

Nucleus Developer Notes

Published Jul 27, 2026

Correspondence to

Joseph Lozier
joseph@findwhatmatters.ai

Open Access

Copyright © 2026 Lozier. This is an open-access article distributed under the terms of the [Creative Commons Attribution 4.0 International](#) license, which enables reusers to distribute, remix, adapt, and build upon the material in any medium or format, so long as attribution is given to the creator.

Open Source

The materials and processes described in this article are distributed under the terms of the [CERN Open Hardware License Permissive 2.0](#), which enables reusers to make and convey any component or device that apply the information made explicitly available under this license, so long as all notices are retained.

Batch Bayesian Optimization

Joseph Lozier  

Abstract

Bayesian optimization is a powerful tool for assisting scientific discovery, but in its basic form experiments are assumed to run one at a time. We discuss possible modifications to the recipe generation step of the Bayesian optimization loop to accommodate laboratory requirements that batches of experiments be run simultaneously. Approaches are visualized on a simple, low-dimensional test function.

Keywords Bayesian Optimization

Note

This article was written by a **human author**.

1. INTRODUCTION

A ubiquitous scientific and engineering problem is finding values for the dozens or more parameters of a novel process, fine tuning of which may be required to produce anything at all. Typically, the underlying behavior is non-linear and noisy; this is a recipe for a slow, expensive development campaign. Bayesian optimization can speed up the cycle of recipe proposal, experimentation and measurement, as can high-throughput batch-based laboratory techniques. But combining the two requires special techniques, which are the subject of this note.

2. THE ROSEN BROCK FUNCTION

To illustrate, we'll use a standard test landscape for (non-probabilistic) optimization, the Rosenbrock function, given by $f(x, y) = (x - a)^2 + b(x^2 - y)^2$ (H. H. Rosenbrock [1]). It has a key property in common with typical difficult real-world problems: a narrow and non-linear valley around the true minimum. For the standard parameter choices $a = 1, b = 100$, the valley is approximately fifty times wider in one direction than the other. This aspect ratio is not unrealistic; think of a process where, say, a precise effective stoichiometric ratio must be maintained (but some non-linearity—a side reaction, perhaps—makes direct scaling difficult). For clearer visualization, here we use $b = 10$, leading to an aspect ratio around the sole minimum at $x = 1, y = 1$ of approximately sixteen.

Another compromise for visualizability is the choice of a function with only two input dimensions and one output dimension. A realistic process may have several outputs and dozens of inputs; knowing a priori a slice or projection onto a two-dimensional manifold that visualizes more than a small fraction of the underlying dynamics would be a rare luxury. Finally, i.i.d. Gaussian noise with standard deviation 0.1 has been added, exemplifying the key departure of real processes from standard numerical optimization: the probabilistic nature of experimentation. Such random error is typically irreducible (or reducible only at great expense) and separating it from signal is the crucial responsibility of the surrogate model in Bayesian optimization.

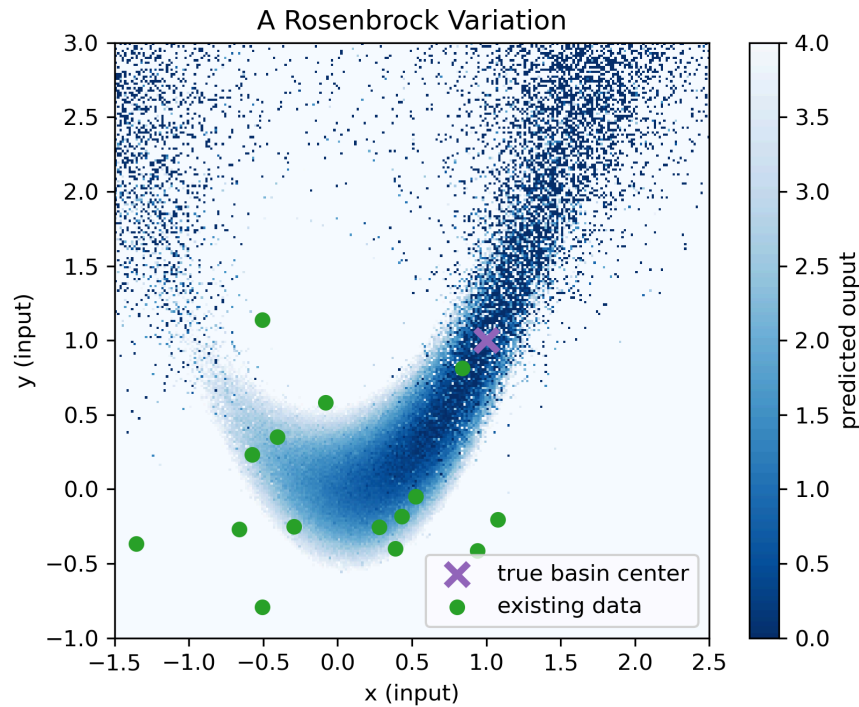


Figure 1. The problem setup before batch optimization. The blue noise represents the predictive uncertainty far from existing training data.

3. BATCH OPTIMIZATION

Bayesian optimization consists of modeling existing data with a probabilistic surrogate model, optimizing the controllable experimental parameters with respect to some acquisition function of the measurements, adding the results to the dataset and repeating until successful or some failure condition is reached (M. Siska, E. Pajak, K. Rosenthal, A. del Rio Chanona, E. von Lieres, and L. M. Helleckes [2]). As our focus is on the optimization step, for simplicity the model is based on two cheats. First, we fit the exact underlying functional form of the Rosenbrock function, a quartic polynomial in two dimensions; a practical model would need to be a more general approximator, perhaps a neural network with all the attendant complications. Second, we form the necessary probabilistic model by ensembling one hundred of these quartics, each trained on a sample of the underlying process. In reality, some sort of bootstrap would use the data much more efficiently.

Efficient use of data is an essential requirement for the scientific and engineering tasks in question, where experiments may be expensive, time-consuming and often both. Reducing the number of iterations until success in the Bayesian optimization loop (or, equivalently, the number of experiments) is the usual figure of merit, with the assumption being that modeling-optimization is redone for every experiment. Indeed, only a pathological strategy would be expected to perform better by not using all information available to it when proposing each recipe.

Practical considerations, however, can complicate the simple Bayesian optimization loop. Commonly, at least some portion of the experimental process can be run in batches to save time, cost or both. For applications that require a human in the optimization loop, batching proposals for expert review can save effort, even if the physical experimentation is essentially sequential. Consider, for example, queuing several assays to run overnight while the domain

expert is unavailable. The tradeoff, of course, is in information: the design of each experiment in the batch can't be based on the results of any of the others, though the cost-per-experiment improvement typically dominates the experiments-until-success increase.

In some domains, even higher efficiency gains can be achieved with techniques that are fundamentally parallel, such as the ubiquitous use of microplates in life science, or, say, running multiple different samples in the same oven. A common tradeoff in this situation (again, well worth it for the throughput gain) is that the experimental parameters of batch members may not be fully independent of each other as an unavoidable consequence of the throughput improvement. Samples in the same oven might have their composition freely chosen, but one heating curve must be chosen for the entire batch. A more complicated example of intra-batch constraints is a liquid handling robot that can prepare a different reaction in each of the dozens or hundreds of wells in a microplate, but must do so by combining a limited number of working solutions.

4. GREEDY METHODS

It remains to explore actual methods for proposing experiments in batches. A common class of approaches are "greedy" or "myopic:" one recipe is generated in the usual way, some modification is made to the setup to avoid redundantly generating the same point, and this is repeated until the batch is complete (Wilson, et al., 2018). Perhaps the simplest of these approaches is the "constant liar" algorithm, so-named because generated batch members are temporarily added back to the dataset with fictional measurements. The fictional measurements are some pre-chosen constant representing a failure to meet the overall Bayesian optimization goals.

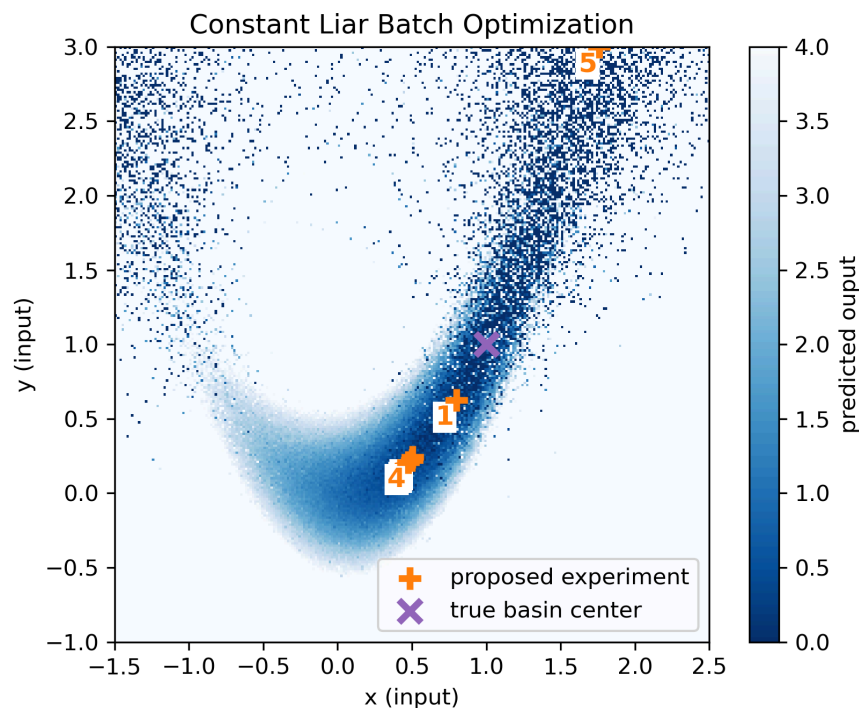


Figure 2. The constant liar algorithm, with the lie chosen to be 1.0. Notice the second, third and fourth recipes are nearly identical; only by the fifth does the optimization result change, this time going all the way to the boundary. This behavior is typical.

The behavior of the constant liar is strongly dependent on the particular choice of constant: too pessimistic (far from goals) and the algorithm over-explores, too optimistic and the algorithm over-exploits. This tuning is complicated in the very common case of multiple outputs, especially when there are multiple ways for an experiment to fail to hit the goals. Furthermore, the introduction of fictitious data with such a specific structure can wreak havoc on models that use cross-validation to choose regularization hyperparameters, especially with small datasets: it becomes very hard to statistically reject the hypothesis that the true function is a constant (the fake data points) plus a large amount of noise (the true signal from the real data points).

An alternative strategy to avoid redundancy in greedy batch generation is to employ some sort of distance-based rejection or penalty explicitly. Due to the curse of dimensionality, simple metrics like Euclidean distance lose their usefulness; approaches with some awareness of the correlations in the inputs, like Mahalanobis distance are better choices. Even better is an output-distribution based distance; a probabilistic model might provide a correlation in the uncertainty between the predictions at two points; this can be used to estimate to what degree the expected information from measuring the two points would overlap.

The distance metric enters as either a soft penalty term during optimization or a hard threshold in a sort of acceptance-rejection sampling. Both require choosing at least a “strength” free parameter, and introduce complications for optimization, especially for large batches. The Swiss-cheese landscape of an optimization problem with “holes” where existing points must be avoided can present difficulties for numerical optimizers, like adding numerous local minima. For rejection sampling with a hard threshold, large batches can lead to abysmal acceptance ratios; tricks like randomly decreasing model fidelity (in a kind of Thompson-like sampling) can combat this.

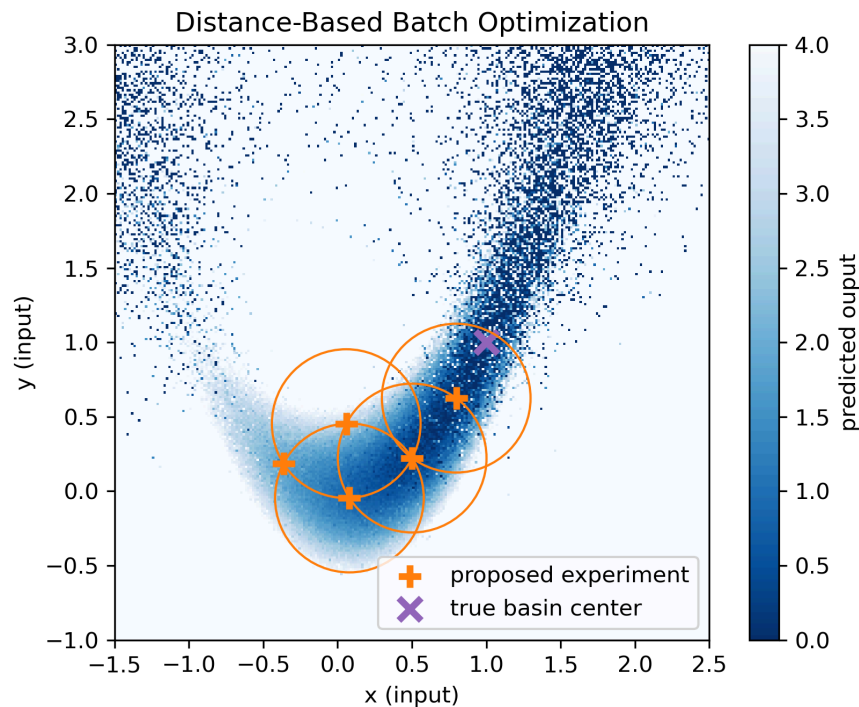


Figure 3. A Euclidean distance based hard constraint with threshold chosen at 0.5, represented by circles. Notice that later suggestions, which are seeded near the origin, “back-up” behind the first.

Intra-batch constraints are in general difficult to handle greedily. For simple enough constraints, an ad hoc approach can work. For example, in the example of a batch that must share identical heating curves, the curve might be optimized freely for the first batch member (simultaneously with its other parameters) then simply frozen for the rest of the batch. For more complicated interdependencies we must introduce parallel batch optimization.

5. PARALLEL METHODS

Parallel batch optimization is conceptually simple: treat all the parameters of all the batch members as one large, flat vector and optimize them simultaneously (Slautin and Kalinin, 2026). Intra-batch constraints become regular constraints that can be handled by standard techniques in numerical optimization such as Lagrange multipliers. The acquisition function chosen is usually an extension of a standard Bayesian-optimization acquisition to the batch. For example, the probability of improvement of a single experiment can be replaced with the probability that at least one batch member improves on the best measurements already present in the dataset. Shifting the complexity onto the numerical optimization algorithm has a cost, of course. Even with low-memory, approximate-Hessian techniques like L-BFGS, optimizing in bk dimensions (where b is batch size and k is the number of parameters per experiment) can quickly become computationally intractable if both factors are large. Another complication is the convexity of the optimization landscape. Consider, for example, that if the acquisition is symmetric on batch members and no two batch members are the same, each local minimum is one of a set of $b!$ near-identical minima because the batch members can be permuted.

Finally, we arrive at manifold-based batch optimization. The core idea is to somehow choose a manifold of experimental parameters, then sample that manifold. This can be effective when

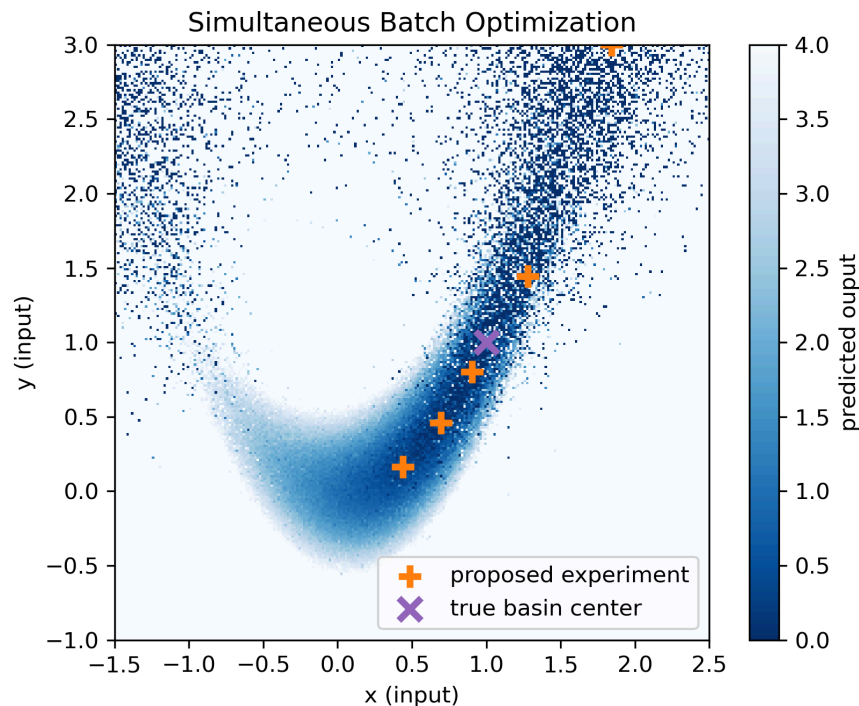


Figure 4. Simultaneous optimization. The acquisition function is the minimum prediction across the proposed batch, averaged across the ensemble. Qualitative performance is good, but this optimization took roughly six times the computation of the other three algorithms.

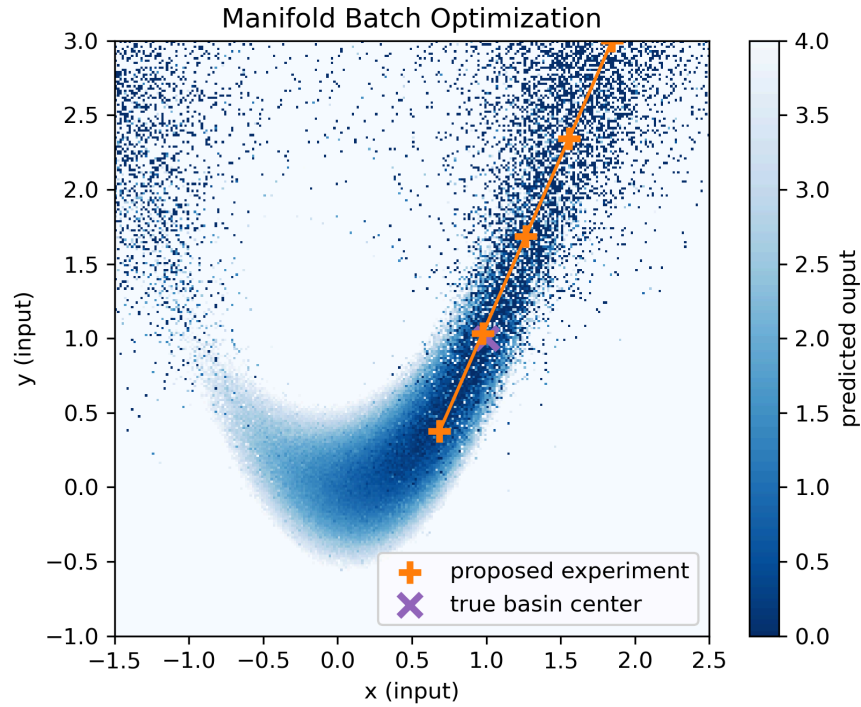


Figure 5. Sampling on a line segment. The same acquisition as above is used, but only the endpoints are optimized; the remaining three points are forced to be equally spaced between them.

each batch is expensive but experiments within each batch are plentiful. If the batch size is large compared to the size of the manifold, tackling the manifold-sampling step with, say, a generic low-discrepancy sampler (not using a model at all) may be sufficient. A function assigning an acquisition value to each manifold is required. This may be the same kind of batch acquisition described above in the non-manifold case; evaluation over the manifold can be done with a Monte Carlo method. In carefully constructed cases the acquisition may be an integral over the manifold that can be evaluated in closed form.

Manifold batch optimization reduces the choice of many batch elements to the choice of relatively few manifold parameters, such as the positions in input space of the vertices of a simplex; further simplification might be achieved by reasoning that if the manifold is to be as intrinsically-voluminous as possible, its vertices should be on the boundary of allowed experimental recipes. The true strength of the manifold approach is that a well-constructed manifold can enforce a particular intra-batch constraint. The simplex vertices described above might be the working solutions of a liquid-handling robot; the interior of the simplex is then exactly those mixtures that can be made from the working solutions. When not required, sampling a lower-dimensional manifold is less than ideal when considering the overall Bayesian optimization loop: a dataset structured into manifolds introduces correlations that hamper a model's effort to disentangle different effects compared to a dataset in general position.

6. CONCLUSIONS

We've outlined a few representative batch optimization strategies as well as their various tradeoffs. The choice of strategy for a single scientific application may be made on an ad hoc basis (especially to adapt to complicated intra-batch constraints). These techniques also

provide a starting point for the design of a general batch-optimization strategy for an AI co-scientist.

REFERENCES

- [1] H. H. Rosenbrock, "An Automatic Method for Finding the Greatest or Least Value of a Function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, Mar. 1960, doi: [10.1093/comjnl/3.3.175](https://doi.org/10.1093/comjnl/3.3.175).
- [2] M. Siska, E. Pajak, K. Rosenthal, A. del Rio Chanona, E. von Lieres, and L. M. Helleckes, "A Guide to Bayesian Optimization in Bioprocess Engineering," *Biotechnology and Bioengineering*, vol. 123, no. 4, pp. 805–830, Jan. 2026, doi: [10.1002/bit.70129](https://doi.org/10.1002/bit.70129).