Variational Entropy Search is Just 1D Regression

Michael Pearce
Graphcore Research
Bristol, UK
michaelp@graphcore.ai

Tom Pollak
Graphcore
Bristol, UK
tomp@graphcore.ai

Luke Hudlass-Galley
Graphcore Research
Bristol, UK
lukehg@graphcore.ai

Abstract

Variational Entropy Search (VES) is a recently proposed class of acquisition functions for Bayesian optimization (BO) that unifies Expected Improvement and Max-Value Entropy Search (MES). In BO, a Gaussian process is fit to a set of observed data $\mathcal{D}$ and in MES, at a given input $x \in \mathbb{R}^d$, one samples scalar output values $y \sim \mathbb{P}[y|x, \mathcal{D}]$, then samples functions that fit both $\mathcal{D}$ and (x, y) and finds the peaks of these functions y^*. These y^* samples are conditioned on y and come from a non-trivial distribution $\mathbb{P}[y^*|y]$. Given x, the MES goal is to estimate how much the potential y value may reduce entropy of this distribution. The VES goal, is to instead learn a variational approximation $q(y^*|y)$ that estimates a lower bound to MES. In this work, for a given point x we reinterpret VES as a simple 1-dimensional *frequentist* regression problem from y to y^*. By framing prior work in this perspective, we explore possible improvements, including generalizing VES to noisy objectives. We explore a variety of simple 1D regression models in BO benchmarks on synthetic data, highlighting open questions for future research.

1 Introduction

Bayesian optimization (BO) [1, 2] is a class of active learning algorithms for solving expensive, non-differentiable, stochastic, black box objective functions $\max_{x \in \mathcal{X}} f(x)$ where $\mathcal{X} \subset \mathbb{R}^d$ is typically a user-defined box-constrained search space, and the output is a single scalar. Starting from a small set of evaluated points $\{(x_1, y_1), ..., (x_n, y_n)\} = \mathcal{D}_n$, classical BO methods consist of two components: a Gaussian process surrogate model trained on the data so far mapping a new input x to a prediction and uncertainty $\mathcal{N}(y|\mu_n(x), \sigma_n^2(x))$, and an acquisition function $\alpha(x) \in \mathbb{R}$ that at any given x uses the model to quantify the expected benefit of hypothetically evaluating the expensive objective $y = f(x)$. The location with the highest acquisition value, $x_{n+1} = \arg\max \alpha(x)$, is evaluated returning $y_{n+1} = f(x_{n+1})$. The new point (x_{n+1}, y_{n+1}) is added to the dataset to make $\mathcal{D}_{n+1}$ and the algorithm repeats until a user-specified stopping criteria is met.

The acquisition function must balance the trade-off between exploration and exploitation. Over the years, many methods have been proposed, Expected Improvement (EI) [3, 4, 5], Knowledge Gradient [6, 7, 8], Entropy Search [9, 10, 11, 12], Max-Value Entropy Search (MES) [13, 14] and a recently proposed MES version, Variational Entropy Search (VES) [15] which is the focus of this work.

Let $\bar{y} = \max(y_1, .., y_n)$ be the best-seen point. We may write Expected Improvement as

$$\alpha_{\text{EI}}(x) = \mathbb{E}_{y_{n+1}}[\max(y_{n+1} - \bar{y}, 0)],$$

where $y_{n+1} \sim \mathcal{N}(y \,|\, \mu_n(x), \sigma_n^2(x))$. Intuitively, at a point x, if "lucky" y_{n+1} will provide an improvement $y_{n+1} - \bar{y}$, or if "unlucky", no improvement. EI is the expectation of these hypothetical outcomes using the Gaussian distribution for y_{n+1} at $x_{n+1} = x$ and the GP model fit to $\mathcal{D}_n$.

For MES, one utilizes the fact that a GP is a generative model of functions by sampling a full function $f^{(i)}(x)$ consistent with the observed points $\mathcal{D}_n$ and finding its maximum output value $y^{*(i)}$, the GP induces a distribution over y^*. This distribution has an (implicit unknown) density $p(y^*|\mathcal{D}_n)$ and a

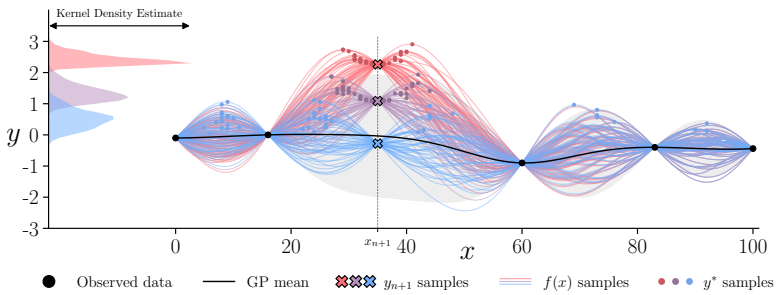


Figure 1: Illustrative example of a Gaussian Process fit to five observed data points. At $x_{n+1} = 35$, there are three samples of y_{n+1}, each of which has 50 sampled functions f and peaks y^*. *(Left)* kernel density estimates (KDEs) of $p(y^*|x_{n+1}, y_{n+1})$ for the three distributions of peaks. Note that as y_{n+1} increases, the KDEs become more narrowly peaked. MES aims to compute the expected entropy over these distributions. VES fits a parametric distribution as a function of y_{n+1}.

corresponding entropy (see Figure 1 for an illustration). At $x_{n+1} = x$, if lucky, the new y_{n+1} reduces this entropy; alternatively entropy is unchanged. MES is the average of these potential outcomes, which is the (negative) expected future entropy

$$\alpha_{\text{MES}}(x) = \mathbb{E}_{y_{n+1}}\left[\,\mathbb{E}_{y^*}[\log p(y^*|y_{n+1})]\,\right]. \tag{1}$$

We omit $x = x_{n+1}$ and $\mathcal{D}_n$ for brevity. As $p(y^*|y_{n+1})$ is non-trivial, recent work [15] proposed to use a variational approximation $q(y^*|y_{n+1})$. The result is a lower bound of the true MES acquisition function, see Appendix B.1, denoted as the Entropy Search Lower Bound objective,

$$\alpha_{\text{ESLBO}}(x|q_\theta(\cdot)) = \mathbb{E}_{y_{n+1}}\left[\,\mathbb{E}_{y^*}[\log q_\theta(y^*|y_{n+1})]\,\right]. \tag{2}$$

The functional form of $q_\theta(\cdot)$ is a user design choice with parameters θ uniquely learned for each x by maximizing ESLBO; the outcome is the Variational Entropy Search (VES) acquisition function at point x. For certain choices, ESLBO and θ can be partially computed analytically (see Equation 12 in [15]). The authors consider deterministic objectives $f(x)$, where the maximum value cannot be lower than any data point $y^* \geq y_{n+1}$, for which they propose VES-Gamma where $q(\cdot)$ is a given by

$$q(y^*|\mathcal{D}_n, x, y_{n+1}, k, \beta) = \text{Gamma}(y^* - \max(\bar{y}, y_{n+1})|k, \beta). \tag{3}$$

The $\max(\bar{y}, y_{n+1})$ term is a lower bound on the range of y^* that varies with y_{n+1}, and the Gamma distribution has support over $[0, \infty)$. The variational parameters (k, β) are learned for each x. Motivated by the observation that "*a deeper interpretation of the ESLBO remains an open research question*" [15], we highlight that the ESLBO is equivalent to 1-dimensional frequentist regression, and use this interpretation to explore alternative approximations.

2 VES as 1-Dimensional Regression

Many 1-dimensional regression models may be represented as an underlying trend line plus random noise, $\tilde{y} = g_\theta(\tilde{x}) + \epsilon$ for some $\tilde{x}$ and $\tilde{y}$. Traditional linear regression assumes $g(\tilde{x}) = m\tilde{x} + c$ and $\epsilon \sim \mathcal{N}(0, 1)$, given a training set of $(\tilde{x}^{(i)}, \tilde{y}^{(i)})$ pairs, the trend line parameters $\theta = (m, c)$ are found by minimizing prediction mean squared error (MSE) which is mathematically equivalent to maximizing Gaussian noise likelihood. In general, for any trend line (linear, quadratic, neural network) and any noise distribution (Gaussian, exponential, log-normal), the log likelihood of each $\tilde{y}^{(i)}$ at $\tilde{x}^{(i)}$ is an objective or goodness-of-fit metric that is used to optimize the parameters θ,

$$\mathcal{L}_{\text{1D}}(\theta) = \log\left(\prod_i p(\tilde{y}^{(i)}|g_\theta(\tilde{x}^{(i)}))\right) = \sum_i \log p(\tilde{y}^{(i)}|g_\theta(\tilde{x}^{(i)})). \tag{4}$$

Regarding ESLBO in Equation 2, at a given x_{n+1}, the predictive distribution of y_{n+1} is a known Gaussian. However, y^* can only be sampled, hence the ESLBO must be (all or partly) evaluated by Monte-Carlo averaging, this requires a set of points $(y^{(i)}_{n+1}, y^{*(i)}) \in \mathbb{R}^2$, such a dataset is shown in Figure 2. In which case the ESLBO objective becomes

$$\mathcal{L}_{\text{ESLBO}}(\theta) = \sum_i \log q_\theta(y^{*(i)}|y^{(i)}_{n+1})$$

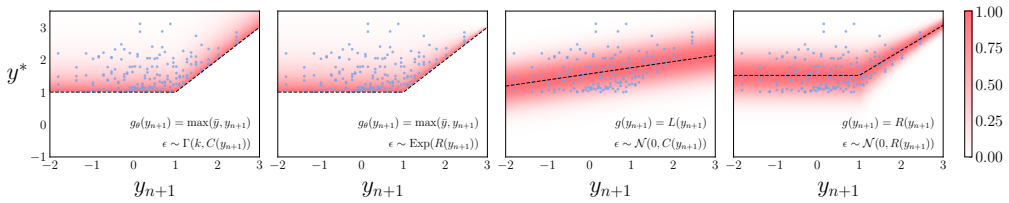


Figure 2: Heatmaps of different regression models optimized on samples of (y_{n+1}, y^*) for a given x_{n+1}. For each model, the dashed line indicates the trend line $g_\theta(y_{n+1})$. Each model has its own subset of the parameters $\{m_g, c_g, m_\epsilon, c_\epsilon, k\}$ that are optimized. Non-parameterized trend lines (such as the first two plots) serve as a strict lower bound of the distribution of y^*. For a full set of regression models, please refer to Appendix A.2.

which is identical to Equation 4, where $\tilde{x} \to y_{n+1}$, $\tilde{y} \to y^*$, and $g()$ is incorporated into $q()$. As a concrete example, one may use basic linear regression (such as `sklearn.linear_model.LinearRegression` in Python), to fit a linear trend line through (y_{n+1}, y^*) points and the negative MSE is an estimate of ESLBO (up to constant terms, see Appendix B.2). The slope and intercept $\theta = (m, c)$ are the variational parameters in ESLBO. For VES-Gamma, the trend line is given by $g(y_{n+1}) = \max(\bar{y}, y_{n+1})$ with noise $\epsilon \sim \text{Gamma}(k, \beta)$ and learned parameters $\theta = (k, \beta)$. In the linear regression example, trend line parameters $\theta = (m, c)$ are learned and the noise parameter $\sigma = 1$ is fixed, while in VES-Gamma, the noise parameters $\theta = (k, \beta)$ are learned and trend line is fixed. In general, we may learn both trend line and noise parameters together. Finally, the next input x_{n+1} would be the location with the highest likelihood or "goodness-of-fit" on it's corresponding 1D regression dataset of samples (y^*, y_{n+1}).

The assumptions in VES-Gamma allow for analytic integration of parts of the ESLBO and is a significant theoretical result. However we note three potential extensions. Firstly, the trend function $g(y_{n+1}) = \max(\bar{y}, y_{n+1})$ is not parameterized. Secondly, the noise distribution $\epsilon \sim \text{Gamma}(k, \beta)$ does not vary with y_{n+1}, it assumes a constant variance/entropy of y^* over the input space y_{n+1}, also known as *homoskedasticity* in 1D regression. As can be seen in Figures 1 and 2, the spread of y^* points varies with y_{n+1}, this can be modeled as *heteroskedasticity* where noise parameters depend on y_{n+1}, capturing that for various y_{n+1} values, y^* entropy is either reduced or unchanged. Thirdly, VES-Gamma was designed for deterministic objectives that assume $y^* \geq y_{n+1}$. However, if the black box $f(x)$ is stochastic, y^* now represents the expected output $\max_x \mathbb{E}[f(x)]$, and the noisy individual $y_1, \ldots, y_{n+1}$ values are not strictly a lower bound on the range of y^*.

These potential issues are all addressed by simply considering alternative 1D regression models.

3 Experiments

As a GP is a generative model of functions, we specify a kernel and synthesize 100 test functions. We set the same kernel and hyperparameters for the GP in the BO algorithm. We specify a discretized 2D input space of 200 points and compute the acquisition function at every point. Thus, GP model fitting and acquisition function optimizer are the "best-case" across all methods isolating the acquisition function as the only difference. We use PyTorch and Adam for parameter learning.

Efficient Sampling At a given x, naively sampling a point (y^*, y_{n+1}) would have $O(|\mathcal{X}|^3)$ complexity. Instead, we pre-sample 30 functions from $\boldsymbol{f} \sim \mathcal{GP}(f(x')|\mathcal{D}_n)$ over the input grid $\mathcal{X}$ with cost $O(|\mathcal{X}|^3)$. Then, at a given $x \in \mathcal{X}$, sampling a single (y_{n+1}, y^*) can be done in $O(|\mathcal{X}|)$ by exploiting cheap updates. Thus we pre-compute 10 Z scores and by parallelizing over $\mathcal{X}$, Z and $\boldsymbol{f}$, we can compute $10 \times 30 = 300$ unbiased (y_{n+1}, y^*) samples per x on modern hardware, see Appendix C.

Table 1: The basis functions used to model trend lines and heteroskedasticity.

Method	Function	Parameters	Heteroskedastic
Constant	$C(y_{n+1}) = c$	c	✗
Linear	$L(y_{n+1}) = my_{n+1} + c$	m, c	✓
ReLU	$R(y_{n+1}) = m \max(\bar{y}, y_{n+1}) + c$	m, c	✓
Monte-Carlo (MC) (App. C.1)	$M(y_{n+1}) = \text{MLE}(y^{*(i)}\|y_{n+1})$	-	✓

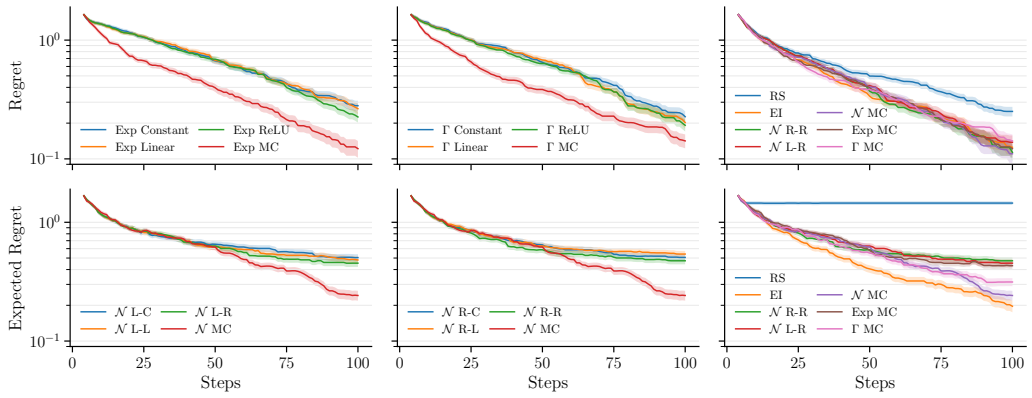


Figure 3: *(Top row)* regret ($y^* - \max(y_{1:n})$) on deterministic test problems. *(Left, Center)* Exponential and Gamma noise models with different noise parameter functions, MC is significantly better suggesting the noise parameter functions are not well fit. *(Right)* best performing methods beat RS and match or outperform EI. *(Bottom row)* expected regret ($y^* - \mathbb{E}[f(x_r)]$) on stochastic functions where x_r is best GP predicted point. *(Left, Center)* varying the trend and Gaussian variance models, $\mathcal{N}$ R-L: ReLU trend and Linear variance etc. *(Right)* all outperform RS, but struggle to match EI.

Heteroskedasticity Firstly, we consider deterministic problems where we fix the "ReLU" trend line $g(y_{n+1}) = \max(\bar{y}, y_{n+1})$ and use Exponential and Gamma noise models with scale parameter β a learnable function of y_{n+1} from the table above and for Gamma a learned constant k. Figure 3 top row shows convergence curves for Exp and Gamma. Exponential shows minor improvement when increasing heteroskedasticity modeling. For Gamma, prior work noted instability and applied regularization to the k parameter, we also performed extensive parameter sweeps on how to regularize k in Appendix C.2 and found it often struggles to outperform RS in our experiments. In both cases, MC appears to work best, suggesting the noise models are underfitting.

Trend Line Secondly, we consider a learning Linear and ReLU trend lines, thus requiring a distribution that can model positive and negative noise deviations from the trend line and so we adopt the Gaussian distribution. We now also consider noisy objective functions. For modeling the Gaussian variance, $\sigma^2(y_{n+1})$, we consider constant, linear and ReLU and MC as shown in Figure 2. Results are in Figure 3. Again, increasing the heteroskedasticity yields a minor improvement while going from linear to ReLU trend surprisingly appears to have a less significant effect.

Other Baselines Finally, we compare the best VES methods, against Random Search (RS) and Expected Improvement (EI) on deterministic and noisy objective functions (for Exp, Gamma, we simply mask out invalid points $y^* < y_{n+1}$), see Figure 3. On deterministic functions, all methods perform similarly and significantly outperform RS while on stochastic functions, the VES methods fall behind with the exception of Gaussian and Gamma MC again suggesting that the noise models are underfitting. Alternatively, these result may challenge the fundamental hypothesis of VES and MES that more accurately estimating entropy reduction does not necessarily translate into a better BO algorithm in this idealized controlled setting.

4 Conclusion

The VES framework allows for any variational approximation, and prior work investigated deterministic objectives and simple approximations. We highlight that any 1D regression model can be used and seamlessly extend to noisy objectives. We explore a range of regression models and note the variation in algorithm quality, these preliminary results in an idealized setting may cast doubt on the fundamental hypothesis of MES, that accurate entropy estimation translates to a better BO algorithm. Future work includes exploring analytic integration of ESLBO with these new $q(\cdot)$ models, and evaluating real world, high-dimensional benchmarks with comparison to traditional MES implementations and other baselines.[1]

[1]All code is available at https://github.com/graphcore-research/mes

References

[1] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[2] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[3] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

[4] Jonas Mockus. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2:117, 1998.

[5] Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Unexpected improvements to expected improvement for bayesian optimization. *Advances in Neural Information Processing Systems*, 36:20577–20612, 2023.

[6] Peter I Frazier, Warren B Powell, and Savas Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

[7] Juan Ungredda, Michael Pearce, and Juergen Branke. Efficient computation of the knowledge gradient for bayesian optimization. *arXiv preprint arXiv:2209.15367*, 2022.

[8] Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

[9] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. Predictive entropy search for multi-objective bayesian optimization. In *International conference on machine learning*, pages 1492–1501. PMLR, 2016.

[10] Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44(4):509–534, 2009.

[11] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *The Journal of Machine Learning Research*, 13(1):1809–1837, 2012.

[12] Carl Hvarfner, Frank Hutter, and Luigi Nardi. Joint entropy search for maximally-informed bayesian optimization. *Advances in Neural Information Processing Systems*, 35:11494–11506, 2022.

[13] Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International conference on machine learning*, pages 3627–3635. PMLR, 2017.

[14] Henry B Moss, David S Leslie, Javier Gonzalez, and Paul Rayson. Gibbon: General-purpose information-based bayesian optimisation. *Journal of Machine Learning Research*, 22(235):1–49, 2021.

[15] Nuojin Cheng, Leonard Papenmeier, Stephen Becker, and Luigi Nardi. A unified framework for entropy search and expected improvement in bayesian optimization. *arXiv preprint arXiv:2501.18756*, 2025.

A Additional Figures

A.1 Gaussian Processes with Noisy Objective Functions

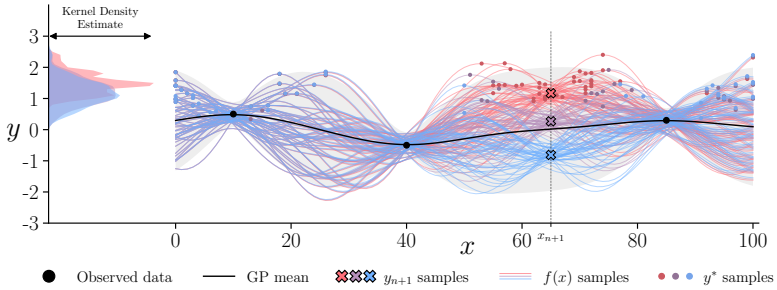


Figure 4: Illustrative example of a Gaussian Process fit to three observed data points. At $x_{n+1} = 65$, there are three samples of y_{n+1}, each of which has 50 sampled functions f and peaks y^*. Unlike in Figure 1, the observed data points are stochastic; there is variance associated with the observed points and hence y^* may be smaller than y_{n+1}. *(Left)* kernel density estimates of $p(y^*|x_{n+1}, y_{n+1})$ for the three distributions of peaks.

A.2 Heatmaps

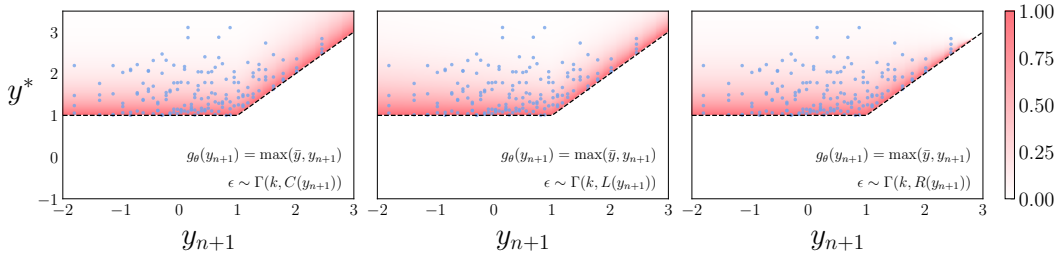


Figure 5: Regression models optimized on samples of (y_{n+1}, y^*) for a given x_{n+1} with noise modeled by the Gamma distribution with varying formulations of its scale parameter (clamped to have a minimum value of 10^{-6}). *(Left)* the scale parameter is constant for all y_{n+1}. *(Middle)* the scale parameter is a linear function of y_{n+1}. *(Right)* The scale parameter is a linear function of the best-seen point $\max(\bar{y}, y_{n+1})$. In all cases of the Gamma distribution, the trend line is not parameterized, and serves as a lower bound of the distribution of y^*.

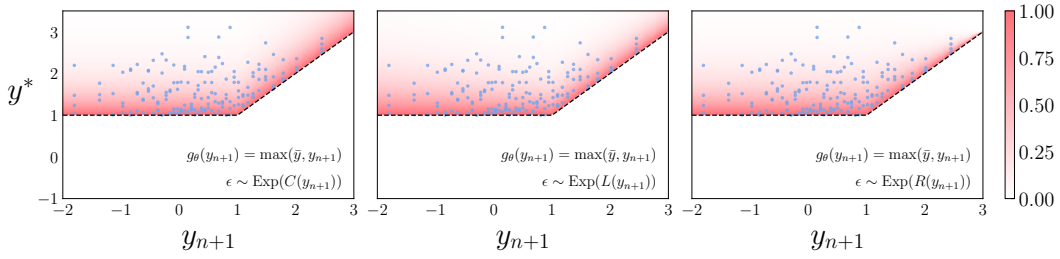


Figure 6: Regression models optimized on samples of (y_{n+1}, y^*) for a given x_{n+1} with noise modeled by the Exponential distribution with varying formulations of its scale parameter. *(Left)* the scale parameter is constant for all y_{n+1}. *(Middle)* the scale parameter is a linear function of y_{n+1}. *(Right)* The scale parameter is a linear function of the best-seen point $\max(\bar{y}, y_{n+1})$. In all cases of the Exponential distribution, the trend line is not parameterized, and serves as a lower bound of the distribution of y^*.

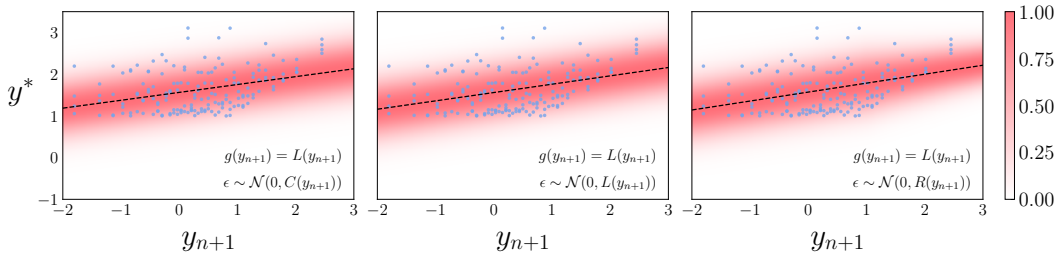


Figure 7: Regression models optimized on samples of (y_{n+1}, y^*) for a given x_{n+1} with the trend modeled by linear regression with Gaussian noise and varying formulations of its variance σ^2. *(Left)* the variance is constant for all y_{n+1}. *(Middle)* the variance is a linear function of y_{n+1}. *(Right)* The variance is a linear function of the best seen point $\max(\bar{y}, y_{n+1})$. Unlike the Gamma and Exponential distributions, these linear regression models are parameterized and sit within the (y_{n+1}, y^*) samples, thus the noise deviations may be positive or negative.

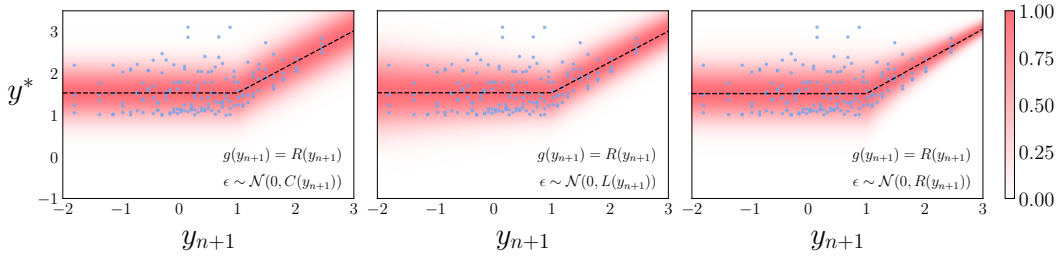


Figure 8: Regression models optimized on samples of (y_{n+1}, y^*) for a given x_{n+1} with the trend modeled by a piecewise linear regression ("ReLU") with Gaussian noise and varying formulations of its variance σ^2. *(Left)* the variance is constant for all y_{n+1}. *(Middle)* the variance is a linear function of y_{n+1}. *(Right)* The variance is a linear function of the best seen point $\max(\bar{y}, y_{n+1})$. Unlike the Gamma and Exponential distributions, these linear regression models are parameterized and sit within the (y_{n+1}, y^*) samples, thus the noise deviations may be positive or negative.

B Derivations

B.1 ESLBO Lower Bound to MES

All expectations over y_{n+1} are conditioned on known $x_{n+1} = x$ and D_n.

$$\text{MES}(x) \quad = \quad \mathbb{E}_{y_{n+1}}[\mathbb{E}_{y^*}[\log p(y^*|y_{n+1})]]$$

Taking the term inside the expectation alone, we may lower bound it as follows

$$\begin{aligned}
\mathbb{E}_{y^*}[\log p(y^*|y_{n+1})] &= \int_{y^*} \log p(y^*|y_{n+1}) p(y^*|y_{n+1}) dy^* \\
&= \int_{y^*} \Big(\log p(y^*|y_{n+1}) - \log q(y^*|y_{n+1}) + \log q(y^*|y_{n+1}) \Big) p(y^*|y_{n+1}) dy^* \\
&= \text{KL}\big(p(\cdot)\big|\big|q(\cdot)\big) + \int_{y^*} \log q(y^*|y_{n+1}) p(y^*|y_{n+1}) dy^* \\
&\geq \int_{y^*} \log q(y^*|y_{n+1}) p(y^*|y_{n+1}) dy^* \\
&= \mathbb{E}_{y^*}[\log q(y^*|y_{n+1})]
\end{aligned}$$

where the inequality sign comes from removing the KL divergence term which is guaranteed to be non-negative. As a result we have that

$$\begin{aligned}
\text{MES}(x) &= \mathbb{E}_{y_{n+1}}[\mathbb{E}_{y^*}[\log p(y^*|y_{n+1})]] \\
&\geq \mathbb{E}_{y_{n+1}}[\mathbb{E}_{y^*}[\log q(y^*|y_{n+1})]] \\
&= \text{ESLBO}(x)
\end{aligned}$$

The tightness of the bound is the KL divergence from the true $p(y^*|\cdot)$ to the approximate $q(y^*|\cdot)$.

B.2 ESLBO is Minimum Mean Squared Error

Assume we have a dataset of $(\tilde{x}_i, \tilde{y}_i)$ points, assume we have a trend line $g(\tilde{x}) = m\tilde{x} + c$ and that each $\tilde{y}$ value is modeled as $\tilde{y} = g(\tilde{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The likelihood is given by

$$\begin{aligned}
\mathcal{L}_{ESLBO}(\theta) &= \log \left(\prod_i^N \mathcal{N}(\tilde{y}_i | g_\theta(\tilde{x}_i), \sigma^2) \right) && (5) \\
&= \sum_i^N \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(\tilde{y}_i - g_\theta(\tilde{x}_i))^2}{2\sigma^2} \right) \right) && (6) \\
&= \sum_i^N -\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (\tilde{y}_i - g_\theta(\tilde{x}_i))^2 && (7) \\
&= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{N}{2\sigma^2} \left(\frac{1}{N} \sum_i^N (\tilde{y}_i - g_\theta(\tilde{x}_i))^2 \right) && (8) \\
&= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{N}{2\sigma^2} \text{MSE}(\tilde{\mathbf{y}}, g_\theta(\tilde{\mathbf{x}})) && (9)
\end{aligned}$$

where $\tilde{\mathbf{x}}$ is the concatenated vector of input points and similar for $\tilde{\mathbf{y}}$. If we set $\sigma^2 = 1$, ESLBO and MSE differ by additive and multiplicative constants that do not vary with the dataset $(\tilde{x}_i, \tilde{y}_i)$ points. In the greater BO context, each $x \in \mathcal{X}$ h has a corresponding bespoke dataset (y_{n+1}, y^*) and if $\sigma = 1$ for every x location, then the location with lowest MSE is the location with highest ESLBO. This does not hold if we learn different σ parameter for each $x \in \mathcal{X}$ and in this work we consider learned σ.

C Implementation

C.1 VES with Maximum Likelihood Estimated Noise Parameters

Using the above proposed method to generate samples (y_{n+1}, y^*), for any single $y_{n+1}^{(i)}$ sample we can generate many $y^{*(ij)}$ samples. Consequently, we may learn an Exponential/Gamma/Gaussian distribution to y^* samples for a single y_{n+1} and *not* learn a function over y_{n+1}, for example

$$\text{MLE}(y^*|{y_{n+1}}^{(i)}) \quad = \quad \max_{\mu_i,\sigma_i} \prod_j \mathcal{N}(y^{*(ij)}|\mu_i, \sigma_i) \tag{10}$$

where the maximization simply yields the data empirical mean and variance in the Gaussian case. The ESLBO becomes

$$\text{ESLBO}(x) = \frac{1}{n_{y_{n+1}}} \sum_i \frac{1}{n_{y^*}} \max_{\mu_i,\sigma_i} \sum_j \log \mathcal{N}(y^{*(ij)}|\mu_i, \sigma_i). \tag{11}$$

Thus the MC model makes no assumption about how the variance of y^* changes with y_{n+1} and hence has the most flexibility, but it does require multiple y^* samples for each y_{n+1} which can be unstable for small sample sizes.

C.2 Regularized Shape Estimation for VES-Gamma

For VES-Gamma we estimate the Gamma shape $k > 0$ at each x by first using the samples (y_{n+1}, y^*) to compute the individual noise values ϵ

$$\epsilon^{(i)} = y^{*(i)} - \max(\bar{y}, y_{n+1}^{(i)})$$

Then we fit a single Gamma distribution to all these noise values, we first compute the following quantity by Monte-Carlo

$$\log \mathbb{E}[\epsilon] - \mathbb{E}[\log(\epsilon)] \approx \frac{1}{n} \sum_i \epsilon^{(i)} - \frac{1}{n} \sum_i \log \epsilon^{(i)} = \Delta$$

where n is the number of samples. Then we may find a value for k by solving the following equation

$$\log(k) - \psi(k) - \Delta = 0. \tag{12}$$

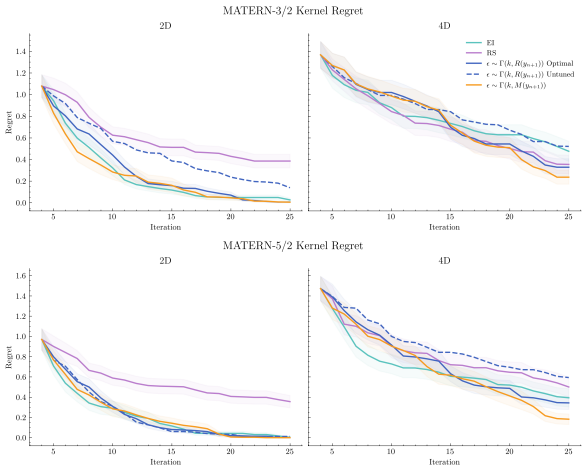
which may be solved using the Newton-Rhaphson method. Ideally, this solver is a monotonic iterative computation converging to the ideal k. However this often becomes unstable especially when $\Delta \approx 0$ as noted in previous work on VES-Gamma [15]. The solution provided in the previous work is to stabilize the estimate with simple L_2 regularizer on k. Concretely, in this work we solve

$$\min_k \ \big(\log(k) - \psi(k) - \Delta\big)^2 + \lambda(k-1)^2 \tag{13}$$

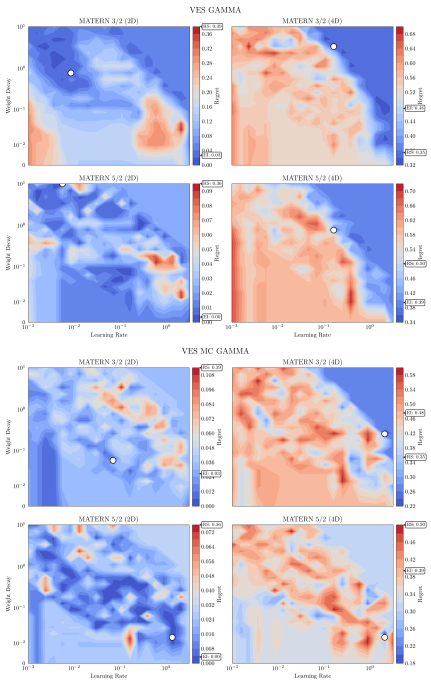
which is ridge regularization centered at $k = 1$, as the gamma distribution with $k = 1$ corresponds to the exponential distribution. The weight parameter λ is set by the user, we perform parameter sweeps below and see how the BO algorithm results change.

C.3 Hyperparameter tuning for VES-Gamma

We study the sensitivity of VES-Gamma to two implementation choices: the Adam step size η used to maximize α_{ESLBO} at each x, and the weight decay λ applied to the Gamma shape parameter k via the ridge penalty in Eq. (13).



(a) Tuned vs. default. Without weight decay ($\lambda = 0$), VES-Gamma can become numerically unstable in higher dimensions ($d \geq 4$), with the learned shape k growing very large (ill-conditioned updates). Introducing modest regularization ($\lambda > 0$) stabilizes training and yields performance competitive with EI.



(b) Regret as a function of Adam step size η and weight decay λ. With the ridge penalty in Equation (13), VES-Gamma performance can vary from matching EI to worse than random search though is stable for regions of the parameter space. VES-MC-Gamma (Appendix C.1) is less sensitive to hyperparameters and typically performs better.

Figure 9: Hyperparameter effects for VES-Gamma. We keep the GP kernel and hyperparameters fixed between the surrogate and the data-generating process, matching the setup in Appendix C.4.

C.4 Efficiently Parallel Sampling (y_{n+1}, y^*) Points

Given a GP model $\mathcal{GP}(\mu(x), k(x, x')|\mathcal{D}_n)$ and a discrete set of points $\mathcal{X} \subset \mathbb{R}^d$ with $|\mathcal{X}| = N$, we evaluate the GP posterior mean and covariance function at the grid yielding a mean vector and a covariance matrix $\mu \in \mathbb{R}^N, \Sigma \in \mathbb{R}^{N \times N}$. We use these to generate a sample function from the multi-variate normal $\mathbf{f} \sim \mathcal{N}(\mu, \Sigma)$ which requires Cholesky decomposition of Σ with complexity $O(N^3)$. For a given index $j \in \{1, ..., N\}$ corresponding to location $x_j \in \mathcal{X}$, we can sample a point pair (y_{n+1}, y^*) using Algorithm 1 with cost $O(N)$. We generate $n_{y_{n+1}}$ z-scores at uniformly spaced quantiles of the Gaussian distribution $\mathbf{Z}$ with $Z_i = \Phi^{-1}(\frac{i-0.5}{n_{y_{n+1}}})$. Parallelizing over all N locations $\mathcal{X}$, the $n_{y_{n+1}}$ scores $\mathbf{Z}$, and the n_{y^*} sampled functions $\mathbf{f}$ allows for GPU acceleration, see code in Appendix C.4 and Github for a PyTorch implementation.

Algorithm 1: Updating a sampled function with one new data point

Input: mean $\mu \in \mathbb{R}^N$, covariance $\Sigma \in \mathbb{R}^{N \times N}$, observation noise $\sigma^2 \in \mathbb{R}$, sampled function $\mathbf{f} \in \mathbb{R}^N$, index to modify i
Output: sample (y_{n+1}, y^*)
$y_{n+1} \leftarrow \mathcal{N}(\mu_i, \Sigma_{ii} + \sigma^2)$
$y_i \leftarrow \mathcal{N}(f_i, \sigma^2)$
$\mathbf{f}' \leftarrow \mathbf{f} + \frac{(y_{n+1} - y_i)}{\Sigma_{ii} + \sigma^2} \Sigma_{i,:}$
$y^* \leftarrow \max(\mathbf{f}')$
return (y_{n+1}, y^*)

Listing 1: Efficient sampling of (y_{n+1}, y^*) in Numpy, see Github for full implementation

```python
def sample_yn1_ymax(
    *,
    y_mean: np.ndarray,
    y_cov: np.ndarray,
    y_noise_std: float,
    n_yn1: int=10,
    n_ymax: int=30,
    batch_size: int=1e9,
    noise_jitter: float=1e-9,
) -> np.ndarray:
    """
    Given the mean and covariance of all the function values at all the x-locations,
    for each x-location, sample y_{n+1} values, for each y_n1, we then sample n_ymax
    full functions from a model fit to data that includess the new (x, y_n1) point
    and then find the peak. This is vecotrised over all dimensions and the output is
    a matrix of (n_x, n_yn1) values for y_n1 and
    a tensor of (n_x, n_yn1, n_ymax) sampled y_max values.

    Args:
        y_mean: np.ndarray, shape (n_x, 1)
        y_cov: np.ndarray, shape (n_x, n_x)
        y_noise_std: float, noise standard deviation of y values for the objective function.
        n_yn1: int, number of y_n1 values to sample for each x-location
        n_ymax: int, number of functions to sample from the model
        batch_size: int, number of x-locations to process in each batch
        noise_jitter: float, noise to add to the covariance matrix

    Returns:
        y_n1_output: np.ndarray, shape (n_x, n_yn1)
        y_funcs_output: np.ndarray, shape (n_x, n_yn1, n_ymax, n_x)
        y_max_output: np.ndarray, shape (n_x, n_yn1, n_ymax)
        y_funcs: np.ndarray, shape (n_x, n_ymax)
    """
    y_mean = y_mean.reshape(-1, 1)
    n_x = y_mean.shape[0]       # total number of x -locations
    bs = min(batch_size, n_x)   # batch size
    y_noise_var = y_noise_std**2

    batch_idx_subsets = np.array_split(np.arange(n_x), n_x // bs)

    # (n_x, n_x) square matrices
    y_cov += noise_jitter * np.eye(y_cov.shape[0])
    chol_k = np.linalg.cholesky(y_cov)

    f_var = np.diag(y_cov)[:, None]
    y_n1_var = f_var + y_noise_var
    y_n1_sd = np.sqrt(y_n1_var)

    # (n_ymax, n_x) matrix, each row is one vector of y-values for the n_x
    # locations generated from the model using current data
    z_ymax = np.random.normal(size=(n_ymax, y_mean.shape[0]))
    y_funcs = y_mean.T + z_ymax @ chol_k.T  # (n_ymax, n_x)
    y_n_funcs = y_funcs + y_noise_std * np.random.normal(size=(y_funcs.shape))

    # (1, n_yn1) vector of z-scores for the y_n1 values to be computed later
    z_yn1 = gaussian_bin_centers(n_yn1).reshape(1, n_yn1)

    # output tensors to be filled with each minibatch and concat at the end
    y_n1_output = []        # (n_x, n_yn1)
    y_funcs_output = []     # (n_x, n_yn1, n_ymax, n_x)
    y_max_output = []       # (n_x, n_yn1, n_ymax)

    for batch_idx in batch_idx_subsets:

        # generate y_{n+1} values for each x in this batch
        y_mean_b = y_mean[batch_idx, :]  # (bs, 1)
        y_sd_b = y_n1_sd[batch_idx, :]   # (bs, 1)
        y_n1_b = y_mean_b + y_sd_b * z_yn1  # (bs, n_yn1)

        # (bs, n_x) each row is the delta to adjust a sample fun for one x in batch
        fn_delta = y_cov[batch_idx, :] / y_n1_var[batch_idx, :]

        # (bs, n_ymax), get the y-values from the sample funs at x locs in this batch
        y_n_funcs_b = y_n_funcs[:, batch_idx].T

        # (bs, n_yn1, n_ymax) <- (bs, n_yn1, 1) - (bs, 1, n_ymax)
        # for each x in batch, get diff between
        #     (1) sampled funcs at x and
        #     (2) sampled y_n1 vals at x
        y_diffs =  y_n1_b[:,:, None] - y_n_funcs_b[:, None, :]

        # (bs, n_yn1, n_ymax, n_x)
        y_funcs_b = (
            y_funcs[None, None, :, :] +       # ( 1,     1, n_ymax, n_x)
            (
                y_diffs[:, :, :, None] *      # (bs, n_yn1, n_ymax,   1)
                fn_delta[:, None, None, :]    # (bs,     1,      1, n_x)
            )
        )
```

```
        y_n1_output.append(y_n1_b)
        y_funcs_output.append(y_funcs_b)

    y_n1_output = np.concatenate(y_n1_output, axis=0)
    y_funcs_output = np.concatenate(y_funcs_output, axis=0)
    y_max_output = y_funcs_output.max(axis=3)

    return y_n1_output, y_funcs_output, y_max_output, y_funcs
```