COMMUNICATION

Active learning for autonomous chemical synthesis

Caroline Solon*[a] , Andrew McCluskey [a] and Josh Dunn [a]

a: School of Chemistry, University of Bristol

Abstract here. The abstract should be a single paragraph which summarises the content of the article. It should be no longer than 50 words (approximately 5-6 lines).

1. Introduction

In the world of chemistry, our main problem is that most experiments are very costly in terms of time because the experiments are time-consuming and because most of the information is uninformative, and in terms of materials because of the lack or high cost of the material. For example, in the field of neutron scattering, beams are rare and in high demand, which makes it difficult to carry out such experiments. We could therefore see the emergence of a real need in the automation of chemical reactions.

The topic has been explored in many papers in different areas of chemistry such as materials, organic and analytical chemistry. Most of them used Bayesian optimisation with Gaussian processes and genetic algorithms.

1.2 Bayesian Optimization

1.2.1. Bayesian optimization overview

Bayesian optimization (BO) is a probabilistic model-based method for global optimization of black-box functions that are expensive to evaluate non-convex or lack an analytical expression. Indeed, Bayesian optimization enable us to predict the behaviour of the function based on past evaluations and determines the next best point to sample based on this model.

1.2.2. Utility of Bayesian optimization

It is particularly useful in scenarios where sampling the function to be optimized is costly, time-consuming, or has noisy outcomes. This makes it highly relevant in fields like machine learning hyperparameter tuning, experimental design, and automated system tuning where direct probing of the function can be prohibitive.

1.2.3. Bayesian optimization algorithm

Step 1: Define the Objective Function

The objective function, which we aim to optimize, must be well-defined. In Bayesian optimization, this function does not need to be convex or even continuous.

Step 2: Choose a Surrogate Model

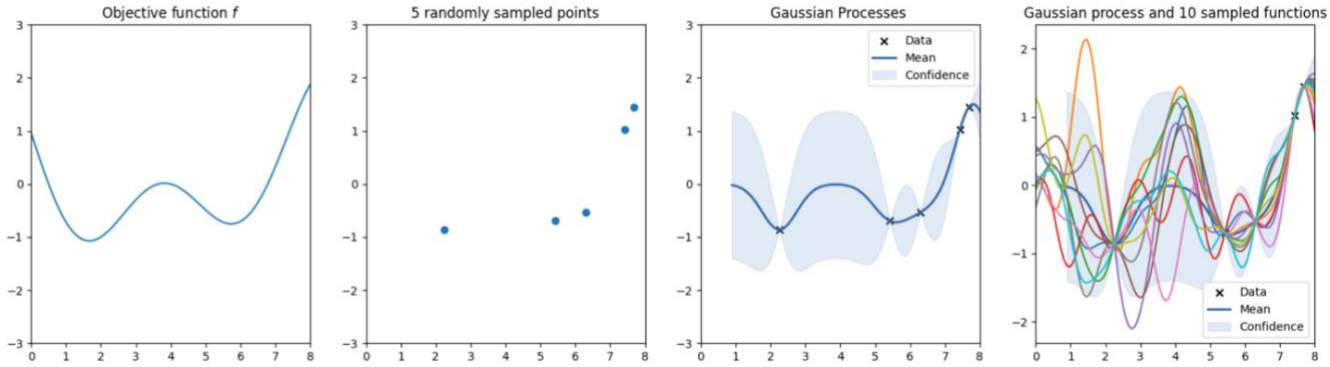
Bayesian optimization typically uses Gaussian Processes (GPs) as the surrogate model to approximate the objective function. GPs are favored because they provide a probabilistic measure of the model's predictions, which is crucial for quantifying the uncertainty in predictions of the objective function's behavior.

Step 3: Select an Acquisition Function

The acquisition function is used to decide where to sample next. It balances exploration (sampling where the model is uncertain) and exploitation (sampling where the model predicts high performance). Common acquisition functions include Expected Improvement, Probability of Improvement, and Upper Confidence Bound.

Step 4: Update the Model

After selecting a point and obtaining a new observation from the objective function, the surrogate model is updated. This update incorporates all the data available, including the newly acquired point, to refine the model's predictions.



Step 5: Repeat

Steps 3 and 4 are repeated until a stopping criterion is met, which could be a maximum number of iterations, a convergence criterion, or when the improvement falls below a threshold.

1.2.4. Parameters of the Bayesian Optimization Algorithm

Three commonly used acquisition functions in this context are Probability of Improvement (PI), Expected Improvement (EI) and Lower Confidence Bound (LCB), each with its own characteristics and advantages.

The *Probability of Improvement (PI*) estimates the probability that a new assessment of the objective function at a given point will be better than the current best assessment. Although this approach can lead to improvements, it tends to favour marginal gains, which can make it less adventurous and potentially less effective for finding global optima.

Expected Improvement (EI), on the other hand, considers both the probability of improvement and the magnitude of that potential improvement. This makes it possible to balance exploration and exploitation more effectively, by favouring points that are not only likely to be better than the current best, but also those that could offer significant improvements.

The *Lower Confidence Bound (LCB)* is an acquisition function that combines the model's average prediction ($\mu(x)$) and its uncertainty ($\sigma(x)$), weighted by a factor α. By adjusting α, we can control the balance between exploring new regions of the parameter space (areas of high uncertainty) and exploiting regions known to offer good performance. A high α favours exploration, while a low or zero α favours exploitation.

$$\boldsymbol{LCB}(\boldsymbol{x}) = \boldsymbol{\mu}(\boldsymbol{x}) - \boldsymbol{\alpha\sigma}(\boldsymbol{x})$$

Exploration and *exploitation* are two key strategies in Bayesian optimisation. Exploitation focuses on using current knowledge to find promising solutions, while exploration actively seeks to reduce model uncertainty by evaluating points in less well-understood regions of the parameter space. A judicious balance between these two approaches is crucial to avoid becoming trapped in local minima and to efficiently identify the global optimum of the objective function.

In summary, the acquisition functions in Bayesian optimisation play a crucial role in determining the points to be evaluated next, by balancing the exploration of the unknown and the exploitation of the knowledge acquired. Choosing the appropriate acquisition function and adjusting its parameters (such as α for LCB) is essential for efficiently navigating the parameter space and finding the optimum of the objective function.

In summary, Bayesian optimization is a robust method for optimizing complex, expensive functions by intelligently choosing the next points to evaluate based on a model updated iteratively with data from previous evaluations. Its use of probabilistic models and acquisition functions to balance exploration and exploitation makes it particularly effective in scenarios where data is costly to acquire.

1.3 Gaussian Processes

1.3.1. Gaussian Processes Overview

Gaussian Processes (GPs) are a fundamental tool in machine learning and statistics for modelling and predicting complex phenomena. They are particularly useful for tasks where one needs to make predictions about the behaviour of a function based on observed data. GPs are defined by a mean function and a covariance function (or kernel), which encapsulates our assumptions about the function's smoothness and how points in the input space are correlated.

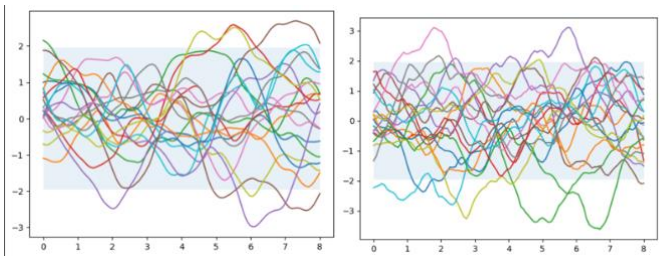
In Gaussian Processes (GPs), the mean function, often represented as m(x), is the expected value of the function at each point x. It is typically assumed to be zero when there is no prior knowledge about the function's behaviour.

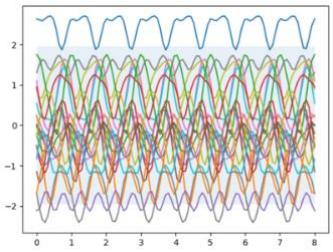
1.3.2. Gaussian Processes parameters

The covariance function, or kernel, denoted as k(x,x), is essential for defining the covariance between function values at any two points x and x' in the input space.

$$\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{e}^{\frac{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}}$$

This function is crucial as it encodes assumptions about the function's properties, such as smoothness and the correlation length scale over which function values are related. Popular choices for the kernel include RBF, Matérn, Periodic, Rational Quadratic kernels, each offering different characteristics of smoothness.





The noise variance, represented as $\boldsymbol{\sigma_n^2}$ accounts for the variance of observational noise, assuming it follows a normal distribution. This parameter is vital for preventing the overfitting of the model to noisy data.

In summary, Gaussian Processes provide a robust framework for modelling and prediction in complex, noisy environments. Their application in autonomous data acquisition systems at synchrotron and neutron facilities showcases their utility in efficiently navigating high-dimensional parameter spaces, crucial for advancing scientific research.

1.4. Genetic algorithm

1.4.1. Genetic algorithm overview

Genetic algorithms (GAs) are a type of evolutionary algorithm used for solving optimization and search problems through techniques inspired by natural evolution, such as mutation, crossover, and selection. These algorithms are particularly useful in areas where finding exact solutions is impractical due to the complexity or size of the problem.

1.4.2. Utility of genetic optimisation

Genetic algorithms are widely used across various fields due to their robustness and ability to find good solutions to complex problems. They are especially valuable in optimization problems where traditional methods fall short. For instance, GAs have been successfully applied in engineering for optimizing design parameters, in economics for modeling supply and demand, and in machine learning for feature selection and hyperparameter tuning. Their adaptability makes them suitable for dynamic environments and problems with multiple local optima, where traditional gradient-based optimization methods might fail.

1.4.3. Genetic algorithm

Step1: Initialization - Begin by generating an initial population of individuals. These individuals are potential solutions to the problem and are usually represented by strings of characters or numbers.

Step2: Evaluation - Assess the fitness of each individual in the population. The fitness function measures how good a solution an individual is with respect to the problem being solved.

Step3: Selection - Select individuals based on their fitness scores to participate in breeding the next generation. Higher fitness scores generally increase an individual's likelihood of being selected.

Step4: Crossover - Combine the genetic information of selected individuals to create new offspring. This is done through methods like one-point crossover or two-point crossover, where segments of parent genomes are mixed.

Step5: Mutation - Introduce random genetic mutations to the offspring. This helps maintain genetic diversity within the population and can prevent the algorithm from getting stuck in local optima.

Step6: Termination - Repeat the process from evaluation to mutation until a stopping condition is met, such as a maximum number of generations or a satisfactory fitness level being achieved.

1.4.4. Key Parameters of Genetic Algorithms

- Population Size: This parameter defines the number of individual solutions in each generation. A larger population can provide a more diverse genetic pool, but it also requires more computational resources to evaluate.

- Crossover Probability: This is the likelihood that two parent solutions will combine to produce offspring. Crossover is a crucial mechanism for combining genetic information from different individuals to explore new parts of the solution space.

- Mutation Probability: Mutation introduces random changes to individual solutions, helping to maintain genetic diversity within the population and preventing the algorithm from becoming too homogeneous and getting stuck in local optima.

- Selection Method: This involves choosing which individuals get to reproduce based on their fitness levels. Common methods include roulette wheel selection, tournament selection, and rank selection. The selection process ensures that better solutions have a higher chance of passing their genes to the next generation.

- Termination Criteria: This can be a set number of generations, a convergence threshold, or a satisfactory fitness level. It determines when the algorithm should stop running.

2. Materials and methods

Finally, the aim is to compare different optimisation algorithms on a simple dataset, then on more and more complex chemical datasets, in order to implement them in an automated platform.

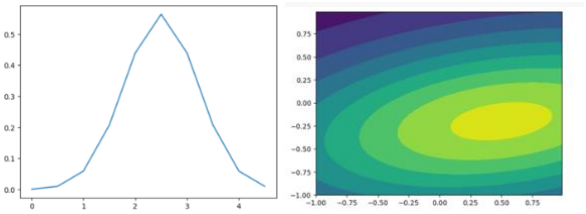
The optimisation of a synthetic process using automation is becoming important in synthetic chemistry, assisted by developments such as the ChemSpeed platform. If the *in-silico* implementation is successful, the project can be expanded to investigate running these algorithms on the Bristol Automated Synthesis Facility.

We have found several packages in Python that are able to do Bayesian optimisations such as BoFire, Dragonfly, with more insight into log Gaussian processes such as Ariane and able to do genetic algorithms such as PySwarm or Differential Evolution.

2.1. Datasets

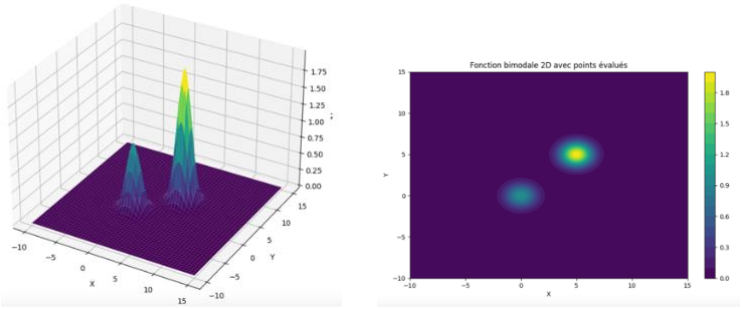
a) Simple function: multivariate distribution

The dataset we will be using is based on the scipy.stats.multivariate_normal function from the SciPy library. This function represents a multivariate normal (Gaussian) distribution, which is a generalization of the one-dimensional normal distribution to higher dimensions. A key characteristic of this distribution is that it is fully defined by two parameters: the mean and the covariance matrix.



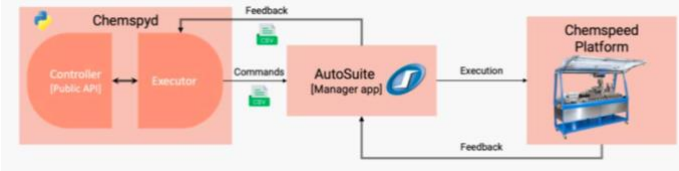
b) Complex function: bimodal function

We will use a bimodal function designed for optimization scenarios. It accepts a two-dimensional vector, computes the values of two Gaussian distributions centered at specified locations with a given standard deviation for each. The function returns a combination of these two Gaussians, adjusted to convert a maximization problem into a minimization problem, making it useful for testing the performance of optimization algorithms against functions with multiple local optima.



2.2 Chemspeed platform

Chemspyd operates by interfacing with Chemspeed's AutoSuite software, employing a dual-class system comprising a Controller and an Executor. The Controller serves as a public API that allows users to craft custom workflows in Python, while the Executor manages direct communication with the robotic platform. This setup not only facilitates the execution of complex experimental sequences but also enables real-time decision-making and adjustments based on ongoing results.



The utility of Chemspyd was demonstrated through several case studies, including the synthesis of silver nanoprisms and the exploration of conditions for Buchwald-Hartwig coupling reactions. These studies highlighted Chemspyd's capability to perform and manage multiple parallel experiments, significantly reducing the time and labor associated with traditional methods. The integration of Chemspyd with large language models to provide a natural language interface further exemplifies its potential to make laboratory automation more accessible and intuitive.

The development of Chemspyd represents a significant step forward in the democratization of laboratory automation technology. By providing an open-source interface, Chemspyd not only enhances the flexibility and adaptability of automated systems but also encourages the scientific community to develop and share new modules and workflows. This could lead to a more collaborative and innovative approach in the automation of scientific experiments.

2.3. Optimisation packages

BoFire

BoFire (Bayesian Optimization Framework Intended for Real Experiments) is a Python package designed to optimize experimental designs using Bayesian optimization techniques. It allows continuous, discrete and categorical inputs to be defined through classes such as ContinuousInput. Users can also define outputs via ContinuousOutput and structure the entire experimental domain using the Domain class, which encompasses inputs, outputs and constraints.

Linear constraints, both equality and inequality, are handled by LinearEqualityConstraint and LinearInequalityConstraint, allowing users to specify precise relationships between variables. The SOBO (Single Objective Bayesian Optimization) optimization strategy, implemented via SoboStrategy, uses acquisition functions such as qNEI (quantitative Expected Improvement) to guide the search to the most promising regions of the design space. This package is particularly useful for experiments where the precision of experimental configurations is crucial and objective function evaluations are costly.

Dragonfly

Dragonfly is a library for Bayesian optimisation and other global optimisation methods. It is particularly well suited to problems where function evaluations are expensive. Dragonfly allows complex optimisation domains to be defined and supports multi-objective and multi-fidelity optimisation. The package uses Gaussian process regression models, as illustrated by the use of ScikitLearnGaussianProcessRegressor in the example, to model the objective function.

Users can configure the search domain with precise bounds for each variable, making it easy to apply to optimisation problems in continuous spaces. The model adapter, ModelAdapter, makes it easy to integrate custom models for evaluation in optimisation, making Dragonfly flexible and powerful for scientific research and engineering applications.

Ariane

Ariane is a tool designed for optimisation and data analysis in the context of neutron and X-ray experiments, using machine learning techniques to improve experimental efficiency. Although the specific example is not directly related to Ariane, it illustrates the use of Gaussian processes for regression, a technique commonly used in modelling and optimisation applications in materials science and physics.

Particle Swarm Optimization (PSO)

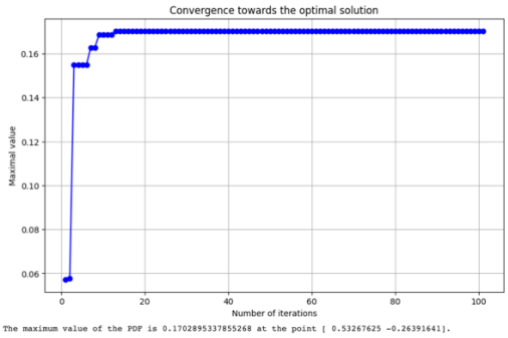
PSO is an optimisation technique that simulates the social behaviour of birds and fish. The pyswarms package makes it easy to implement PSO to optimise non-differentiable or noisy functions. In the example, PSO is used to minimise the objective function, which is common in optimisation problems where the minimum value point is sought in a given search space.

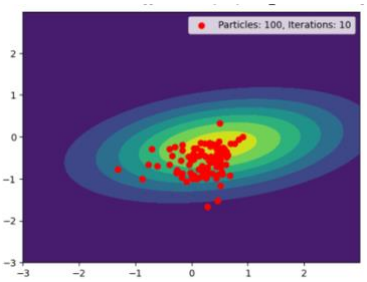
Differential Evolution (DE)

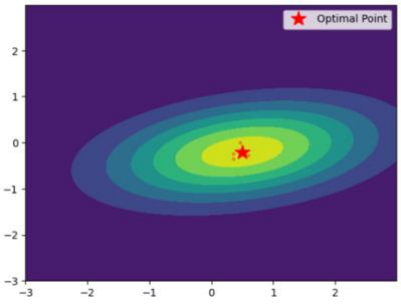
The Differential Evolution algorithm is implemented in SciPy and is used to optimise real multidimensional functions. This algorithm is robust for problems with complex objective functions and can handle large search spaces. The example shows how DE can be used to minimise a function, a common approach in many scientific and technical optimisation problems.

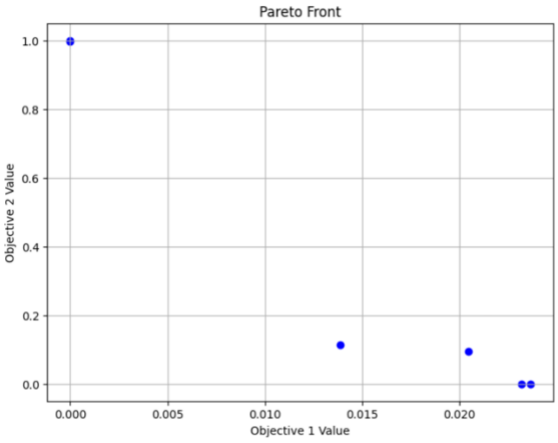
Each of these packages offers unique tools for tackling complex optimisation problems, with applications ranging from experimental design to global optimisation in a variety of scientific and engineering fields.

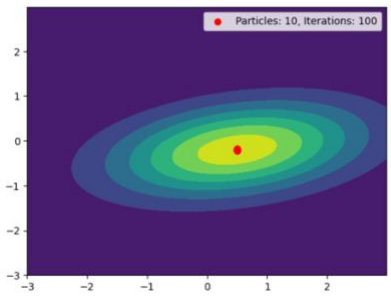
3. Results and discussion

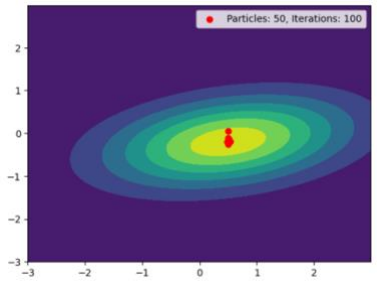


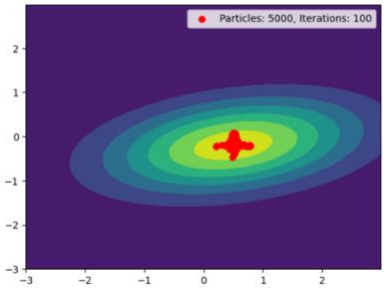


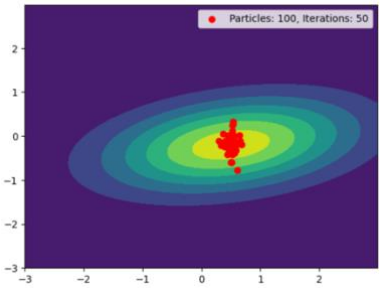


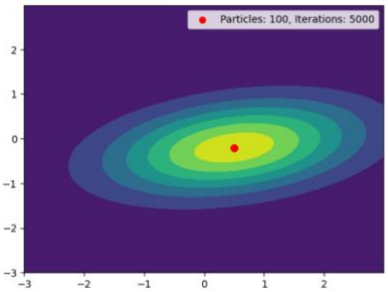












Conclusions

The conclusions section should come in this section at the end of the article. Please remove the "**Conclusions**" heading for articles submitted to *Chemical Communications*.

Author Contributions

Caroline SOLON, author of this communication

Conflicts of interest

There are no conflicts to declare.

Notes and references

‡ Footnotes relating to the main text should appear here. These might include comments relevant to but not central to the matter under discussion, limited experimental and spectral data, and crystallographic data.

§
§§
etc.

1 Citations should appear here in the format A. Name, B. Name and C. Name, *Journal Title*, 2000, **35**, 3523; A. Name, B. Name and C. Name, *Journal Title*, 2000, **35**, 3523.
2 …

We encourage the citation of primary research over review articles, where appropriate, in order to give credit to those who first reported a finding. Find out more about our commitments to the principles of San Francisco Declaration on Research Assessment (DORA).