Enhancing finite-difference-based derivative-free optimization with machine learning

Timothé Taminiau*[1], Geovani Nunes Grapiglia†[1], and Estelle Massart‡[1]

[1]Université Catholique de Louvain, ICTEAM/INMA, Avenue Georges Lemaître 4, B-1348, Louvain-la-Neuve, Belgium

July 3, 2026

Abstract

Derivative-Free Optimization (DFO) involves methods that rely solely on evaluations of the objective function. One of the earliest strategies for designing DFO methods is to adapt first-order methods by replacing gradients with finite-difference approximations. The execution of such methods generates a rich dataset about the objective function, including iterates, function values, approximate gradients, and successful step sizes. In this work, we propose a simple auxiliary procedure to leverage this dataset and enhance the performance of finite-difference-based DFO methods. Specifically, our procedure trains a surrogate model using the available data and applies the gradient method with Armijo line search to the surrogate until it fails to ensure sufficient decrease in the true objective function, in which case we revert to the original algorithm and improve our surrogate based on the new available information. As a proof of concept, we integrate this procedure with the derivative-free method proposed in (Optim. Lett. 18: 195–213, 2024). Numerical results demonstrate significant performance improvements, particularly when the approximate gradients are also used to train the surrogates.

1 Introduction

In this work, we address the problem of minimizing a possibly nonconvex function $f : \mathbb{R}^n \to \mathbb{R}$ that is bounded from below and is assumed to be differentiable with a Lipschitz continuous gradient. We focus on the black-box setting, where $f(\,\cdot\,)$ is accessible only through function evaluations, i.e., no derivatives can be explicitly computed. The minimization of black-box functions arises, for example, in the calibration of mathematical models that require computer simulations for evaluation, where we want to find

*Email: timothe.taminiau@uclouvain.be. Supported by "Fonds spéciaux de Recherche", UCLouvain.

†Email: geovani.grapiglia@uclouvain.be. Partially supported by FRS-FNRS, Belgium (Grant CDR J.0081.23).

‡Email: estelle.massart@uclouvain.be.

the vector x of model parameters that minimizes the misfit $f(x)$ between the model predictions and the available data [5, 16]. Another relevant class of applications is the synthesis of materials or molecules, where x defines the design variables and $f(x)$ measures the discrepancy between the desired and actual material's properties, with the latter accessed through laboratory experiments [20, 27, 10]. Optimization problems of this type require derivative-free optimization (DFO) methods [2, 6, 21], which rely solely on function evaluations.

Local deterministic DFO methods can be categorized into three main classes: direct-search methods [1, 26], model-based trust-region methods [24, 7], and finite-difference-based methods [13, 25]. At the k-th iteration of a direct-search method, the objective function is evaluated at several poll points around the current iterate x_k. If one of these poll points yields a function value sufficiently smaller than $f(x_k)$, it is accepted as the new iterate x_{k+1}. Otherwise, the method sets $x_{k+1} = x_k$ and evaluates a new, closer set of poll points. In contrast, model-based trust-region methods construct an interpolation model of the objective function using function values at points near x_k. This model is minimized within a trust region, typically a ball centered at x_k. If model minimizer provides sufficient decrease in the objective function, it is accepted as x_{k+1}; otherwise, the method sets $x_{k+1} = x_k$, reduces the trust-region radius, and updates the interpolation model accordingly. Finite-difference-based methods, on the other hand, adapt standard first-order optimization techniques (e.g., gradient descent) by replacing the gradient $\nabla f(x_k)$ with a finite-difference approximation $g(x_k)$, combined with careful choices for the differencing parameter. Each iteration of a deterministic DFO method may require $\mathcal{O}(n)$ evaluations of the objective function $f(\,\cdot\,)$, which are used to evaluate poll points, construct interpolation models, or compute finite-difference gradient approximations, depending on the type of method. However, very often in DFO problems, function evaluations are computationally expensive because they involve time-consuming computer simulations or laboratory experiments. Therefore, a key objective in DFO is to design methods that can efficiently approximate solutions with as few function evaluations as possible.

The execution of a DFO method generates an increasingly rich dataset about the objective function, comprising iterates with associated function values, trust region radii or stepsizes from successful iterations, and potentially gradient approximations. In recent years, researchers have investigated the possibility of leveraging this data to improve the performance of DFO methods. The central idea is to use the collected data to train surrogate models and then use these models to identify promising points or search directions, at a very low cost in terms of evaluations of $f(\,\cdot\,)$. For example, [18] proposed the use of neural network surrogates to accelerate the Implicit Filtering DFO method [19]. Specifically, at the k-th iteration of their new method, NNAIF (Neural Network Accelerated Implicit Filtering), the dataset $\mathcal{F} = \{(y_i, f(y_i))\}_{i=1}^{N}$ of all previously computed points with associated function values is used to train a neural network surrogate $m_\theta(\,\cdot\,)$ by approximately solving the problem

$$\min_{\theta} \; L_{\mathcal{F}}(\theta) \equiv \frac{1}{N} \sum_{i=1}^{N} \left(m_\theta(y_i) - f(y_i)\right)^2 . \tag{1}$$

Then, gradient descent is applied to minimize $m_\theta(\cdot)$ with a fixed budget of iterations, returning a point x_k^+. If

$$f(x_k^+) < f(x_k) - \epsilon$$

for a user-defined $\epsilon > 0$, the method sets $x_{k+1} = x_k^+$. Otherwise, x_{k+1} is computed from x_k via a standard iteration of the Implicit Filtering method. The authors established a liminf-type global convergence result for NNAIF and reported numerical results on 31 noisy problems from the CUTEst collection, where NNAIF achieved significant improvements over the standard Implicit Filtering method on large-scale problems. In a different direction, [11] investigated surrogate-based variants of a Full-Low-Evaluation DFO method [4], which originally relies on BFGS search directions computed using finite-difference gradient approximations. Specifically, instead of computing a finite-difference approximation of $\nabla f(x_k)$, they used a dataset of the form $\mathcal{F}$ described above to train a surrogate model $m_\theta(\cdot)$ by solving (1). The search direction was then computed as

$$p_k = -H_k \nabla m_\theta(x_k),$$

where matrix H_k was obtained from H_{k-1}, $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla m_\theta(x_k) - g(x_{k-1})$, via the BFGS update, with $g(x_{k-1})$ being the gradient approximation used in the previous iteration. The authors conducted numerical experiments on a subset of CUTEst problems to test various types of surrogates, including polynomials, radial basis functions, and neural networks with different activation functions. However, none of the surrogate variants was able to outperform the base method that relies on finite-difference gradient approximations.

Inspired by the promising results of [18], in this work we propose a new heuristic to enhance the performance of virtually any finite-difference-based DFO method using surrogate models. Given a base method, our heuristic is invoked at the end of each (outer) iteration. Instead of relying solely on $\mathcal{F}$, it employs the augmented dataset $\mathcal{F} \cup \mathcal{G}$, with $\mathcal{G} = \{(z_j, g(z_j))\}_{j=1}^M$, where $g(z_j) \approx \nabla f(z_j)$, to train a model $m_\theta(\cdot)$ by (approximately) solving the problem

$$\min_\theta \; L_{\mathcal{F},\mathcal{G}}(\theta) \equiv \frac{1}{N}\sum_{i=1}^{N} \left(m_\theta(y_i) - f(y_i)\right)^2 + \frac{1}{M}\sum_{j=1}^{M} \|\nabla m_\theta(z_j) - g(z_j)\|_2^2 + \lambda\|\theta\|_2^2, \quad (2)$$

a technique commonly referred to as Sobolev learning in the Machine Learning community [8, 23]. Once the surrogate is trained, our heuristic applies gradient descent with Armijo line search to $m_\theta(\cdot)$ until it fails to ensure a sufficient decrease of the true objective function, returning then the best point found, which is taken as the next iterate. As a proof of concept, we integrate this procedure with the derivative-free method proposed in [14]. We prove that the enhanced method requires no more than $\mathcal{O}\left(n\epsilon^{-2}\right)$ function evaluations to find an ϵ-approximate stationary point of $f(\cdot)$, where the constant factor in the complexity bound is inversely proportional to the average number of successful surrogate steps per outer iteration. Numerical experiments using radial basis functions and neural network surrogate models demonstrate significant performance improvements compared to the base method.

The paper is organized as follows. In Section 2, we present our surrogate-based heuristic and show how it can be incorporated into a DFO method based on finite differences. We establish a worst-case complexity bound that reflects the performance improvement that successful surrogate steps produce. In Section 3, we describe two classes of models that can be used as surrogates, namely, neural networks (NNs) and radial basis functions (RBFs). We also propose there a new interpretation of Sobolev learning (when combined with finite-difference gradients) as a form of model regularization that penalizes the curvature of the model. Finally, in Section 4, we present numerical results illustrating the performance gains achieved with our proposed heuristic.

2 Surrogate-based heuristic

This section presents our surrogate-based heuristic, detailed in Algorithm 1, which can be seamlessly integrated into any derivative-free optimization algorithm that relies on the (typically expensive) computation of a gradient approximation $g(x) \approx \nabla f(x)$. The core idea of the heuristic is to learn a surrogate model $m_\theta(\cdot)$ of the objective function $f(\cdot)$, as described in Step 1 of Algorithm 1. The surrogate is trained using two datasets:

$$\mathcal{F} = \{(y_i, f(y_i))\}_{i=1}^N, \quad \text{and} \quad \mathcal{G} = \{(z_j, g(z_j))\}_{j=1}^M,$$

which store evaluations of the objective function and its gradient approximation, respectively. Since the surrogate m_θ is an explicit and differentiable model—not a black-box—we can apply a first-order method to it with the goal of indirectly minimizing f. To that end, we perform a descent step on the surrogate using an Armijo-type line search (Step 2), which continues until an iterate fails to yield a sufficient decrease in the true objective f (Step 4). All evaluations of f computed during Step 3 are added to an extended dataset $\mathcal{F}^+$, which is then used to enrich the training set in future calls to Algorithm 1.

Algorithm 1 `Surrogate`$(v,\, f,\, \mathcal{F},\, \mathcal{G},\, \sigma,\, \rho,\, \lambda,\, \gamma,\, \epsilon)$

Inputs: A reference point $v \in \mathbb{R}^n$, a zeroth-order oracle for $f(\,\cdot\,)$, datasets $\mathcal{F} = \{(y_i, f(y_i))\}_{i=1}^N$ and $\mathcal{G} = \{(z_j, g(z_j))\}_{j=1}^M$ with $(v, f(v)) \in \mathcal{F}$, and constants $\sigma, \rho, \lambda, \gamma, \epsilon > 0$.

Step 1: Train a continuously differentiable surrogate model $m_\theta : \mathbb{R}^n \to \mathbb{R}$ by (approximately) solving (2). Set $v_0 = v$, $L_0 = \sigma$, $\mathcal{F}^+ := \emptyset$ and $t := 0$.

Step 2: Find the smallest integer $\ell_t \geq 0$ such that the point

$$\hat{v}_t = v_t - \frac{1}{2^{\ell_t} L_t} \nabla m_\theta(v_t)$$

satisfies the inequality

$$m_\theta(v_t) - m_\theta(\hat{v}_t) \geq \frac{\rho}{2^{\ell_t} L_t} \|\nabla m_\theta(v_t)\|_2^2.$$

Step 3: Compute $f(\hat{v}_t)$ and set $\mathcal{F}^+ := \mathcal{F}^+ \cup \{(\hat{v}_t, f(\hat{v}_t))\}$.

Step 4: If

$$f(v_t) - f(\hat{v}_t) \geq \frac{1}{\gamma\sigma} \epsilon^2, \tag{3}$$

define $v_{t+1} = \hat{v}_t$, $L_{t+1} = 2^{\ell_t - 1} L_t$, set $t := t + 1$ and go back to Step 2. Otherwise, set $t^+ = t$, $v^+ = v_t$ and STOP.

Return $v^+, t^+, \mathcal{F}^+$.

In Step 1 of Algorithm 1, any class of continuously differentiable models can be used, including multivariate polynomials, Hermite interpolation models, RBFs, and NNs.

As condition (3) in Step 4 ensures that

$$f(v) - f(v^+) \geq \frac{t^+}{\gamma\sigma} \epsilon^2,$$

we will refer to t^+ as the number of *successful surrogate steps*. In order to prove finite termination of Algorithm 1, we will consider the following assumption.

Assumption A1 $f(\,\cdot\,)$ is bounded from below by $f_{\text{low}} \in \mathbb{R}$.

Suppose that A1 holds. Then Algorithm 1 has finite termination.

Suppose by contradiction that Algorithm 1 never stops and consider an iteration index t_* such that

$$t_* > \max\left\{1, \gamma\sigma(f(v) - f_{\text{low}})\epsilon^{-2}\right\}.$$

Then, by (3),

$$f(v_t) - f(v_{t+1}) \geq \frac{1}{\gamma\sigma} \epsilon^2 \text{ for } t = 0, 1, \ldots, t_* - 1.$$

Summing up these inequalities, it follows that

$$f(v) - f(v_{t_*}) \geq \frac{t_*}{\gamma\sigma}\epsilon^2 > f(v) - f_{\text{low}}.$$

This implies that $f(v_{t_*}) < f_{\text{low}}$, contradicting A1.

To illustrate the applicability of Algorithm 1, we will integrate it with a simplified version[1] of the DFO method proposed by [14], which will be referred here as the base method. This method is a derivative-free variant of the gradient descent method with Armijo line search where $\nabla f(x_k)$ is approximated using forward finite differences. When f is bounded from below and has Lipschitz continuous gradient, this method requires no more than $\mathcal{O}\left(n\epsilon^{-2}\right)$ function evaluations to find an ϵ-approximate stationary point. The detailed method is presented in Algorithm 2.

Algorithm 2 Base Method (adapted from [14])

Inputs: A starting point $x_0 \in \mathbb{R}^n$, $\sigma_0 \geq \sigma_{\min} > 0$, $\epsilon > 0$, $k := 0$.

Step 1. Set $i := 0$.

Step 1.1. For

$$h_i = \frac{2\epsilon}{5\sqrt{n}(2^i\sigma_k)}$$

compute the gradient approximation $g_{h_i}(x_k)$ defined by

$$[g_{h_i}(x_k)]_j = \frac{f(x_k + h_i e_j) - f(x_k)}{h_i},\ j = 1, \ldots, n.$$

Step 1.2. If

$$\|g_{h_i}(x_k)\|_2 < \frac{4\epsilon}{5},$$

set $i := i + 1$ and go back to Step 1.1.

Step 1.3. If

$$f(x_k) - f\left(x_k - \frac{1}{2^i\sigma_k} g_{h_i}(x_k)\right) \geq \frac{1}{8(2^i\sigma_k)}\|g_{h_i}(x_k)\|_2^2$$

then set $i_k = i$ and

$$x_k^+ = x_k - \frac{1}{2^{i_k}\sigma_k} g_{h_{i_k}}(x_k)$$

Otherwise, set $i := i + 1$ and go back to Step 1.1.

Step 2. Set $x_{k+1} = x_k^+$, $\sigma_{k+1} = \max\{2^{i_k - 1}\sigma_k, \sigma_{\min}\}$, $k := k + 1$ and go back to Step 1.

The surrogate-accelerated variant of this algorithm is presented in Algorithm 3. As it can be seen, incorporating Algorithm 1 in Algorithm 2 requires only minor modifications, which are highlighted using surrounding boxes.

[1] The original version of the method includes a Hessian approximation B_k which can be updated with a BFGS formula based on the finite-difference gradient approximation. Algorithm 2 corresponds to the simple case with $B_k = 0$ for all k.

Algorithm 3 Base Method with Surrogate Heuristic

Inputs: Given $x_0 \in \mathbb{R}^n$, $\sigma_0 \geq \sigma_{\min} > 0$, $\epsilon > 0$, and $\boxed{\rho, \lambda, \gamma > 0}$, set $\boxed{\mathcal{F} = \{(x_0, f(x_0))\},\ \mathcal{G} = \emptyset}$ and $k := 0$.

Step 1. Set $i := 0$.

Step 1.1. For

$$h_i = \frac{2\epsilon}{5\sqrt{n}(2^i \sigma_k)}$$

compute the gradient approximation $g_{h_i}(x_k)$ defined by

$$[g_{h_i}(x_k)]_j = \frac{f(x_k + h_i e_j) - f(x_k)}{h_i},\ j = 1, \ldots, n.$$

Set $\boxed{\mathcal{F} := \mathcal{F} \cup \{(x_k + h_i e_j, f(x_k + h_i e_j))\}_{j=1}^n}$

Step 1.2. If

$$\|g_{h_i}(x_k)\|_2 < \frac{4\epsilon}{5},$$

set $i := i + 1$ and go back to Step 1.1.

Step 1.3. If

$$f(x_k) - f\left(x_k - \frac{1}{2^i \sigma_k} g_{h_i}(x_k)\right) \geq \frac{1}{8(2^i \sigma_k)} \|g_{h_i}(x_k)\|_2^2$$

then set $i_k = i$,

$$x_k^+ = x_k - \frac{1}{2^{i_k} \sigma_k} g_{h_{i_k}}(x_k)$$

and update the datasets:

$$\boxed{\mathcal{F} := \mathcal{F} \cup \{(x_k^+, f(x_k^+))\}, \quad \mathcal{G} := \mathcal{G} \cup \{(x_k, g_{h_{i_k}}(x_k))\}.}$$

Otherwise, set $i = i + 1$ and go back to Step 1.1.

Step 2. Call Algorithm 1 and attempt performing surrogate steps:

$$\boxed{(v_k^+, t_k, \mathcal{F}^+) = \ \texttt{Surrogate}\left(x_k^+, f, \mathcal{F}, \mathcal{G}, 2^{i_k}\sigma_k, \rho, \lambda, \gamma, \epsilon\right)}$$

Step 3. Set $\boxed{x_{k+1} = v_k^+,\ \mathcal{F} := \mathcal{F} \cup \mathcal{F}^+}$,

$$\sigma_{k+1} = \max\{2^{i_k - 1}\sigma_k, \sigma_{\min}\},$$

$k := k + 1$, and go back to Step 1.

Note that calling Algorithm 1 at Step 2 of Algorithm 3 implies the additional cost of training the surrogate models, which can be significant when the surrogate is a neural network. A detailed discussion on the choice of surrogates and the trade-off between oracle efficiency and computational time is presented in Sections 3 and 4. Next, we

analyze the worst-case complexity of Algorithm 3. Our analysis is deliberately general and does not require a particular choice of the surrogate model in Algorithm 1. In fact, we only consider the following additional assumption:

Assumption A2 $\nabla f(\,\cdot\,)$ is L-Lipschitz continuous, that is

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2, \ \forall x, y \in \mathbb{R}^n.$$

Suppose that A2 holds. Then the k-th iteration of Algorithm 3 is well-defined whenever $\|\nabla f(x_k)\|_2 > \epsilon$. In addition, if

$$\|\nabla f(x_k)\|_2 > \epsilon, \text{ for } k = 0, 1, \ldots, T-1,$$

for $T \geq 1$, then

$$\sigma_k \leq 2\max\{\sigma_0, 2L\} = \sigma_{\max}, \text{ for } k = 0, 1, \ldots, T \tag{4}$$

and

$$f(x_k) - f(x_k^+) \geq \frac{1}{2C_f}\|\nabla f(x_k)\|_2^2, \text{ for } k = 0, 1, \ldots, T-1, \tag{5}$$

where

$$C_f = \frac{81}{8}\max\left\{\sigma_{\max}, \frac{L^2}{\sigma_{\min}}\right\}.$$

It follows directly from Lemmas 1-4 in [14].

Let

$$T(\epsilon) = \inf\{k \in \mathbb{N} \mid \|\nabla f(x_k)\|_2 \leq \epsilon\}$$

and suppose that $T(\epsilon) \geq 1$. Given T, with $0 < T \leq T(\epsilon)$, we will denote by $S(T)$ the average number of successful surrogate steps over the first T iterations of Algorithm 3, i.e.,

$$S(T) = \frac{1}{T}\sum_{k=0}^{T-1} t_k,$$

where t_k is an output of our surrogate procedure (Algorithm 1), called at iteration k (see Step 2 in Algorithm 2).

Suppose that A1-A2 hold and let $\{x_k\}_{k=0}^{T(\epsilon)}$ be generated by Algorithm 3. Then

$$T(\epsilon) \leq \frac{2C_{\max}(f(x_0) - f_{\text{low}})}{1 + S(T(\epsilon))}\epsilon^{-2}, \tag{6}$$

where

$$C_{\max} = \max\{C_f, \gamma\sigma_{\max}\}$$

with C_f and $\sigma_{\max}$ defined in Section 2.

Given $k \in \{0, 1, \ldots, T(\epsilon) - 1\}$, it follows from Algorithm 1 and inequality (4) in Section 2 that

$$f(x_k^+) - f(v_k^+) \geq \frac{t_k}{\gamma(2^{i_k}\sigma_k)}\epsilon^2 \geq \frac{t_k}{2\gamma\sigma_{\max}}\epsilon^2. \tag{7}$$

Thus, combining (7) with inequality (5) in Section 2, we obtain

$$\begin{aligned} f(x_k) - f(x_{k+1}) &= f(x_k) - f(v_k^+) \\ &= f(x_k) - f(x_k^+) + f(x_k^+) - f(v_k^+) \\ &\geq \frac{1}{2C_f}\|\nabla f(x_k)\|_2^2 + \frac{t_k}{2\gamma\sigma_{\max}}\epsilon^2 \\ &\geq (1+t_k)\frac{\epsilon^2}{2C_{\max}}. \end{aligned}$$

Summing up these inequalities for $k = 0, 1, \ldots, T(\epsilon) - 1$ and using A1, it follows that

$$\begin{aligned} f(x_0) - f_{\text{low}} &\geq f(x_0) - f(x_{T(\epsilon)}) \\ &= \sum_{k=0}^{T(\epsilon)-1} (f(x_k) - f(x_{k+1})) \\ &\geq \Big(T(\epsilon) + \sum_{k=0}^{T(\epsilon)-1} t_k\Big)\frac{\epsilon^2}{2C_{\max}} \\ &= (1 + S(T(\epsilon)))T(\epsilon)\frac{\epsilon^2}{2C_{\max}}, \end{aligned}$$

which implies that (6) is true. Since the average number of successful surrogate steps $S(T(\epsilon))$ is nonnegative, it follows from Section 2 that Algorithm 3 takes at most $\mathcal{O}\left(\epsilon^{-2}\right)$ outer iterations to find an ϵ-approximate stationary point of f, which is the iteration complexity of the original base method. This lemma also shows that larger values of $S(T(\epsilon))$ lead to a smaller bound for the number of outer iterations to reach ϵ-approximate stationarity, thereby quantifying the impact of successful surrogate steps on the complexity of Algorithm 3. The next result gives us a worst-case complexity bound for the number of function evaluations required by Algorithm 3 to achieve this goal. Moreover, it shows that Algorithm 3 preserves the worst-case complexity bound of $O(n\epsilon^{-2})$ function evaluations of the base method with a refined upper bound.

Suppose that A1-A2 hold. Let $\{x_k\}_{k=0}^{T(\epsilon)}$ be generated by Algorithm 3 and denote by $\text{FE}(\epsilon)$ the corresponding number of evaluations of $f(\,\cdot\,)$ that were computed over the $T(\epsilon)$ first iterations of Algorithm 3. Then

$$\text{FE}(\epsilon) \leq 4\eta(S(T(\epsilon))(n+1)C_{\max}(f(x_0) - f_{\text{low}})\epsilon^{-2} + (n+1)\log_2\left(\frac{\sigma_{\max}}{\sigma_0}\right) + T(\epsilon), \quad (8)$$

where

$$\eta(S(T(\epsilon)) = \frac{1 + \frac{S(T(\epsilon))}{2(n+1)}}{1 + S(T(\epsilon))}$$

and $\sigma_{\max}$, $C_{\max}$ are defined in Section 2 and Section 2.

Notice that

$$\text{FE}(\epsilon) \leq \sum_{k=0}^{T(\epsilon)-1} (i_k + 1)(n+1) + (t_k + 1). \quad (9)$$

By the update rule for σ_k, we have

$$2^{i_k-1}\sigma_k \leq \max\{2^{i_k-1}\sigma_k, \sigma_{\min}\} = \sigma_{k+1}$$

and so

$$i_k + 1 \leq 2 + \log_2(\sigma_{k+1}) - \log_2(\sigma_k). \tag{10}$$

Now combining (9) and (10), and using Section 2 and Section 2, it follows that

$$\begin{aligned}
\mathrm{FE}(\epsilon) &\leq \sum_{k=0}^{T(\epsilon)-1} \left[2 + \log_2\left(\frac{\sigma_{k+1}}{\sigma_k}\right)\right](n+1) + \sum_{k=0}^{T(\epsilon)-1}(t_k+1) \\
&= (n+1)\left[2T(\epsilon) + \log\left(\frac{\sigma_{T(\epsilon)}}{\sigma_0}\right)\right] + T(\epsilon)S(T(\epsilon)) + T(\epsilon) \\
&\leq (n+1)\left[2\Big(1 + \frac{S(T(\epsilon))}{2(n+1)}\Big)T(\epsilon) + \log_2\left(\frac{\sigma_{\max}}{\sigma_0}\right)\right] + T(\epsilon) \\
&\leq 4\eta(S(T(\epsilon)))(n+1)C_{\max}(f(x_0) - f_{\low})\epsilon^{-2} + (n+1)\log_2\left(\frac{\sigma_{\max}}{\sigma_0}\right) + T(\epsilon).
\end{aligned}$$

Note that the function $\eta : \mathbb{R}_+ \to \mathbb{R}$ defined by

$$\eta(S) = \frac{1 + \frac{S}{2(n+1)}}{1+S}$$

is a nonnegative decreasing function with $\eta(0) = 1$. Therefore, even when our surrogate heuristic does not produce any successful step (i.e., $S(T(\epsilon)) = 0$), by Section 2, we have $\mathrm{FE}(\epsilon) = O(n\epsilon^{-2})$, which matches the worst-case complexity bound already established for Algorithm 2. On the other hand, when $S(T(\epsilon)) > 0$, we have $\eta(S(T(\epsilon))) < 1$, which gives an improvement in the oracle complexity bound (8). For this reason, in what follows we will refer to $\eta(S(T(\epsilon)))$ as the *surrogate mismatch*. In particular, if $\boxed{S(T(\epsilon)) \geq n}$, we have

$$\eta(S(T(\epsilon))) \leq \frac{3}{2(n+1)},$$

which, combined with (8), yields

$$\mathrm{FE}(\epsilon) \leq 6C_{\max}(f(x_0) - f_{\low})\epsilon^2 + (n+1)\log_2\left(\frac{\sigma_{\max}}{\sigma_0}\right) + T(\epsilon).$$

Therefore, in this ideal case, the dependence of the complexity bound on the problem dimension would be just due to the term $(n+1)\log_2(\frac{\sigma_{\max}}{\sigma_0})$, which comes from the lack of knowledge of the Lipschitz constant L.

Finally, note that the cost of training the surrogate is not included in our analysis, as we assume it to be negligible relative to the cost of a single function evaluation—an assumption that holds in many applications where evaluating f entails expensive computer simulations or laboratory experiments. When each evaluation takes several minutes or even hours, spending a few seconds to train a surrogate model is generally well justified.

3 Surrogate models of the objective

The surrogate model m_θ used in Algorithm 1 and Algorithm 3 is trained by solving (2); this form of model training, that penalizes errors on the model derivatives, is known in the machine learning community as Sobolev learning [8]. We show next that, under some specific assumptions, Sobolev learning can be interpreted as a regularization on the curvature of the surrogate model.

Note that the approximate gradients $g_{h_1}(z_1), \ldots, g_{h_M}(z_M)$ were obtained in Algorithm 3 using finite differences:

$$[g_h(z)]_l = \frac{f(z + he_l) - f(z)}{h}, \text{ for } l = 1, \ldots, n. \tag{11}$$

This expression requires evaluating f at the $n+1$ points $z_j, z_j + h_j e_1, \ldots, z_j + h_j e_n$, so that

$$\{(z_j, f(z_j)), (z_j + h_j e_1, f(z_j + h_j e_1)), \ldots, (z_j + h_j e_n, f(z_j + h_j e_n))\}_{j=1,\ldots,M} \subset \mathcal{F}.$$

The following proposition shows that, assuming exact interpolation of the model for all these points, Sobolev learning with forward finite-difference gradients induces a regularization on the curvature of the model.

Let $m_\theta : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable, and $z \in \mathbb{R}^n$ be such that

$$m_\theta(z) = f(z) \tag{12}$$

$$m_\theta(z + he_l) = f(z + he_l), \text{ for } l = 1, \ldots, n. \tag{13}$$

Then, there exists $\zeta_1, \ldots, \zeta_n \in [0, 1]$ such that

$$\|g_h(z) - \nabla m_\theta(z)\|_2^2 = \frac{h^2}{4} \sum_{l=1}^{n} [\nabla^2 m_\theta(z + h\zeta_l e_l)]_{ll}^2, \tag{14}$$

where $g_h(z)$ is defined by (11).

By the interpolation conditions (12) and (13), for all $l = 1, \ldots, n$,

$$[g_h(z)]_l = \frac{f(z + he_l) - f(z)}{h} = \frac{m_\theta(z + he_l) - m_\theta(z)}{h}.$$

By the second-order Taylor expansion of $m_\theta(\,\cdot\,)$ with remainder, there exists $\zeta_l \in [0, 1]$ such that

$$m_\theta(z + he_l) = m_\theta(z) + h[\nabla m_\theta(z)]_l + \frac{h^2}{2}[\nabla^2 m_\theta(z + h\zeta_l e_l)]_{ll},$$

so that

$$\frac{m_\theta(z + he_l) - m_\theta(z)}{h} = [\nabla m_\theta(z)]_l + \frac{h}{2}[\nabla^2 m_\theta(z + h\zeta_l e_l)]_{ll}.$$

There follows that

$$[g_h(z) - \nabla m_\theta(z)]_l = \frac{h}{2}[\nabla^2 m_\theta(z + h\zeta_l e_l)]_{ll},$$

which implies that (14) is true.

It follows from Lemma 3 that, when finite difference gradients are used and all points in $\mathcal{F}$ are exactly interpolated, Sobolev learning can be interpreted as a penalty on a term related to the curvature of the model.

In this work, we consider two families of models: NNs and RBFs.

***Shallow NNs*:** The model m_θ is here chosen as a one-hidden-layer neural network:

$$m_\theta(x) = W_2\phi(W_1 x + b_1) + b_2,$$

where $\phi : \mathbb{R}^q \to \mathbb{R}^q$ is an (entrywise) activation function, $W_1 \in \mathbb{R}^{q\times n}, W_2 \in \mathbb{R}^{1\times q}$ are weight matrices and $b_1 \in \mathbb{R}^q, b_2 \in \mathbb{R}$ are biases. While NNs are ubiquitous in machine learning, solving (2) for the parameters $\theta = \{W_1, W_2, b_1, b_2\}$ requires in general iterative optimization algorithms, sometimes incurring substantial additional costs.

***RBFs*:** The model m_θ is chosen as

$$m_\theta(x) = \sum_{i=1}^{N} \alpha_i \psi(\|x - y_i\|_2) + \beta^T x + \delta, \tag{15}$$

where $\psi : \mathbb{R} \to \mathbb{R}$ is a radial basis function, $\alpha \in \mathbb{R}^N$, $\beta \in \mathbb{R}^n$ and $\delta \in \mathbb{R}$ are parameters, and the point y_i for $i = 1, \ldots, N$, are the abscissas of the points in $\mathcal{F}$. Taking $\lambda = 0$, (2) can be efficiently solved for the parameters $\theta = \{\alpha_1, \ldots, \alpha_N, \beta, \delta\}$ by a least-squares solver.

4 Numerical results

In this section, we compare our proposed surrogate-acceleration method (Algorithm 3) to the base method (Algorithm 2) on the benchmark problem set proposed in [15] which is a subset of the well-known CUTEst collection [12].

4.1 Benchmarking procedure

The benchmark test set contains 134 problems whose dimension n ranges from 1 to 110 with an average value around 15. To compare the different methods considered, we rely on data profiles with the convergence test

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_{\text{best}}), \tag{16}$$

where $x_0 \in \mathbb{R}^n$ is the starting point of the method, $\tau > 0$ (here, $\tau = 10^{-4}$) is a tolerance parameter and f_{best} is the best function value found among all the methods within a fixed budget of function evaluations; see [22] for more information. Each curve of a data profile gives thus the proportion of problems for which a method found $x \in \mathbb{R}^n$ satisfying the convergence test (16) depending on the allowed number of function evaluations. In order to account for the difference in dimension among the problems, the number of function evaluations is converted into the number of simplex gradients, where one simplex gradient corresponds to $n + 1$ function evaluations.

4.2 Implementation detail

Algorithm 2 and Algorithm 3 are implemented with the parameters $\sigma_0 = 1$, $\sigma_{\min} = 10^{-2}$, $\rho = 10^{-4}$, $\gamma = \frac{25}{2}$ and $\epsilon = 10^{-5}$. Since training a model based on a large dataset may be computationally expensive or lead to badly-conditioned problems, we limit the number of points in $\mathcal{F}$ and $\mathcal{G}$, ensuring that $N \leq 10(n+1)$ and $M \leq 10$; when one of these thresholds is reached, the oldest points are removed first.

The shallow NN model is made of one fully connected hidden layer with width $q = 5n$. The regularization parameter in the training problem (2) is $\lambda = 10^{-4}$, and (2) is solved with low-memory BFGS, that we stop at iteration K if $\|\nabla L_{\mathcal{F},\mathcal{G}}(\theta_K)\|_2 \leq 10^{-6} \max\{1, \|\nabla L_{\mathcal{F},\mathcal{G}}(\theta_0)\|_2\}$, or when the maximum budget is reached, that is, either more than 1000 iterations, or more than 20 seconds of running time. During the first call to Algorithm 1, the surrogate model is initialized using He initialization [17]. All subsequent calls to Algorithm 1 rely on warm-start initialization, i.e., the new surrogate is initialized based on the last trained surrogate.

For the RBF models, we set $\lambda = 0$ in (2) and solved this problem with a direct least-square solver. If there is more than one optimal solution, the one with the coefficients that have the smallest norm is used.

4.3 Experiments

Our first experiment highlights the benefits of NN-based surrogate acceleration compared to the base method. For the NN surrogate, we rely on the SoftPlus activation:

$$[\phi(x)]_i = \log\left(1 + e^{[x]_i}\right), \text{ for } i = 1, \ldots, q,$$

where q is the hidden layer dimension. The data profile comparing the base method (Algorithm 2) and the NN-accelerated method (Algorithm 3) is given in Figure 1a. We also included in Figure 1a the variant of Algorithm 3 in which Sobolev learning is dismissed, namely, problem (1) is solved instead of (2) in Step 1 of Algorithm 1, using the same solver and stopping criterion as for (2). This figure shows that our NN surrogate model outperforms the base method and that Sobolev learning provides better performance by taking into account previously computed finite-difference gradients.

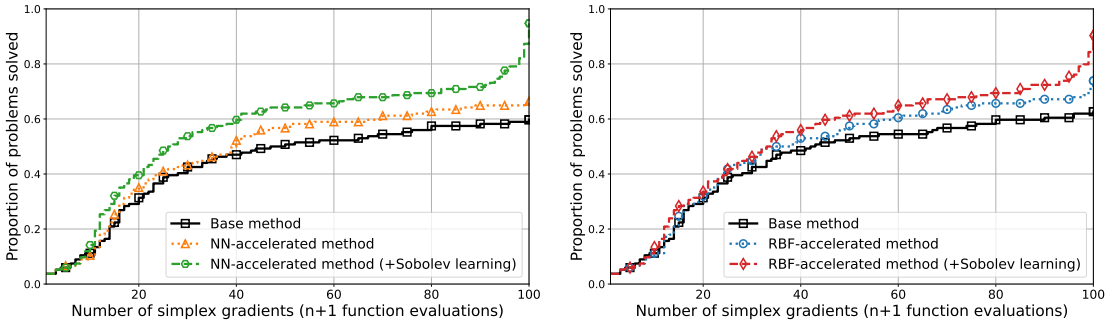
Figure 1b shows the performance of the base method and the surrogate-accelerated methods with the Gaussian RBF approximation model, taking

$$\psi(r) = e^{-r^2},$$

using either standard learning or Sobolev learning with finite-difference gradients. Similarly to the NN-accelerated method, using a RBF surrogate outperforms the base method. Here again, Sobolev learning allows to improve further the performance of the RBF-accelerated method.

To validate the worst-case complexity bounds derived in Section 2, we compute for each test problem the surrogate mismatch defined as

$$\eta(S(T)) = \frac{1 + \frac{S(T)}{2(n+1)}}{1 + S(T)},$$



(a) SoftPlus shallow neural network surrogate

(b) Gaussian RBF surrogate

Figure 1: Data profiles comparing the base method to the surrogate-accelerated methods with standard learning and Sobolev learning, for a budget of 100 simplex gradients.

where $S(T)$ is the average number of surrogate steps over T iterations. The surrogate mismatch $\eta(S(T)) \in [0, 1]$ measures the improvement over the base method that is captured by the worst-case complexity bound. Poor surrogate models will almost never be used and therefore give a surrogate mismatch $\eta(S(T)) \approx 1$, hence only a small improvement in the theoretical performance of the method; in contrast, efficient surrogates will lead to $\eta(S(T)) \ll 1$ and to a significant reduction in the required number of function evaluations to reach a given accuracy. In Figure 2, we show the distribution of $\eta(S(T_{\max}))$ on our benchmark problems set, where $T_{\max}$ is the maximum number of iterations performed by the surrogate-accelerated method on a given problem with a budget of 100 simplex gradients.

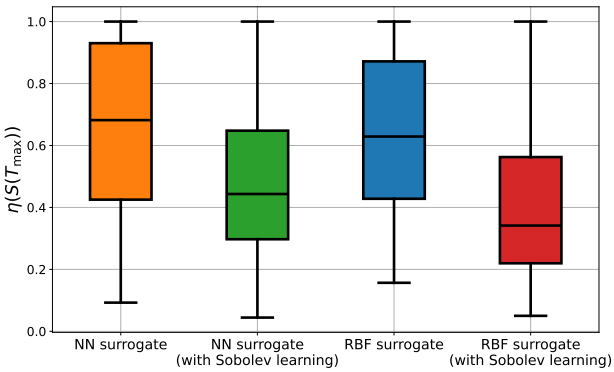


Figure 2: Box plots for the distribution of the surrogate mismatch over the full test set with a budget of 100 simplex gradients.

Figure 2 indicates that $\eta(S(T_{\max}))$ is usually smaller when we incorporate the knowledge of finite-difference gradient approximations, which agrees with the results given in Figure 1a and Figure 1b: the estimated surrogate mismatch $\eta(S(T_{\max}))$ has a median

close to 0.4 and 0.3 respectively for the NN and RBF surrogates with Sobolev learning. Without Sobolev learning, the median surrogate mismatchs are closer to one, with approximate values of 0.7 and 0.6 for NN and RBF surrogates, respectively.

To further compare the two types of surrogates, Figure 3 shows that the NN-accelerated method outperforms the RBF-accelerated method on this test set. In both cases, the gap with respect to the base method increases overall with the maximum budget of function evaluations. Note that this contrasts with the study of the surrogate mismatch $\eta(S(T_{\max}))$ in Figure 2, which predicts a slightly better complexity bound for the RBF, that is not verified numerically. This discrepancy could be explained by the fact that the RBF surrogate steps, even if accepted more often, lead to smaller reductions in the objective than the NN surrogate steps.

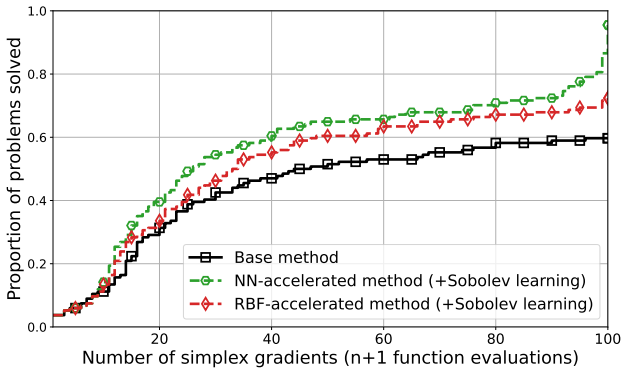


Figure 3: Data profile comparing the base method, the surrogate-accelerated methods based on SoftPlus shallow NN and Gaussian RBF with Sobolev learning, for a budget of 100 simplex gradients.

Figure 4 presents a performance profile [9] with respect to CPU time for solving the test problems according to the criterion in (16), using a maximum budget of 100 simplex gradients and a tolerance of $\tau = 10^{-4}$. As expected, methods based on RBF surrogates, which are easier to train, are generally faster than their neural network (NN) counterparts. Nevertheless, the NN-based method trained using the Sobolev approach, although slower, solves a larger number of problems than the variants based on RBF surrogates.

To conclude, Figure 5 compares the performance of our surrogate-accelerated method with two other derivative-free algorithms, NNAIF and NOMAD [3]. For a fair comparison with NNAIF, which also employs surrogates, we used the same class of surrogate models for both methods, namely a shallow neural network with a softplus activation function. The results show that our NN-accelerated method significantly outperforms NNAIF while remaining competitive with NOMAD.

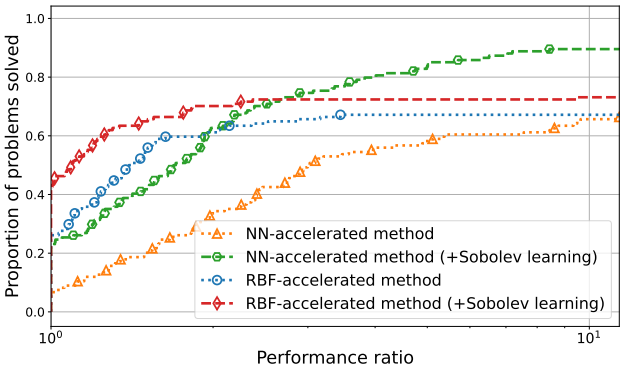


Figure 4: Performance profile with respect to CPU time comparing the different surrogate-accelerated methods.

5 Concluding remarks

In this paper, we proposed a surrogate-based heuristic (Algorithm 1) to enhance the performance of DFO methods that rely on finite-difference gradient approximations. We incorporated the proposed heuristic into an existing DFO method (Algorithm 2) and showed that the resulting scheme (Algorithm 3) requires no more than $\mathcal{O}(n\epsilon^{-2})$ function evaluations to find an ϵ-approximate stationary point of the objective function, where the constant factor in the complexity bound is inversely proportional to the average number of surrogate steps. While each outer iteration of the method costs $\mathcal{O}(n)$ function evaluations, each surrogate step costs only a single function evaluation. Thus, our theoretical results support the intuition that improved surrogates lead to greater efficiency in the method, particularly in terms of reducing calls to the zeroth-order oracle. We also tested Algorithm 3 numerically with different types of models and different learning strategies. The results demonstrate that our surrogate-based heuristic can lead to a significant performance improvement over the base method, particularly when data about the approximate gradients is leveraged through Sobolev learning.

Acknowledgements

The authors are very grateful to the three anonymous referees whose insightful comments helped improve the paper.

References

[1] Charles Audet and J. E. Dennis. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006.

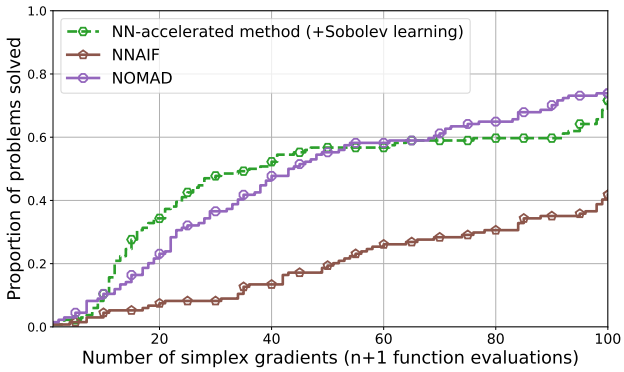


Figure 5: Data profile comparing our surrogate-accelerated method to NNAIF [18], with the same surrogate type and NOMAD [3], for the tolerance $\tau = 10^{-4}$ and budget of 100 simplex gradients.

[2] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, 2017.

[3] Charles Audet, Sébastien Le Digabel, Viviane Rochon Montplaisir, and Christophe Tribes. Algorithm 1027: NOMAD Version 4: Nonlinear Optimization with the MADS Algorithm. *ACM Trans. Math. Softw.*, 48(3):35:1–35:22, 2022.

[4] A. S. Berahas, O. Sohab, and L. N. Vicente. Full-low evaluation methods for derivative-free optimization. *Optimization Methods and Software*, 38(2):386–411, 2023.

[5] Mingjie Chen, Osman A. Abdalla, Azizallah Izady, Mohammad Reza Nikoo, and Ali Al-Maktoumi. Development and surrogate-based calibration of a CO2 reservoir model. *Journal of Hydrology*, 586, 2020.

[6] Andrew R. Conn, Katya Scheinberg, and Luis N. Vicente. *Introduction to Derivative-Free Optimization*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2009.

[7] Andrew R. Conn, Katya Scheinberg, and Luís N. Vicente. Global Convergence of General Derivative-Free Trust-Region Algorithms to First- and Second-Order Critical Points. *SIAM Journal on Optimization*, 20(1):387–415, 2009.

[8] Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. Sobolev Training for Neural Networks. page arXiv:1706.04859, 2017.

[9] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[10] Peter I. Frazier and Jialei Wang. Bayesian optimization for materials design. *arXiv e-prints*, 225:arXiv:1506.01349, 2016.

[11] Tommaso Giovannelli, Oumaima Sohab, and Luis Nunes Vicente. The limitation of neural nets for approximation and optimization. *Journal of Global Optimization*, 2024.

[12] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

[13] Geovani N. Grapiglia. Quadratic regularization methods with finite-difference gradient approximations. *Computational Optimization and Applications*, 85:683–703, 2023.

[14] Geovani N. Grapiglia. Worst-case evaluation complexity of a derivative-free quadratic regularization method. *Optimization Letters*, 18:1–19, 2024.

[15] Serge Gratton and Philippe L. Toint. OPM, a collection of Optimization Problems in Matlab. *arXiv e-prints*, page arXiv:2112.05636, 2021.

[16] Adina Grimmert, Florian Pachnek, and Petra Wiederkehr. Temperature modeling of creep-feed grinding processes for nickel-based superalloys with variable heat flux distribution. *CIRP Journal of Manufacturing Science and Technology*, 41:477–489, 2023.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[18] Brian Irwin, Eldad Haber, Raviv Gal, and Avi Ziv. Neural Network Accelerated Implicit Filtering: Integrating Neural Network Surrogates With Provably Convergent Derivative Free Optimization Methods. In *Proceedings of the 40th International Conference on Machine Learning*, pages 14376–14389. PMLR, 2023.

[19] C. T. Kelley. *Implicit Filtering*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics, 2011.

[20] Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. ChemBO: Bayesian Optimization of Small Organic Molecules with Synthesizable Recommendations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR, 2020.

[21] Jeffrey Larson, Matt Menickelly, and Stefan M. Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019.

[22] Jorge Moré and Stefan Wild. Benchmarking Derivative-Free Optimization Algorithms. *SIAM Journal on Optimization*, 20:172–191, 2009.

[23] Thomas O'Leary-Rosenberry, Chen Peng, Villa Humberto, and Omar Ghattas. Derivative-informed neural operator: An efficient framework for high-dimensional parametric derivative learning. *Journal of Computational Physics*, 496, 2024.

[24] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. In G. Di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 255–297. Springer US, Boston, MA, 2006.

[25] Hao-Jun Michael Shi, Melody Qiming Xuan, Figen Oztoprak, and Jorge Nocedal. On the numerical performance of finite-difference-based methods for derivative-free optimization. *Optimization Methods and Software*, 38(2):289–311, 2023.

[26] L. N. Vicente. Worst case complexity of direct search. *EURO Journal on Computational Optimization*, 1(1):143–153, 2013.

[27] Mao Wang and Jianwen Jiang. Accelerating Discovery of Polyimides with Intrinsic Microporosity for Membrane-Based Gas Separation: Synergizing Physics-Informed Performance Metrics and Active Learning. *Advanced Functional Materials*, 34(23):2314683, 2024.

Declarations

- Conflict of interest: The authors have no conflicts of interest to declare that are relevant to the content of this article.
- Data availability: The OPM test set from [15] is available at `https://github.com/gratton7/OPM` and the code for the surrogate-accelerated method is available at `https://github.com/taminiaut/Surrogate-Accelerated-DFO-Method`