ-cover sampling, as in (De Freitas et al., 2012), for all models (where in our case the expected utility is the model-marginalised expected improvement). The sampling scheme works by iteratively sampling the utility more densely in $\mathcal{X}$ around the location of current highest obtained utility. Specifically, we double the concentration of an uniform sample density at each auxiliary iteration by multiplication of each side of the current sampling hypercube by a factor of $2^{-1/Q}$ for 30 iterations. Without loss of generality, all function domains are re-scaled to unit hypercubes for a consistent parameterization of priors across functions. In all experiments we start from 2 uniformly drawn initial observations and stop after 50 and 100 observations, respectively, as indicated in the experiments.

5 Benchmarks

5.1 Property labels

For benchmark property labels, see Table 1.

Table 1. Function properties as defined in SigOpt (McCourt, 2016).

Property	Meaning
boring	A mostly boring function that only has a small region of action
oscillatory	A function with a general trend and an short range oscillatory component
complicated	These are functions that may fit a behavior, but not in the most obvious or satisfying way

5.2 Domains

$$\mathrm{pow10}(x) = 10^x \tag{2}$$

$$\mathrm{pow2int}(x) = \mathrm{int}(2^x) \tag{3}$$

For benchmark domains, see Table 2.

Table 2. Benchmark function domains used as specified in SigOpt (McCourt, 2016) 'evalset'. The domains of Corrupted Holder Table and Corrupted Exponential correspond to the ones of Holder Table and Exponential, respectively.

Benchmark	Dim	Properties	Domain
Branin01	2	none	$[[-5,10],[0,15]]$
Branin02	2	none	$[[-5,15],[-5,15]]$
Powell Triple Log	12	none	$[-4,1]^{12}$
Beale	2	boring	$[-4.5,4.5]^2$
Hartmann	6	boring	$[0,1]^6$
Griewank	2	oscillatory	$[-50,20]^2$
Shubert01	2	oscillatory	$[-10,10]^2$
Levy13	2	oscillatory	$[-10,10]^2$
Cosine Mixture	10	oscillatory	$[-1,1]^{10}$
Drop-Wave	10	oscillatory	$[-2,5.12]^{10}$
Deflected Corrugated Spring	10	oscillatory	$[0,7.5]^{10}$
Weierstrass	8	complicated	$[-0.5,0.2]^8$
Cross In Tray	2	complicated, oscillatory	$[-10,10]^2$
Holder Table	2	complicated, oscillatory	$[-10,10]^2$
Ackley $[-10,30]^d$	2	complicated, oscillatory	$[-10,30]^2$
Ackley $[-10,30]^d$	6	complicated, oscillatory	$[-10,30]^6$
Corrupted Holder Table	2	complicated, oscillatory	$[-10,10]^2$
Corrupted Exponential	8	complicated, oscillatory	$[-0.7,0.2]^8$
NN Boston	9	unknown	Table 3
NN Climate Model Crashes	9	unknown	Table 3
Robot Pushing	4	unknown	Table 4

Table 3. Neural Network hyperparameter domains as from (Head et al., 2018). Categorical options are set as specified below.

Parameter	Domain
hidden_layer_sizes	pow2int([1.0, 10.0])
alpha	pow10([-5.0, -1])
batch_size	pow2int([5.0, 10.0])
max_iter	pow2int([5.0, 8.0])
learning_rate_init	pow10([-5.0, -1])
power_t	[0.01, 0.99]
momentum	[0.1, 0.98]
beta_1	[0.1, 0.98]
beta_2	[0.1, 0.9999999]
learning_rate	constant
solver	adam
activation	relu
nesterovs_momentum	False

Table 4. Active learning task of 'robot pushing' as from (Kaelbling & Lozano-Pérez, 2017). Code is available at https://github.com/zi-w/Max-value-Entropy-Search. All instances of np.random.normal(0, 0.01) was replaced by np.random.normal(0, 1e-6) in push_world.py to make the function virtually noise-free. The goal location was set to a fixed location for reproducibility, as specified below.

Parameter	Domain
robot$_x$	[-5, 5]
robot$_y$	[-5, 5]
θ_{initial}	[0, 2π]
simulation steps	int([10., 300.])
goal$_x$	3.0
goal$_y$	2.0

5.3 Regret version of results

For regret version of results table, see Table 5.

5.4 Corrupted Holder Table and Corrupted Exponential

The following corruption functions in Figure 3 was used for the benchmarks Corrupted Holder Table and Corrupted Exponential. *small_corruption_func* and *large_corruption_func* was applied to the SigOpt benchmarks (McCourt, 2016) Holder Table 2D and Exponential 8D, respectively. The input dimensions are re-scaled to be between 0 and 1 before the corruption is applied. The new function minimum of each function (due to the corruption) was estimated via $1e6$ uniformly drawn samples in the domains.

References

Spearmint. `https://github.com/HIPS/Spearmint`. URL `https://github.com/HIPS/Spearmint`.

Betancourt, M., Byrne, S., and Girolami, M. Optimizing the integrator step size for hamiltonian monte carlo. *arXiv preprint arXiv:1411.6669*, 2014.

De Freitas, N., Smola, A., and Zoghi, M. Exponential regret bounds for gaussian process bandits with deterministic observations. *arXiv preprint arXiv:1206.6457*, 2012.

González, J., Dai, Z., Damianou, A., and Lawrence, N. D. Preferential Bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1282–1291, 2017.

Head, T., MechCoder, Louppe, G., Shcherbatyi, I., fcharras, Vincius, Z., cmmalone, Schrder, C., nel215, Campos, N., Young, T., Cereda, S., Fan, T., rene rex, Shi, K. K., Schwabedal, J., carlosdanielcsantos, Hvass-Labs, Pak, M., SoManyUsernamesTaken, Callaway, F., Estve, L., Besson, L., Cherti, M., Pfannschmidt, K., Linzberger, F., Cauet, C., Gut, A., Mueller, A., and Fabisch, A. scikit-optimize/scikit-optimize: v0.5.2, March 2018. URL `https://doi.org/10.5281/zenodo.1207017`.

```python
import numpy as np
from scipy import signal

def corruption(x, a0, a1, a2, a3):
  assert np.all(x >= 0)
  assert np.all(x <= 1)

  a = 0.0 + 1.0 * signal.square(4 * 2 * np.pi * x)
  b = 0.5 + 0.5 * signal.square(4 * 2 * np.pi * x)
  base = a * b

  p0 = 0.3 * np.pi
  p1 = 0
  p2 = np.pi
  p3 = 0.5 * np.pi

  s0 = a0 * signal.sawtooth(p0 + 15 * 2 * np.pi * x)
  s1 = a1 * signal.sawtooth(p1 + 10 * 2 * np.pi * x)
  s2 = a2 * signal.sawtooth(p2 + 30 * 2 * np.pi * x)
  s3 = a3 * signal.sawtooth(p3 + 40 * 2 * np.pi * x)
  return base * (s0 + s1 + s2 + s3)

def corrupt(func, bounds, f_min, f_max, corruption_func):
  f_range = f_max - f_min
  lower_limits = np.array([b[0] for b in bounds])
  upper_limits = np.array([b[1] for b in bounds])
  ranges = upper_limits - lower_limits
  def normalise(v):
    return (v - lower_limits) / ranges
  return lambda v:
    func(v) + \
    f_range * np.max([corruption_func(v_dim_norm) for v_dim_norm in normalise(v)])

small_corruption_func = lambda x: corruption(x, a0=-0.03, a1=0.05, a2=0.08, a3=0.03)
large_corruption_func = lambda x: corruption(x, a0=-0.03, a1=0.20, a2=0.16, a3=0.06)
```

Figure 3. Corruption functions in Python.

Table 5. Regret for various test functions; lower is better. The upper table shows the results after 50 objective function evaluations and the lower table after 100 evaluations. Due to computational cost, Warped GP results are only reported after 50 evaluations.

Benchmark	Evals	Dim	Func Max	Func Min	Initial regret	GP	Warped GP	Homosced GP	Heterosced GP	LGP
Hartmann	50	6	0.00	-3.32	2.764 ± 0.490	0.117 ± 0.109	1.360 ± 0.783	0.343 ± 0.575	0.074 ± 0.081	0.190 ± 0.436
Griewank	50	2	3.19	0.00	1.151 ± 0.441	0.104 ± 0.230	0.379 ± 0.179	0.220 ± 0.150	0.105 ± 0.101	0.102 ± 0.081
Shubert	50	2	210.45	-186.73	183.2 ± 9.820	114.6 ± 51.93	129.3 ± 31.52	113.1 ± 47.77	95.70 ± 60.38	75.34 ± 66.56
Ackley $[-10, 30]^d$	50	2	22.27	0.00	16.45 ± 4.202	1.391 ± 2.101	12.79 ± 3.550	1.641 ± 0.723	1.282 ± 2.481	1.093 ± 0.741
Cross In Tray	50	2	-0.26	-2.06	0.393 ± 0.158	0.018 ± 0.052	0.195 ± 0.123	0.026 ± 0.032	0.009 ± 0.038	0.026 ± 0.062
Holder Table	50	2	0.00	-19.21	15.51 ± 2.571	0.983 ± 1.329	1.433 ± 0.687	1.451 ± 1.182	1.081 ± 1.376	0.112 ± 0.398
Corrupted Holder Table	50	2	3.46	-20.99	18.37 ± 2.444	4.816 ± 4.671	3.508 ± 0.983	3.138 ± 1.123	5.157 ± 5.433	1.836 ± 0.690
Branin01	100	2	308.13	0.40	6.975 ± 5.110	0.001 ± 0.000		0.001 ± 0.001	0.001 ± 0.001	0.000 ± 0.000
Branin02	100	2	506.98	5.56	27.666 ± 28.450	0.253 ± 0.500		0.736 ± 0.690	0.253 ± 0.501	0.380 ± 0.572
Beale	100	2	181853.61	0.00	1152.964 ± 3639.596	0.288 ± 0.302		0.259 ± 0.210	0.241 ± 0.253	0.250 ± 0.251
Hartmann	100	6	0.00	-3.32	2.764 ± 0.490	0.038 ± 0.082		0.162 ± 0.471	0.044 ± 0.066	0.058 ± 0.100
Griewank	100	2	3.19	0.00	1.151 ± 0.441	0.028 ± 0.020		0.099 ± 0.054	0.032 ± 0.023	0.055 ± 0.056
Levy	100	2	454.13	0.00	58.26 ± 37.30	0.289 ± 1.018		0.026 ± 0.028	0.205 ± 0.727	0.034 ± 0.025
Shubert $[-10, 10]^d$	100	2	210.45	-186.73	183.2 ± 9.820	90.09 ± 56.44		88.72 ± 50.59	60.34 ± 58.95	22.70 ± 45.82
Deflected Corrugated Spring	100	10	25.87	-1.00	5.919 ± 1.364	3.921 ± 1.848		0.944 ± 0.574	3.411 ± 1.243	1.797 ± 0.880
Weierstrass	100	8	144.00	112.00	12.90 ± 1.430	5.070 ± 0.997		3.790 ± 0.767	5.401 ± 1.320	4.798 ± 1.436
Cross In Tray	100	2	-0.26	-2.06	0.393 ± 0.158	0.000 ± 0.000		0.001 ± 0.002	0.000 ± 0.000	0.000 ± 0.000
Holder Table	100	2	0.00	-19.21	15.51 ± 2.571	0.459 ± 1.048		0.495 ± 1.079	0.591 ± 1.176	0.005 ± 0.006
Ackley $[-10, 30]^d$	100	2	22.27	0.00	16.45 ± 4.202	0.480 ± 0.720		1.318 ± 0.556	0.347 ± 0.597	0.395 ± 0.332
Ackley $[-10, 30]^d$	100	6	22.27	0.00	19.89 ± 0.809	10.92 ± 6.656		4.191 ± 0.751	11.20 ± 6.438	5.819 ± 5.449
Corrupted Holder Table	100	2	3.46	-20.99	18.37 ± 2.444	2.921 ± 3.728		2.034 ± 0.781	3.396 ± 3.798	1.474 ± 0.342
Corrupted Exponential	100	8	-0.04	-0.99	0.391 ± 0.101	0.172 ± 0.109		0.057 ± 0.016	0.165 ± 0.087	0.074 ± 0.038
HPO: NN Boston	100	9	5.00	-0.85	0.886 ± 1.199	0.023 ± 0.012		0.023 ± 0.009	0.019 ± 0.013	0.021 ± 0.010
HPO: NN Climate Model Crashes	100	9	5.00	0.11	0.161 ± 0.099	0.047 ± 0.017		0.035 ± 0.017	0.040 ± 0.021	0.039 ± 0.017
Active learning: Robot Pushing	100	4	unknown	0.00	3.783 ± 1.842	0.468 ± 0.729		0.829 ± 0.673	0.369 ± 0.537	0.193 ± 0.128

Kaelbling, L. P. and Lozano-Pérez, T. Learning composable models of parameterized skills. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 886–893. IEEE, 2017.

McCourt, M. Optimization test functions. `https://github.com/sigopt/evalset`, 2016.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. 104(1):148–175, 2016. ISSN 0018-9219. doi: 10.1109/JPROC.2015.2494218.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012a.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2951–2959. Curran Associates, Inc., 2012b.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. Input warping for bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, pp. 1674–1682, 2014.