Olympus: a benchmarking framework for noisy optimization and experiment planning

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
23 October 2020

REVISED
8 February 2021

ACCEPTED FOR PUBLICATION
11 March 2021

PUBLISHED
12 July 2021

PAPER

Olympus: a benchmarking framework for noisy optimization and experiment planning

Florian Häse[1,2,3,4,5,9], **Matteo Aldeghi**[2,3,4,9], **Riley J Hickman**[3,4], **Loïc M Roch**[2,3,4,5], **Melodie Christensen**[6,7], **Elena Liles**[7], **Jason E Hein**[7] and **Alán Aspuru-Guzik**[2,3,4,8,*]

[1] Department of Chemistry and Chemical Biology, Harvard University, Cambridge, MA 02138, United States of America
[2] Vector Institute for Artificial Intelligence, Toronto, ON M5S 1M1, Canada
[3] Department of Computer Science, University of Toronto, Toronto, ON M5S 3H6, Canada
[4] Chemical Physics Theory Group, Department of Chemistry, University of Toronto, Toronto, ON M5S 3H6, Canada
[5] Atinary Technologies Sàrl, Lausanne VD 1006, Switzerland
[6] Process Research and Development, Merck & Co., Inc., Rahway, NJ, United States of America
[7] Department of Chemistry, The University of British Columbia, Vancouver, BC V6T 1Z1, Canada
[8] Lebovic Fellow, Canadian Institute for Advanced Research, Toronto, ON M5G 1Z8, Canada
[9] These authors contributed equally.
[*] Author to whom any correspondence should be addressed.

E-mail: alan@aspuru.com

Keywords: reaction optimization, experiment planning, probabilistic modeling, autonomous experimentation

Supplementary material for this article is available online

Abstract

Research challenges encountered across science, engineering, and economics can frequently be formulated as optimization tasks. In chemistry and materials science, recent growth in laboratory digitization and automation has sparked interest in optimization-guided autonomous discovery and closed-loop experimentation. Experiment planning strategies based on off-the-shelf optimization algorithms can be employed in fully autonomous research platforms to achieve desired experimentation goals with the minimum number of trials. However, the experiment planning strategy that is most suitable to a scientific discovery task is *a priori* unknown while rigorous comparisons of different strategies are highly time and resource demanding. As optimization algorithms are typically benchmarked on low-dimensional synthetic functions, it is unclear how their performance would translate to noisy, higher-dimensional experimental tasks encountered in chemistry and materials science. We introduce Olympus, a software package that provides a consistent and easy-to-use framework for benchmarking optimization algorithms against realistic experiments emulated via probabilistic deep-learning models. Olympus includes a collection of experimentally derived benchmark sets from chemistry and materials science and a suite of experiment planning strategies that can be easily accessed via a user-friendly Python interface. Furthermore, Olympus facilitates the integration, testing, and sharing of custom algorithms and user-defined datasets. In brief, Olympus mitigates the barriers associated with benchmarking optimization algorithms on realistic experimental scenarios, promoting data sharing and the creation of a standard framework for evaluating the performance of experiment planning strategies.

1. Introduction

Optimization tasks are ubiquitous across science, engineering, and economics. They typically involve the identification of specific choices for controllable parameters under which a system of interest yields a desired response. The development of efficient strategies that lead to the discovery of such optimal parameter choices is of significant importance and has long been of interest to many scientific communities. Selecting an appropriate optimization strategy for a problem with unknown structure is non-trivial given that a single, overall superior strategy does not exist [1, 2]. Specifically, the qualities of a single optimization strategy

including convergence, computational demand, or requirements on the function to be optimized, could be ideal for some applications but render the same strategy inapplicable to other tasks. Understanding the challenges of optimization tasks in specific domains and the behavior of different algorithms for such tasks is essential to the development of efficient search strategies that are suitable to the considered application. Empirical assessments of the performance of different optimization strategies on realistic and domain-relevant scenarios is thus of paramount practical relevance.

One aspect where optimization has recently gained increased attention is the digitization of scientific discovery with autonomous platforms [3–5]. The emergence of ever more sophisticated and reliable automated experimentation equipment in chemistry and materials science over the last decades has increasingly allowed for formulation of scientific discovery as an optimization task [6–8]. In this formulation, compositions of candidate materials and processing conditions to fabricate multi-component materials are optimized to reach desired goals with respect to the physical and chemical properties of the synthesized material. Key missions in these fields relate to the discovery of functional molecules and advanced materials to tackle societal challenges such as climate change, renewable energy, sustainability, or clean water, which can be directly approached by modifying the structures and compositions of candidate materials to optimize their physical and chemical properties [9].

Automated instrumentation is now being combined with data-driven optimization strategies to enable autonomous molecule and materials development in self-driving laboratories [10]. Autonomous experimentation leverages these data-driven strategies to suggest molecules or materials candidates that are synthesized and characterized by robotic platforms [10–13], with real-time feedback on the suggested candidates being collected in the form of physical or chemical measurements. In this vision, the experimentation process requires minimal human intervention once the experimental campaign has been defined. The integration of algorithmic experiment planners with robotic hardware into an autonomous platform has already been shown to substantially lower the development costs of organic photovoltaic materials [14], identify novel chemical reactions [15], yield unexpected findings for thin film technologies [16], the discovery of photocatalysts for hydrogen production from water [17], and mechanical design [18], amongst other applications.

Several different optimization strategies have already been used for automated scientific discovery. While some of these optimization algorithms have been designed for broad applicability across general optimization tasks, other approaches have been developed with the more specific goal of planning laboratory experiments and are based on assumptions about the expected experimental response surfaces. For example, design of experiments (DoE) constitutes a frequently employed strategy to identify optimal conditions for chemical reactions [19, 20], where the system of interest is probed on a grid of different parameter choices. Chemical reactions have also been optimized with the Snobfit algorithm [21–23], variants of the Simplex method [24–26], or even gradient-based strategies [26]. Bayesian optimization frameworks have been demonstrated on materials science applications, most often realized using Gaussian processes [27–29] or random forests [30].

While the experiment planning strategies deployed in the aforementioned examples enabled autonomous workflows, it is not clear whether they are the most efficient ones for the considered task. In fact, it has recently been reported that ill-chosen planners can increase the budget requirements for scientific discovery in the context of materials science by up to an order of magnitude [31]. Without comprehensive benchmarks, availability and ease-of-use might be the primary considerations behind the choice of experiment planning strategy, while other factors such as the speed of convergence or the computational demand are neglected. The lack of the ability to evaluate the effectiveness of different experiment planning strategies thus poses a major obstacle to the development of autonomous research platforms.

To resolve this challenge, we propose to benchmark experiment planning strategies on probabilistic models. These models can emulate noisy experimental responses after being trained on experimental data, as previously demonstrated in the context of multi-objective optimization with autonomous research platforms [32]. In particular, we suggest to use Bayesian neural networks (BNNs) due to their robustness, scalability and non-local generalization capabilities. The outcome (e.g. reaction yield, solubility, etc) of a specific set of experimental parameters (e.g. concentration, temperature, etc) can be emulated by drawing a predictive sample from the BNN, conditioned on these parameters. This approach provides a viable avenue to benchmarking experiment planning algorithms in the presence of noise and on realistic, experimentally-derived response surfaces.

Following this idea of emulating experimental response based on real data, we introduce Olympus, a comprehensive software package that provides the possibility to probe the performance of experiment planning strategies on emulated experimental surfaces collected from experiments in chemistry and materials science. Olympus implements a common interface to 18 different experiment planning strategies and thus simplifies the implementation of closed-loop autonomous workflows. Olympus further provides a

Table 1. Feature comparison of different benchmarking tools. This work introduces Olympus, which targets benchmark applications related to autonomous scientific discovery in chemistry and materials science.

Toolkit	Optimizer interfaces	Synthetic benchmarks	Noisy surfaces	Emulated experiments	Visualizations	Community contributions	Primary purpose
Coco [33]	No	Yes	Yes	No	Yes	No	Continuous optimization
OpenAI Gym [34]	No	Yes	Yes	No	Yes	No	Reinforcement learning
Sherpa [35]	Yes	No	No	No	Yes	No	Hyperparameter optimization
Optuna [36]	Yes	No	No	No	Yes	No	Hyperparameter optimization
Pygmo [37]	Yes	Yes	No	No	Yes[a]	No	Parallel optimization
Summit [38]	Yes	Yes	Yes	Yes	No	No	Experiment planning
Olympus	Yes	Yes	Yes	Yes	Yes	Yes	Experiment planning

[a] Limited capability.

collection of 10 experimental datasets for which emulators have been trained to serve as a standard set of benchmarks, and a collection of 23 analytical surfaces which can be modulated by different sources of stochastic noise. An automated benchmarking process that determines the most efficient planner for a given application is available. As such, Olympus provides the means to run comprehensive comparisons of novel optimization algorithms and planning strategies to existing ones, allowing to identify the strengths and limitations of individual tools for various scientific discovery tasks. Its capacity to construct probabilistic approximations to experimental surfaces from collected data, modeling both the expected response and the noise modulations, makes Olympus a realistic benchmark suite without the need for excessive and resource demanding experimentation.

In the following, we summarize the datasets and emulators available through Olympus as well as the experiment planning strategies for which intuitive yet flexible interfaces have been implemented. We further highlight the application programming interface of Olympus, demonstrate how individual planners can be accessed and comprehensive benchmarks constructed with only a few lines of code. We conclude by providing a performance baseline comprised of a uniform random search and invite the community to develop and demonstrate more efficient experiment planning strategies on the Olympus benchmarks.

2. Background and related works

While the goal of an optimization task is usually well defined, the setting in which this task is approached might differ from application to application. Thus, the applicability of optimization strategies to certain tasks can be assessed based on multiple criteria, which are designed to highlight strengths and shortcomings of individual strategies on the considered application.

Several benchmarks and software packages have been introduced for different applications and at various levels of accessibility. Prominent examples in the field of machine learning (ML) include the MNIST [39] and CIFAR-10 [40] datasets for image recognition, which have been made available through a selection of libraries and interfaces. In the field of chemistry, comprehensive collections of datasets such as MoleculeNet [41] or the QMx series [42–47] serve similar purposes. Frameworks such as GuacaMol [48] and Moses [49] offer benchmarking functionalities for *de novo* molecular design. These examples provide datasets which aim to model realistic abstractions of the targeted applications on which optimization algorithms can be benchmarked. Yet, the requirements of comprehensive benchmarking frameworks go beyond realistic use cases and also include: (1) intuitive interfaces to interact with these datasets, (2) interfaces to established algorithms to benchmark, (3) tools to store and analyze collected results, and (4) the flexibility to allow the community to extend the framework with additional datasets and algorithms.

Table 1 reports a set of currently available benchmarking toolkits for different applications. Coco is a platform for the systematic comparison of real-parameter global optimizers [33]. It provides benchmark function testbeds, experimentation templates which are easy to parallelize, and tools for processing and visualizing data generated by one or several optimizers. Coco focuses on runtime as the central performance measure and optimization tasks on continuous domains with dimensionalities beyond those typically

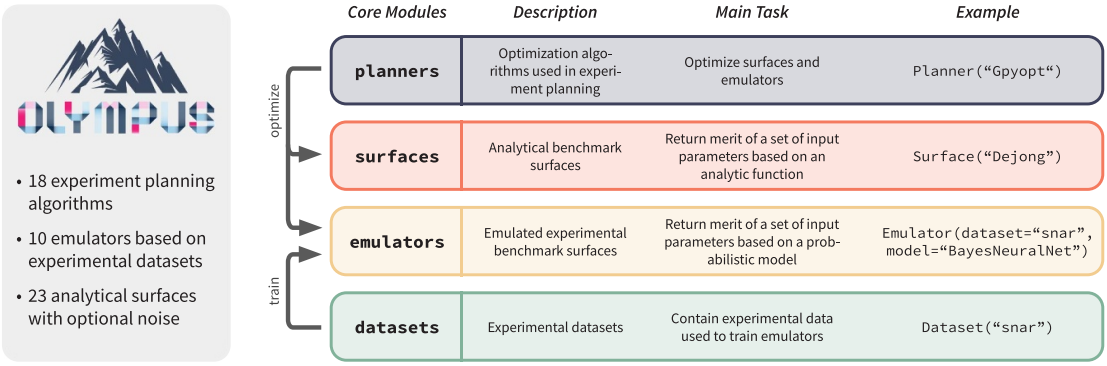

Figure 1. High-level overview of Olympus and its four core modules: (i) `planners`, which provide interfaces to common or custom optimization algorithms for experiment planning, (ii) `surfaces`, which constitute standardized interfaces to established synthetic benchmark surfaces, (iii) `emulators`, which describe a set of ML models trained to reproduce experimentally derived response surfaces encountered in chemistry and materials science, and (iv) `datasets`, which form a collection of experimental campaigns. All four core modules offer the possibility to implement and add custom methods and data.

encountered in chemistry and materials science. The OpenAI Gym offers a series of environments and tasks to test reinforcement learning algorithms [34]. Sherpa is a Python toolkit for hyperparameter tuning of machine learning models [35]. As such, Sherpa offers the automated optimization of hyperparameters via a choice of hyperparameter optimization algorithms including Bayesian optimization, evolutionary approaches and Bandit/Early-stopping schemes. Sherpa orchestrates the entire optimization process and results can be visualized in a comprehensive dashboard. However, Sherpa does not provide synthetic or noisy benchmark cases. Optuna is another toolkit that focuses on the optimization of hyperparameters for machine learning models [36]. In contrast to Sherpa, Optuna implements a *define-by-run* interface for a dynamic construction of search spaces. However, it also does not provide benchmark cases. Pygmo is a library for massively parallel optimization, which provides a unified interface to a number of gradient and heuristic based optimization algorithms, as well as to synthetic benchmark problems. Pygmo also provides algorithms and benchmarks for constrained and multi-objective optimization problems.

The aforementioned software packages have been developed with ML applications in mind. Summit, however, provides a selection of chemically motivated virtual benchmarks and a selection of experiment planning strategies. Although the application space of Summit is heavily focused on reaction optimization, it targets a realistic modeling of its use cases via physical and statistical models. In contrast, Olympus is tailored to the needs of optimization in a broader range of experimental disciplines, including self-driving laboratories and autonomous experimentation workflows. Specifically, it constitutes a framework to assess the algorithmic performance of data-driven experiment planning strategies in the context of autonomous experimentation for chemistry and materials science. It targets optimization tasks in chemistry and materials science, where the number of parameters to optimize is typically smaller than 10. To serve this purpose, Olympus provides interfaces to optimization algorithms commonly used for experiment planning tasks and offers interfaces to noisy emulators of experimental optimization tasks. In addition, the benchmarking capabilities of Olympus are open to be extended by the community who can contribute their own datasets (see section 4.6).

3. Package overview

Olympus is a modular software package that allows user interactions at different levels and can be used for data-driven experimentation as well as benchmarking experiment planning strategies. With this modularity, Olympus allows for both beginner and expert use and enables performing several optimization and benchmarking tasks in a few lines of code. Some common use-case scenarios are detailed in section 4, including (1) the use of different experiment planners for an autonomous workflow, (2) benchmarking an experiment planner on an emulator, and (3) constructing an emulator from a user-provided dataset. At the heart of Olympus are four modules, `planners`, `surfaces`, `datasets`, and `emulators`, which are highlighted conceptually in this section and in figure 1.

The `planners` module (see section 3.1) provides a consistent interface to 18 different experiment planning strategies via its core `Planner` class. Olympus translates a standardized access protocol to the interfaces of individual planners, making it easy to switch the experiment planning strategy of an

Table 2. List of algorithms available in Olympus. Convergence is categorized into *global* (converges to global optimum), *global** (does not necessarily converge to global optimum but can overcome local minima), *local* (does not necessarily converge to global optimum and does not overcome local minima).

Planner	Strategy	Convergence	Derivative-free
GPyOpt [50]	Bayesian	Global	Yes
HyperOpt [51–53]	Bayesian	Global	Yes
Phoenics [54]	Bayesian	Global	Yes
Genetic [55]	Evolutionary	Global*	Yes
CMA-ES [56, 57]	Evolutionary	Global*	Yes
Particle Swarms [58, 59]	Evolutionary	Global*	Yes
Differential Evolution [60]	Evolutionary	Global*	Yes
Steepest Descent [61, 62]	Gradient-based	Local	No
Conjugate Gradient [62, 63]	Gradient-based	Local	No
LBFGS [64–66]	Gradient-based	Local	No
SLSQP [67]	Gradient-based	Local	No
Grid Search [68–70]	Grid-like	Global	Yes
Latin Hypercube [68–70]	Grid-like	Global	Yes
Sobol Sequence [71]	Grid-like	Global	Yes
Random Search	Grid-like	Global	Yes
Snobfit [72]	Heuristic	Global	Yes
Basin Hopping [73]	Heuristic	Global	Yes
Simplex [74]	Heuristic	Local	Yes

autonomous workflow. This module also provides the basis for integrating customized algorithms into the package. Available planners are listed in table 2.

The `surfaces` module provides a set of synthetic response surfaces, which are functions commonly used to evaluate and compare the performance of optimization algorithms. Similar to the `planners` module, a convenient `Surface` class allows for easy retrieval of the desired analytical surface. Available surfaces are listed in table 4. While these surfaces return deterministic function evaluations by default, it is possible to pass a noise object that results in stochastic evaluations, as shown in section 4.2.

The `datasets` module in Olympus offers 10 core experimentally derived datasets from chemistry and materials science. These datasets vary in size and represent optimization tasks with dimensionalities from 3 to 6. The core class of this module is `Dataset`, which allows for retrieval and manipulation of the desired dataset. Available datasets are listed in table 3. Users can also load their own dataset, which can then be used to benchmark experiment planning strategies for the specific problem of interest. Furthermore, users can share their datasets with the community by uploading it to the Olympus repository of user-provided datasets at https://github.com/aspuru-guzik-group/olympus_datasets. Any user can then download these additional datasets to be used in Olympus via the same interface used for the core datasets.

The `emulators` module provides access to probabilistic models trained on the core Olympus datasets, reproducing the experimental responses of the corresponding experiments. With its `Emulator` class, this module also offers a high-level interface for the training of such probabilistic models on user-provided datasets. In this spirit, emulators constitute stochastic response surfaces resembling those encountered in real-life applications, thus allowing to benchmark experiment planning strategies on close-to-reality optimization tasks.

3.1. Summary of included planners

This section details the types of experiment planning strategies and algorithms available in Olympus and listed in table 2. More information about each specific planner can be found on the online documentation.

Gradient approaches use derivative information (gradient or Hessian) at the current point to determine the location of the next point to be evaluated. Such strategies are efficient on convex optimization problems, but are not guaranteed to find the global optimum on non-convex surfaces [75, 76]. Most gradient-based approaches condition both the stepping direction and the step size on the local gradient. The numerical approximation of gradients generally poses a challenge in the context of experimentation where the response surface is subject to noise. Nevertheless, gradient-based search strategies have been reported for the optimization of some chemical processes [77].

Grid-like searches constitute a more common approach to experiment planning [68–70]. These strategies define a set of selected parameter points in the parameter space to be evaluated at the start of the optimization campaign. At every step of the campaign, the next point to be evaluated is chosen deterministically. Although grid-like searches mitigate the locality issue of gradient approaches and can reliably identify global optima, their cost scales exponentially with the dimensionality of the parameter space. Alternatives to standard full

grid approaches involve the use of low-discrepancy sequences, such as Latin Hypercube or Sobol Sequence, to sample more effectively high dimensional spaces. The discrepancy of a sequence is considered to be low if the proportion of points falling into an arbitrary subset of the considered parameter domain is roughly proportional to the measure of this subset. Low-discrepancy sequences are also known as quasi-random sequences and are commonly used to finding characteristic functions of probability density functions, higher-order moments of statistical distributions, and integration and sampling of high-dimensional deterministic functions. Random Search reduces the correlation between consecutive proposals even further and has been shown to be particularly effective in higher-dimensional search spaces [52, 78, 79].

Evolutionary algorithms are population and heuristic-based approaches inspired by biological evolution [80–84]. Each individual in the population represent a point in the search space, and their fitness corresponds to the objective evaluated at that point. Evolutionary strategies, like CMA-ES, Particle Swarms, and Differential Evolution, evolve a population of candidate solutions simultaneously and generate new candidates based on some heuristics. The population is frequently updated, with better candidates replacing worse performing candidates [85]. Genetic algorithms constitute a subclass of evolutionary strategies which mimic mechanisms such as reproduction, mutation, recombination, and selection to iteratively improve the fitness of a population.

Other heuristic-based approaches are not inspired by biological evolution specifically. For example, Basin Hopping is a two-step approach that uses both local and global searches and is inspired by the energy landscape of atom clusters [73]. Snobfit also combines both local and global approaches and the strategy was designed with the goal of addressing a number of practical challenges [72]. Finally, the Simplex algorithm by Nelder and Mead exploits the geometry of simplexes to define an update rule that proposes new points in a downhill direction [74].

Bayesian optimization methods are sequential, model-based approaches for the global optimization of black-box functions [86–88]. The function to be optimized is approximated by a surrogate model that is refined as more data is collected. Based on this model, an acquisition function that evaluates the utility of candidate points can be defined, leading to the balanced exploration and exploitation of the search space of interest. Similar to evolutionary strategies, no gradient information is required and they can be used for the global optimization of black-box functions. What distinguishes Bayesian optimization approaches are primarily the surrogate model and acquisition functions used. GPyOpt uses a Gaussian process to model the objective function [50], Phoenics adopts a mixture of Gaussian kernels [54], and HyperOpt uses a tree-structured Parzen estimator [51–53].

We note that the algorithms available in Olympus present different computational scalings with respect to the number of samples collected and the dimensionality of the optimization domain. These algorithmic and implementation aspects result in different runtimes and memory requirements, which ultimately affect the applicability of each algorithm to different problems. However, given the typical problem dimensionalities ($<$10 parameters) and runtimes ($>$10 min per experiment) encountered experimentally, the computational cost of any of the algorithms above described is not expected to be of significant importance in autonomous workflows.

3.2. Summary of included datasets

Olympus ships with a total of ten core datasets collected from experiments spanning chemistry and materials science. The datasets have either been collected from the literature or were generated in-house. With these datasets, Olympus can construct experiment emulators using probabilistic machine learning models, notably BNNs, to emulate the overall response surface of the considered experiment for an arbitrary choice of parameter values (see section 3.3 for details). As such, the provided datasets constitute the basis for realistic benchmarks of experiment planning strategies.

Table 3 summarizes core information about each dataset and further details are provided in the supplementary information (available online at stacks.iop.org/MLST/16/035021/mmedia) (see section 1). All datasets are collected from experimental campaigns with three to six independently controllable parameters, one property of interest, and contain from a few tens to more than 1000 data samples. Five datasets are related to the optimization of organic chemistry reactions, one is derived from the calibration of analytical chemistry instrumentation, two address the identification of polymer blends of photovoltaic materials with favorable photodegradation properties, and two are related to the identification of the colorant mixture displaying a chosen target color. This core set of datasets can be extended by community datasets contributed from individual research groups. Details are provided in section 4.6.

3.3. Summary of included emulators

The experiment emulators offered through Olympus provide a core functionality to benchmarking data-driven experiment planning strategies: the opportunity to query the response of a quasi-experimental

Table 3. List of the core OLYMPUS datasets. Datasets contributed in this work are marked with *.

Olympus label	Topic	Discipline	# Data points	# Parameters
`alkox`*	Alkoxylation reaction	Organic chemistry	208	4
`colors_bob` [89]	Colorant mixture with 3D printed robot	Colorimetry	241	5
`colors_n9` [89]	Colorant mixture with commercial robot	Colorimetry	102	3
`fullerenes` [90]	Synthesis of o-xylenyl C_{60} adducts	Organic chemistry	246	3
`hplc` [89]	Calibration of an automated HPLC	Analytical chemistry	1386	6
`benzylation` [91]	N-benzylation reaction	Organic chemistry	73	4
`photo_pce10` [12]	Photostability of organic photovoltaics with PCE10 polymers	Materials science	1040	4
`photo_wf3` [12]	Photostability of organic photovoltaics with WF3 polymers	Materials science	1040	4
`snar` [91]	Nucleophilic aromatic substitution	Organic chemistry	66	4
`suzuki`*	Carbon-carbon cross-coupling reaction	Organic chemistry	247	4

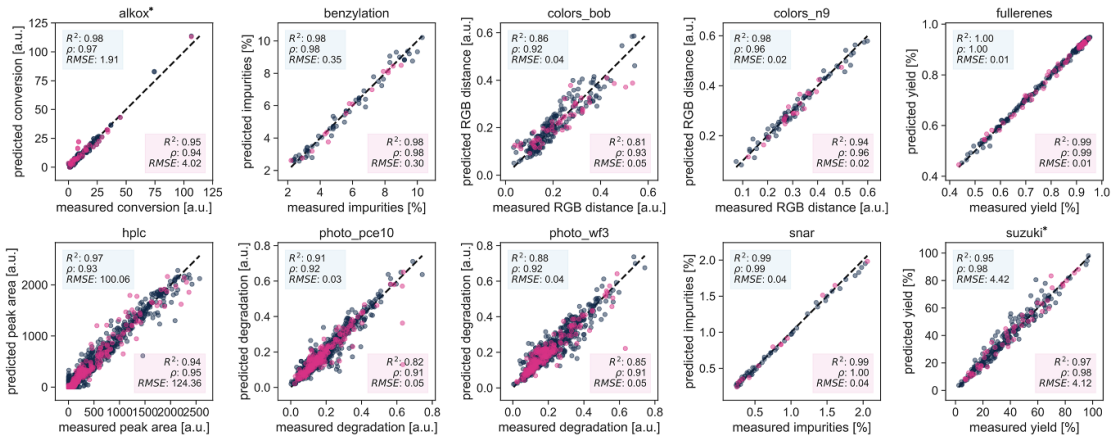

Figure 2. Parity plots with experimental versus predicted target values for all emulators based on BNN models. Performance on the training set (80% of data; blue markers) is shown in the top-left corner of each plot and on blue background; performance on the test set (20% of data; pink markers) is shown in the bottom-right corner of each plot and on pink background. R^2 is the coefficient of determination, ρ is the Spearman's rank correlation coefficient, and *RMSE* is the root-mean-square error. Emulators trained on datasets introduced in this work are marked with *.

surfaces inexpensively within milliseconds. Balancing robustness and prediction accuracy on the data-scarce datasets reported in section 3.2, we construct OLYMPUS emulators from feedforward fully connected BNNs. BNNs constitute probabilistic machine learning models which, contrary to standard neural network (NN), define a distribution of possible target values conditioned on the input features. To this end, the conventional weight and bias parameters of standard NNs are modeled as distributions themselves and the BNN is trained via Bayesian inference. While, in theory, weights and biases can be modeled by any valid distribution, in practice the distributions are often explicitly modeled via a set of parameters, such as the location and scale of a normal distribution. This approximation can greatly accelerate inference computations and make the training of a BNN overall computationally tractable. In addition to probabilistic, BNN-based emulators, we also provide deterministic, NN-based ones. These emulators return the same target value given a set of input features.

Emulators are trained on 80% of the data and tested on 20% using a random split. The training set is furthermore split into training and validation sets for cross-validation. By default, five folds are used, but users can choose how many folds to use when creating their own emulators. The test set is used to probe the generalizability of the emulator. Model performances on both training and test sets are shown in figure 2 for BNN-based emulators, and in figure S6 for NN-based emulators. Emulators are constructed with different

Table 4. Available analytical surfaces in Olympus.

Type	Property	Surfaces
Continuous	Convex	`Dejong`, `HyperEllipsoid`, `Zakharov`
	Non-convex	`AckleyPath`, `Branin`, `Levy`, `Michalewicz`, `Rastrigin`, `Rosenbrock`, `Schwefel`, `StyblinskiTang`
Piece-wise constant	Convex	`LinearFunnel`, `NarrowFunnel`
	Non-convex	`DiscreteAckley`, `DiscreteDoubleWell`, `DiscreteMichalewicz`
Mixture model	Non-convex	`GaussianMixture`, `Denali`, `Everest`, `K2`, `Kilimanjaro`, `Matterhorn`, `MontBlanc`

choices for hyperparameters, including the number of layers, the number of neurons per layer, activation functions, and others, which can be accessed directly through the `emulator` objects. Activation functions for the output layer have been chosen to satisfy physical constraints, such as positivity of the property of interest. Other hyperparameters have been manually selected to achieve promising prediction accuracies. We define emulators as accurate if they achieve a Spearman's rank correlation coefficient above 0.90 for both training and test sets, given that a monotonic relationship between predicted and measured values preserves the relative ranking of all extrema. Typical evaluations of trained emulators take less than 1 ms on a standard laptop. This cheap experiment emulation approach enable the large-scale querying of experimental responses and the rigorous benchmarking of data-driven experiment planning strategies.

3.4. Summary of included analytical surfaces

In addition to experimentally-derived benchmarks, Olympus provides a suite of analytical functions traditionally used to evaluate optimization algorithms (table 4). These functions include 11 analytic and smooth functions, such as the Branin and Rosenbrock functions, 5 piece-wise constant functions, and 6 Gaussian mixture model functions derived from a parent Gaussian mixture generator. This generator takes a number of dimensions as argument, and draws random means and covariances. By default, a full covariance matrix is drawn, but a diagonal matrix can also be requested. By fixing the random seed, the Gaussian mixture generator creates reproducible surfaces. In fact, the six Gaussian mixture models available have been obtained by fixing the random seed of each mountain-named surface (e.g. Everest) to the height of the respective mountain peak in meters (e.g. 8848).

For all analytical surfaces in Olympus, it is possible to specify noise to be added to the evaluations. In such a way, the output of these toy surfaces will be stochastic. A few commonly used noise functions, like Gaussian, uniform, and gamma-distributed noise, are already implemented and readily available in Olympus. However, custom types of noise can also be defined by the users and provided to the surface of interest, which will then return noisy evaluations. Note, that noise modulations are currently only supported for experimental responses. However, realistic experiments might also be subject to noise on the preparation of experimental parameters, such as the dispensing of desired amounts of chemicals or controlling the temperature in an experimental setup. In Olympus, users may add noise to the experimental parameters by taking advantage of the `Planners` interface.

Table 4 summarizes the synthetic benchmark functions available in Olympus. Further details as well as illustrations of the surfaces are reported in the supplementary information (see section 3).

4. Using Olympus

In this section, we detail the usage of Olympus on selected applications and use cases. A detailed documentation of the package is provided on GitHub [92].

4.1. Installation and dependencies

Olympus is available for download on GitHub [92] and can be installed via `pip` and `conda`.

```
# Option 1 (recommended): installation via pip
>> pip install olymp
# Option 2: installation via anaconda
>> conda install -c conda-forge olymp
# Option 3: installation from source
>> git clone https://github.com/aspuru-guzik-group/olympus.git
>> cd olympus
>> python setup.py develop
```

The installation requires Python 3.6+ with support for `numpy` and `pandas`. However, to access specific features of the package, such as running an emulator, using specific experiment planners, or plotting the results of completed campaigns, the installation of additional packages might be required. Details are provided in the documentation [92].

4.2. Evaluate analytical surfaces

The analytical surfaces in Olympus can be accessed via the `olympus.surfaces` module or the `Surface` function, with the latter loading a surface with default argument.

```
from olympus.surfaces import Michalewicz
surface = Michalewicz(param_dim=2, m=12)
# or, to load with default arguments
from olympus import Surface
surface = Surface("Michalewicz")
```

The above example defines a surface with deterministic output. However, noise can be added to have a surface instance that returns stochastic evaluations.

```
from olympus.surfaces import Dejong
from olympus.noises import GaussianNoise
noise = GaussianNoise(scale=0.5)
surface = Dejong(param_dim=2, noise=noise)
```

Surfaces can then be evaluated sequentially or in batches as follows.

```
# evaluate a single point in 2 dimensions
surface.run([0.5, 0.5])
>>> [[0.0]]
# evaluate a batch of 2 points in 2 dimensions
surface.run([[0.5, 0.5], [0.75, 0.75]])
>>> [[0.0], [3.16]]
```

4.3. Run a simulated campaign

The datasets available in Olympus can be accessed via the `Dataset` class, using the keyword associated with each dataset.

```
# load an Olympus dataset
from olympus import Dataset
dataset = Dataset("snar")
```

NN or BNN based emulators are already available in Olympus for all datasets provided. However, the user also has the freedom to train new emulators by customizing the models provided in the `olympus.models` module.

```
from olympus import Emulator
emulator = Emulator(dataset="snar", model="BayesNeuralNet")
# or customize the model
from olympus.models import BayesNeuralNet
model = BayesNeuralNet(hidden_depth=4, out_act="sigmoid")
emulator = Emulator(dataset="snar", model=model)
```

All algorithms described in the previous section can easily be accessed from the `olympus.planners` module or via the `Planner` function. While the former allows the user to choose specific settings for each planner, the latter loads them with default arguments.

```
from olympus.planners import Gpyopt
planner = Gpyopt(goal="minimize", model_type="GP_MCMC", acquisition_type="EI_MCMC")
# or, to load with default arguments:
from olympus import Planner
planner = Planner("Gpyopt", goal="minimize")
```

Once a planning algorithm and an emulator have been defined, it is possible to start a simulated optimization campaign using the `optimize` method.

```
emulator = Emulator(dataset="snar", model="BayesNeuralNet")
planner = Planner("Phoenics", goal="minimize")
campaign = planner.optimize(emulator=emulator, num_iter=50)
```

4.4. Train custom emulator

With Olympus you can create an `Emulator` in order to generate a custom emulated response surface for a new dataset. For instance, if you have data for a chemical reaction of interest, for which you would like to optimize the yield, you can load the dataset from a table as follows.

```
# load a custom dataset
from olympus import Dataset
import pandas as pd
mydata = pd.from_csv("mydata.csv")
dataset = Dataset(data=mydata, target_ids=['yield'])
```

After this step, you can load one of the available models from the `olympus.models` module and pass it to a new `Emulator` instance, which will allow you to cross-validate and train the emulator. Users can override default model hyperparameters by passing custom values as arguments to the `olympus.models` module. Once you obtain an emulator with satisfactory performance, you can save it.

```
from olympus.models import BayesNeuralNet
mymodel = BayesNeuralNet(hidden_depth=4, out_act="sigmoid")
emulator = Emulator(dataset=mydataset, model=mymodel)
emulator.cross_validate()
>>> ...

emulator.train()
>>> ...

emulator.save("my_new_emulator")
```

4.5. Test your planning algorithm

Olympus allows you to create a `Planner` implementing your own custom algorithm, such that you can test it against established methods on a set of challenging benchmarks. To create such `Planner` you just need to inherit from the `CustomPlanner` class and implement the `_ask` method. This method should return the next query point based on the algorithm's strategy. For instance, a random sampler can be implemented as follows.

```
from olympus.planners import CustomPlanner
from olympus import ParameterVector as PV
import numpy as np
class RandomSampler(CustomPlanner):
  def _ask(self):
    new_params = []
    for param in self._param_space:
      new_param = np.random.uniform(low=param['domain'][0], high=param['domain'][1])
      new_params.append(new_param)
    return PV(array=new_params, param_space=self.param_space)
```

The `_ask` method takes advantage of the `_param_space` attribute present in `CustomPlanner`, which is a list of dictionaries defining the parameter space over which to optimize. In addition, `_params` and `_values` contain the parameters and associated merits for all previous observations, respectively. These attributes will be needed for any algorithm in which the set of parameters proposed depend on the previous observations. Finally, note `_ask` returns a `ParameterVector` object, which can be instantiated with an array or dictionary of parameters.

In the above example, an `__init__` method is not specified. This is because the following default one is inherited from `CustomPlanner`.

```
def __init__(self, goal='minimize'):
    AbstractPlanner.__init__(**locals())
```

If you would like to initialize your own `Planner` with more options, you can expand upon the above `__init__` method. Note it is required to keep the argument `goal` and to initialize `AbstractPlanner` as above. A tutorial on the creation of custom `Planner` classes with further details on possible customization is available as part of the online documentation.

4.6. Download/upload community datasets

In addition to the set of core datasets distributed with Olympus, we allow users to share their own datasets with the community. These additional datasets are stored on GitHub and provide an extended set of benchmarks built by the autonomous experimentation community. Olympus provides intuitive command line tools to upload and download these datasets. For instance, to download the *excitonics* dataset and make it available to your local Olympus installation.

```
>> olympus download -n excitonics
```

After the download, the *excitonics* dataset will be available and you will be able to load it in the same way as the core datasets.

```
from olympus import Dataset
dataset = Dataset("excitonics")
```

Note that, for community-provided datasets, trained `Emulator` instances are not readily available, such that you will need to train the relevant `Emulator`.

```
from olympus import Dataset
from olympus.models import NeuralNet
dataset = Dataset("excitonics")
model = NeuralNet(hidden_depth=3, out_act="relu")
emulator = Emulator(dataset=dataset, model=model)

emulator.train()
>>> ...

emulator.save("excitonics_nn_emulator")
```

If you have a dataset that you think would be a useful benchmark for the community, you can upload it to the Olympus pool of datasets using the Olympus command line tools as follows.

```
>> olympus upload -n <dataset_name> -p <dataset_path>
```

4.7. Plotting benchmark results

Results collected in several campaigns can be plotted automatically via a comprehensive plotting interface. Plots are generated from a `Database` object, like the one generated by Olympus when running a benchmark campaign. The following example illustrates the generation of a plot that illustrates the results of the executed benchmark. The generated plot is shown in figure 3.

```
# plot collected results
from olympus import Olympus, Plotter
olymp = Olympus()
olymp.benchmark(num_iter=100,dataset ='fullerenes',
planners=['random','simplex'], num_ind_runs=10)
>>> ...
Plotter().plot_from_db(olymp.database)
```

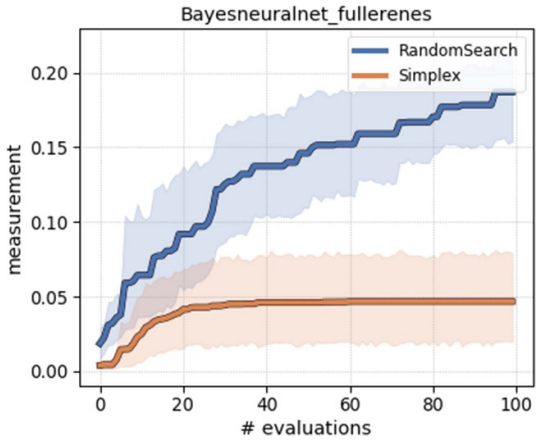

Figure 3. Plot generated with the `Plotter` module within Olympus illustrating the average best measurements collected during five independent runs with Random Search and Simplex over 100 iterations on the `fullerenes` emulator.

Further details on the capabilities and the usage of the `Plotter` module are reported in the online documentation [92].

5. A random search baseline

The promise of data-driven strategies to identify desired parameter choices for experimental setups in closed-loop workflows is based on their capacity to condition design choices on feedback collected from previous experiments. The performance of a data-driven experiment planning strategies can, for example, be quantified via the number of experiments required to locate parameter values which yield the desired experimental outcomes. In this paragraph, we aim to provide a baseline for the performance of data-driven experiment planning strategies which can indicate the degree of difficulty that each constructed emulator poses to a planner.

We construct the baseline by probing the performance of the random search strategy on each of the emulators. Random search as an experiment planning approach can be considered to be a naive strategy as it does not leverage any feedback collected in previous experiments. Parameter choices for future experiments are generated by drawing random samples from uniform distributions supported within the allowed ranges of each of the parameters. As such, the suggested parameter values are independent from one another and are not influenced by any past measurements. Data-driven strategies for experiment planning that do condition their design choices on previous feedback are therefore expected to outperform the random search baseline. The magnitude by which random search is outperformed can be used as a proxy to quantify the efficiency of the planner for each emulator.

The provided baseline consists of 100 independent campaigns with 10 000 emulator evaluations per campaign for each of the emulators. Note that results from random baselines are not shipped with the software package and need to be downloaded separately. Random baselines are available on Github [92] and can be downloaded from there or via the Olympus command line interface.

```
# download random baseline
>> olympus baseline --get
```

All parameters are generated with the random search planner. The results of these baseline calculations can be accessed through Olympus as follows.

```
# load the baseline
from olympus import Baseline
base = Baseline()
summary = base.get('snar', kind='summary')
campaigns = base.get('snar', kind='campaigns')
database = base.get('snar', kind='db')
```

While the full traces of the random search baselines are available through Olympus, we suggest to compare the achieved feedback after a specified set of emulator evaluations. We propose to use [1, 3, 10, 30,

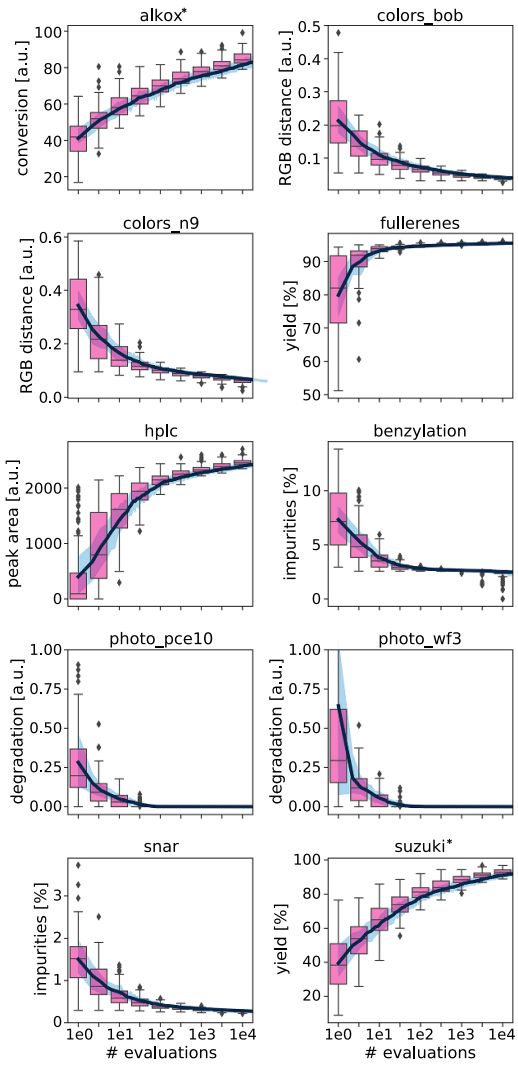

Figure 4. Performance of random search on all ten emulated surfaces in Olympus. We illustrate the best achieved property values up to the specified number of evaluations for all ten datasets. Solid blue lines show the average best values over 100 independent executions of the optimization starting from different random seeds. Box plots show the distribution of these 100 values at a few specific number of evaluations. Results obtained for emulators trained on datasets introduced in this work are marked with *.

100, 300, 1000, 3000, 10 000]. This choice is inspired by the fact that most experimental campaigns reported for autonomous experimentation platforms are limited to about 100 experiments. This set of evaluation numbers allows to estimate the performance of each planner in the regime of little data (∼10 evaluations), medium data (∼100 evaluations), abundant data (∼1000 evaluations) and asymptotic behavior (∼10 000 evaluations).

The results of random search against all core Olympus datasets are illustrated in figure 4. Based on these results, we can identify a subset of the emulated surfaces for which random search reaches near-optimal property values in a small number of evaluations. This subset includes `photobleaching_pce10`, `photobleaching_wf3`, `colormix_bob`, `colormix_n9`, `snar`, and `hplc_n9`. Given that random search does not leverage any feedback from collected measurements for future decisions, these emulated surfaces might be considered to be the simpler cases for a more sophisticated experiment planner. The remaining surfaces, however, including `alkox`, `fullerenes`, `nbenzylation`, and `suzuki`, might pose a bigger challenge to experiment planners given that asymptotic property values are only achieved after a significant number of random evaluations or not even reached after 10 000 evaluations. Numerical values for the baseline are available through the Olympus package [92]. We hope that the results from this random search can serve as a baseline to compare the performance of different experiment planning strategies such as those already included in Olympus, but also new strategies developed by the community.

6. Conclusion

Standardized and challenging benchmarks are necessary to facilitate precise comparison between different approaches and allow scientific and technological advances to be quantified. Widely used benchmark sets like MNIST and CIFAR-10/100 [39, 40], which are comprised of images of hand-written digits and various objects and animals, respectively, have allowed to measure constant advances in machine vision, providing clear feedback to the community on the most promising research directions. MoleculeNet, a collection of quantum mechanical, physical, biophysical and physiological molecular properties, provides a similar example in the field of chemistry and biophysics [41]. Olympus constitutes an orthogonal set of benchmarks, with a focus on optimization and experiment planning in chemistry and materials science, as opposed to prediction. It provides a framework with the potential to spark and streamline the development of powerful algorithms and data-driven approaches aimed at efficient experiment planning. To this end, Olympus also provides intuitive interfaces to a variety of experiment planning strategies to simplify their implementation, deployment, and testing in autonomous discovery workflows. With every user being able to supply their own datasets through our standardized interfaces, Olympus also encourages the free exchange of data across the community and promotes the establishment of standard, reproducible optimization challenges. In summary, Olympus provides a unified framework for the deployment and testing of experiment planning strategies. We thus invite the community to take advantage of Olympus in the implementation and testing of novel approaches to autonomous workflows, as well as to share experimental data that can prove valuable in moving this exciting new field forward.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/aspuru-guzik-group/olympus.

Acknowledgments

The authors acknowledge generous support from Natural Resources Canada (NRCAN). F H acknowledges financial support from the Herchel Smith Graduate Fellowship and the Jacques-Emile Dubois Student Dissertation Fellowship. M A is supported by a Postdoctoral Fellowship of the Vector Institute. R J H gratefully acknowledges the Natural Sciences and Engineering Research Council of Canada (NSERC) for provision of the Postgraduate Scholarships-Doctoral Program (PGSD3-534584-2019). This work relates to Department of Navy Award (N00014-19-1-2134) issued by the Office of Naval Research. This work was supported by the Defense Advanced Research Projects Agency under the Accelerated Molecular Discovery Program under Cooperative Agreement No. HR00111920027 dated 1 August 2019. The content of the information presented in this work does not necessarily reflect the position or the policy of the Government. A A G would like to thank Dr Anders Frøseth for his support. All computations reported in this paper were completed on the computing clusters of the Vector Institute and the Odyssey cluster supported by the FAS Division of Science, Research Computing Group at Harvard University. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

ORCID iDs

Florian Häse https://orcid.org/0000-0003-3711-9761
Matteo Aldeghi https://orcid.org/0000-0003-0019-8806
Riley J Hickman https://orcid.org/0000-0002-5762-1006
Loïc M Roch https://orcid.org/0000-0003-1771-2023
Jason E Hein https://orcid.org/0000-0002-4345-3005
Alán Aspuru-Guzik https://orcid.org/0000-0002-8277-4434

References

[1] Wolpert D H and Macready W G 1997 *IEEE Trans. Evol. Comput.* **1** 67
[2] Droste S, Jansen T and Wegener I 2002 *Theor. Comput. Sci.* **287** 131
[3] Wilbraham L, Mehr S H M and Cronin L 2021 *Acc. Chem. Res.* **54** 253–62
[4] Krishnamurthy D, Weiland H, Barat Farimani A, Antono E, Green J and Viswanathan V 2018 *ACS Energy Lett.* **4** 187
[5] Aspuru-Guzik A, Lindh R and Reiher M 2018 *ACS Cent. Sci.* **4** 144
[6] Le T C and Winkler D A 2016 *Chem. Rev.* **116** 6107

[7] Houben C and Lapkin A A 2015 *Curr. Opin. Chem. Eng.* **9** 1
[8] Danielson E, Golden J H, McFarland E W, Reaves C M, Weinberg W H and Wu X D 1997 *Nature* **389** 944
[9] Tabor D P *et al* 2018 *Nat. Rev. Mater.* **3** 5
[10] Häse F, Roch L M and Aspuru-Guzik A 2019 *Trends Chem.* **1** 282–91
[11] Stein H S and Gregoire J M 2019 *Chem. Sci.* **10** 9640
[12] Coley C W, Eyke N S and Jensen K F 2020 *Angew. Chem., Int. Ed.* **59** 22858
[13] Coley C W, Eyke N S and Jensen K F 2020 *Angew. Chem. Int. Ed.* **59** 23414
[14] Langner S, Häse F, Perea J D, Stubhan T, Hauch J, Roch L M, Heumueller T, Aspuru-Guzik A and Brabeca C J 2020 *Adv. Mater.* **32** 1907801
[15] Granda J M, Donina L, Dragone V, Long D-L and Cronin L 2018 *Nature* **559** 377
[16] MacLeod B P *et al* 2020 *Sci. Adv.* **6** eaaz8867
[17] Burger B *et al* 2020 *Nature* **583** 237
[18] Gongora A E, Xu B, Perry W, Okoye C, Riley P, Reyes K G, Morgan E F and Brown K A 2020 *Sci. Adv.* **6** eaaz1708
[19] Reizman B J, Wang Y-M, Buchwald S L and Jensen K F 2016 *React. Chem. Eng.* **1** 658
[20] Ingham R J, Battilocchio C, Hawkins J M and Ley S V 2014 *Beilstein J. Org. Chem.* **10** 641
[21] Krishnadasan S, Brown R, Demello A and Demello J 2007 *Lab Chip* **7** 1434
[22] Walker B E, Bannock J H, Nightingale A M and deMello J C 2017 *React. Chem. Eng.* **2** 785
[23] Bédard A-C, Adamo A, Aroh K C, Russell M G, Bedermann A A, Torosian J, Yue B, Jensen K F and Jamison T F 2018 *Science* **361** 1220
[24] Fitzpatrick D E, Battilocchio C and Ley S V 2016 *Org. Process Res. Dev.* **20** 386
[25] Cortés-Borda D *et al* 2018 *J. Org. Chem.* **83** 14286
[26] McMullen J P and Jensen K F 2010 *Org. Process Res. Dev.* **14** 1169
[27] Xue D, Balachandran P V, Hogden J, Theiler J, Xue D and Lookman T 2016 *Nat. Commun.* **7** 11241
[28] Noack M M, Yager K G, Fukuto M, Doerk G S, Li R and Sethian J A 2019 *Sci. Rep.* **9** 1
[29] Wigley P B *et al* 2016 *Sci. Rep.* **6** 25890
[30] Nikolaev P, Hooper D, Webber F, Rao R, Decker K, Krein M, Poleski J, Barto R and Maruyama B 2016 *npj Comput. Mater.* **2** 1
[31] Rohr B, Stein H S, Guevarra D, Wang Y, Haber J A, Aykol M, Suram S K and Gregoire J M 2020 *Chem. Sci.* **11** 2696
[32] Häse F, Roch L M and Aspuru-Guzik A 2018 *Chem. Sci.* **9** 7642
[33] Elhara O, Varelas K, Nguyen D, Tusar T, Brockhoff D, Hansen N and Auger A 2019 (arXiv:1903.06396)
[34] Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J and Zaremba W 2016 (arXiv:1606.01540)
[35] Hertel L, Collado J, Sadowski P, Ott J and Baldi P 2020 (arXiv:2005.04048)
[36] Akiba T, Sano S, Yanase T, Ohta T and Koyama M 2019 *Proc. 25th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining* pp 2623–31
[37] Biscani F and Izzo D 2020 *J. Open Source Softw.* **5** 2338
[38] Felton K, Rittig J and Lapkin A 2021 *Chemistry—Methods* **1** 116
[39] LeCun Y, Cortes C and Burges C 2010 *ATT Labs* vol 2 (available at: http://yann.lecun.com/exdb/mnist)
[40] Krizhevsky A 2009 (Toronto) (https://www.cs.toronto.edu/~kriz/cifar.html)
[41] Wu Z, Ramsundar B, Feinberg E N, Gomes J, Geniesse C, Pappu A S, Leswing K and Pande V 2018 *Chem. Sci.* **9** 513
[42] Blum L C and Reymond J-L 2009 *J. Am. Chem. Soc.* **131** 8732
[43] Ruddigkeit L, van Deursen R, Blum L C and Reymond J-L 2012 *J. Chem. Inf. Model.* **52** 2864
[44] Rupp M, Tkatchenko A, Müller K-R and von Lilienfeld O A 2012 *Phys. Rev. Lett.* **108** 058301
[45] Ramakrishnan R, Dral P O, Rupp M and von Lilienfeld O A 2014 *Sci. Data* **1** 140022
[46] Ramakrishnan R, Hartmann M, Tapavicza E and von Lilienfeld O A 2015 *J. Chem. Phys.* **143** 084111
[47] Glavatskikh M, Leguy J, Hunault G, Cauchy T and Mota B D 2019 *J. Cheminformatics* **11** 69
[48] Brown N, Fiscato M, Segler M H and Vaucher A C 2019 *J. Chem. Inf. Model.* **59** 1096
[49] Polykovskiy D *et al* 2018 (arXiv:1811.12823)
[50] The GPyOpt authors 2016 Gpyopt: a Bayesian optimization framework in python (available at: http://github.com/SheffieldML/GPyOpt)
[51] Bergstra J S, Yamins D and Cox D D 2013 *Proc. 30th Int. Conf. on Machine Learning* pp 115–23
[52] Bergstra J S and Bengio Y 2012 *J. Mach. Learn. Res.* **13** 281
[53] Bergstra J S, Bardenet R, Bengio Y and Kégl B 2011 *Advances in Neural Information Processing Systems* pp 2546–54
[54] Häse F, Roch L M, Kreisbeck C and Aspuru-Guzik A 2018 *ACS Cent. Sci.* **4** 1134
[55] Fortin F-A, De Rainville F-M, Gardner M-A, Parizeau M and Gagné C 2012 *J. Mach. Learn. Res.* **13** 2171
[56] Hansen N, Müller S D and Koumoutsakos P 2003 *Evol. Comput.* **11** 1
[57] Hansen N and Ostermeier A 2001 *Evol. Comput.* **9** 159
[58] Eberhart R and Kennedy J 1995 *Proc. 6th Int. Symp. Micro Machine and Human Science (MHS'95)* (IEEE) pp 39–43
[59] Shi Y and Eberhart R 1998 *1998 IEEE Int. Conf. Evolutionary Computation Proc.. IEEE World Congress Computational Intelligence* (IEEE) Cat. No. 98TH8360 pp 69–73
[60] Storn R and Price K 1997 *J. Glob. Optim.* **11** 341
[61] Curry H B 1944 *Q. Appl. Math.* **2** 258
[62] Bouwmeester H, Dougherty A and Knyazev A V 2015 *Proc. Comput. Sci.* **51** 276
[63] Hestenes M R *et al* 1952 *J. Res. Natl Bur. Stand.* **49** 409
[64] Zhu C, Byrd R H, Lu P and Nocedal J 1997 *ACM Trans. Math. Softw.* **23** 550
[65] Byrd R H, Lu P, Nocedal J and Zhu C 1995 *SIAM J. Sci. Comput.* **16** 1190
[66] Nocedal J and Wright S J 2006 *Numerical Optimization* (New York: Springer) pp 529–62
[67] Kraft D 1988 A software package for sequential quadratic programming (Koln, Germany: DLR German Aerospace Center) Tech. Rep. DFVLR-FB 88-28
[68] Anderson M J and Whitcomb P J 2016 *DOE Simplified: Practical Tools for Effective Experimentation* (Boca Raton, FL: CRC Press)
[69] Box G E P, Hunter J S and Hunter W G 2005 *Statistics for Experimenters: Design, Innovation and Discovery* vol 2 (New York: Wiley)
[70] Fisher R A 1937 *The Design of Experiments* (Edinburgh: Oliver and Boyd)
[71] Sobol' I M 1967 *Zh. Vychisl. Mat. Mat. Fiz.* **7** 784
[72] Huyer W and Neumaier A 2008 *ACM Trans. Math. Softw.* **35** 1
[73] Wales D J and Doye J P 1997 *J. Phys. Chem. A* **101** 5111

[74] Nelder J A and Mead R 1965 *Comput. J.* **7** 308
[75] Bubeck S *et al* 2015 *Found. Trends Mach. Learn.* **8** 231
[76] Boyd S, Boyd S P and Vandenberghe L 2004 *Convex Optimization* (Cambridge: Cambridge University Press)
[77] Lucia A and Xu J 1990 *Comput. Chem. Eng.* **14** 119
[78] Baba N 1981 *J. Optim. Theory Appl.* **33** 451
[79] Matyas J 1965 *Autom. Remote Control* **26** 246
[80] Rechenberg I 1978 *Simulationsmethoden in der Medizin und Biologie* (Berlin: Springer) pp 83–114
[81] Schwefel H-P 1977 *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie* (Berlin: Springer) pp 123–76
[82] Zames G, Ajlouni N, Ajlouni N, Ajlouni N, Holland J, Hills W and Goldberg D 1981 *Inf. Technol. J.* **3** 301
[83] Koza J R and Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* vol 1 (Cambridge, MA: MIT Press)
[84] Srinivas M and Patnaik L M 1994 *Computer* **27** 17
[85] Wierstra D, Schaul T, Glasmachers T, Sun Y, Peters J and Schmidhuber J 2014 *J. Mach. Learn. Res.* **15** 949
[86] Mockus J 2012 *Bayesian Approach to Global Optimization: Theory and Applications* vol 37 (Netherlands: Springer Science & Business Media)
[87] Mockus J, Tiesis V and Zilinskas A 1978 *Towards Global Optimization* vol 2 (Amsterdam: Elsevier) p 2
[88] Močkus J 1975 *Optimization Techniques IFIP Technical Conference* (Berlin: Springer) pp 400–4
[89] Roch L M, Häse F, Kreisbeck C, Tamayo-Mendoza T, Yunker L P E, Hein J E and Aspuru-Guzik A 2020 *PLoS One* **15** 1
[90] Walker B, Bannock J, Nightingale A M and deMello J 2017 *React. Chem. Eng.* **2** 785–98
[91] Schweidtmann A, Clayton A, Holmes N, Bradford E, Bourne R and Lapkin A 2018 *Chem. Eng. J.* **352** 277–82
[92] The Olympus authors 2020 Olympus: a benchmarking framework for noisy optimization and experiment planning (available at: https://github.com/aspuru-guzik-group/olympus)