Optimization and Engineering (2023) 24:2827–2853
https://doi.org/10.1007/s11081-023-09795-y

RESEARCH ARTICLE

Hermite least squares optimization: a modification of BOBYQA for optimization with limited derivative information

Mona Fuhrländer[1] · **Sebastian Schöps**[1]

Received: 23 February 2023 / Revised: 15 March 2023 / Accepted: 15 March 2023 /
Published online: 13 April 2023

Abstract
Derivative-free optimization tackles problems, where the derivatives of the objective function are unknown. However, in practical optimization problems, the derivatives of the objective function are often not available with respect to all optimization variables, but for some. In this work we propose the Hermite least squares optimization method: an optimization method, specialized for the case that some partial derivatives of the objective function are available and others are not. The main goal is to reduce the number of objective function calls compared to state of the art derivative-free solvers, while the convergence properties are maintained. The Hermite least squares method is a modification of Powell's derivative-free BOBYQA algorithm. But instead of (under-determined) interpolation for building the quadratic subproblem in each iteration, the training data is enriched with first and—if possible—second order derivatives and then least squares regression is used. Proofs for global convergence are discussed and numerical results are presented. Further, the applicability is verified for a realistic test case in the context of yield optimization. Numerical tests show that the Hermite least squares approach outperforms classic BOBYQA if half or more partial derivatives are available. In addition, it achieves more robustness and thus better performance in case of noisy objective functions.

Keywords Optimization · BOBYQA · Hermite interpolation · Least squares · Noise · Derivative-free

✉ Mona Fuhrländer
mona.fuhrlaender@tu-darmstadt.de

Sebastian Schöps
sebastian.schoeps@tu-darmstadt.de

[1] Computational Electromagnetics Group (CEM) and Centre for Computational Engineering (CCE), Technische Universität Darmstadt, Schlossgartenstr. 8, 64289 Darmstadt, Germany

1 Introduction

In optimization we typically distinguish between gradient based and derivativefree optimization (DFO) approaches. If gradients are available, they provide helpful information about the descent in specific points and thus, can improve the performance of the algorithm. With performance of the optimization algorithm we refer to the ability of reaching the optimal solution and the computational effort in order to reach this solution. However, in practice (e.g. in engineering problem settings using blackbox codes) gradients are often not available. In this research we are interested in optimization problems where the objective function is expensive to evaluate, e.g., involving finite element simulations or Monte Carlo analysis. Compared to evaluating such an objective function, the algebraic cost of the optimization solver itself is negligible. Hence, the computing effort of the methods considered in this work is measured by the number of objective function calls during the optimization. Thus, we focus on the possibility of reducing this number by using all information we have.

DFO methods are often divided into direct and model-based search algorithms, and into stochastic and deterministic algorithms, cf. Conn et al. (2009); Rios and Sahinidis (2013). In deterministic model-based algorithms, the objective function is approximated by a surrogate model and evaluations of this surrogate model are considered to determine the search direction. One example are derivative-free trust region methods, to which belong BOBYQA (using interpolation as surrogate model) and the proposed Hermite least squares method (using least squares regression as surrogate model). Alternatively or additionally, approximations of the derivatives can be developed and used—at the cost of additional computing effort. A global deterministic model-based method is Kriging or Bayesian optimization (Currin et al. 1988), which employs stochastic processes as interpolation model. On the other hand, direct search algorithms evaluate the original objective function. Deterministic examples are the Nelder-Mead simplex algorithm (Nelder and Mead 1965) and the DIRECT algorithm (DIvide a hyper-RECTangle) Jones et al. (1993). Stochastic methods run random search steps, see e.g. simulated annealing (Kirkpatrick et al. 1983), particle swarm (Kennedy and Eberhart 1948) and genetic algorithms (Audet and Hare 2017). For more details, further references and numerical comparisons of DFO methods, we refer to Conn et al. (2009) and Rios and Sahinidis (2013).

In case of multidimensional optimization it often occurs that for some directions the partial derivatives are available and for others not. In Sect. 6 we will provide a practical example in the context of yield optimization. In this work we provide an optimization strategy well suited to benefit from the known derivatives, without requiring derivatives (or approximations of derivatives) for each direction.

Trust region methods are commonly used for solving gradient based and derivative-free nonlinear optimization problems. These methods are based on approximating the objective function quadratically in each iteration and solving this subproblem in a trust region in order to obtain the new iterate solution for the original problem. In sequential quadratic programming (SQP) gradients are used to build the quadratic approximation based on second order Taylor expansions (Ulbrich and Ulbrich 2012, Chap. 19). For DFO Powell proposed the BOBYQA (Bound constrained Optimization BY Quadratic Approximation) method using polynomial interpolation for building the quadratic approximation (Powell 2009). For both, SQP and BOBYQA, there are several modifications and various popular implementations, see e.g. (Cartis et al. 2019, 2021; Powell 1994, 2015; Kraft et al. 1988). Other DFO methods based on quadratic approximations and additionally considering noisy objective data are proposed for example in Cartis et al. (2019) using sample averaging and restarts, in Billups et al. (2013) using weighted regression, and in Menhorn et al. (2022), using Gaussian process regression.

However, to our best knowledge, all these methods have in common that they use derivatives for *all* directions or for *none*. If some derivatives are available, SQP would approximate the missing ones using for example finite differences, classic DFO methods would ignore all derivatives. Finite differences require at least one additional function evaluation per direction each time the gradient has to be calculated. Especially for higher dimensional problems, this leads to an enormous increase of the computational cost. Further, finite differences approximations are sensitive to noisy data. On the other hand, we assume that BOBYQA could perform better, i.e., would need less iterations and thus function evaluations, if we would provide all information we have. For that reason we propose to modify BOBYQA to enable the usage of *some* derivative information. More precisely, we extend the Python implementation PyBOBYQA by Cartis et al. (2019), such that available derivative information is exploited and the (underdetermined) interpolation is replaced by least squares regression. We propose this new variant and, in accordance with the term *Hermite interpolation*, cf. (Hermann 2011, Chap. 6.6) or Sauer and Xu (1995), we call it *Hermite least squares*. Further we investigate the impact of noisy objective functions and observe higher robustness compared to the original BOBYQA and SQP.

This work is structured as follows. We start with the formulation of the problem setting in Sect. 2 and an introduction into DFO and BOBYQA in Sect. 3. In Sect. 4 we propose the Hermite least squares approach and in Sect. 5 we provide numerical results. We conclude the paper with a practical example from the field of electrical engineering in Sect. 6 and some final remarks.

2 Problem setting

Even though SQP is able to handle general nonlinear constraints, BOBYQA, as indicated by its name, only accepts bound constraints. Powell also proposed two more methods, LINCOA (LINearly Constrained Optimization Algorithm), which allows linear constraints, and COBYLA (Constrained Optimization BY Linear Approximations), which allows general constraints but uses only linear approximations (Powell 2015, 1994). However, we focus on BOBYQA and thus, we consider a bound constrained optimization problem with multiple optimization variables, i.e., for a function $f : \mathbb{R}^n \to \mathbb{R}$, an optimization variable $\mathbf{x} \in \mathbb{R}^n$ and lower and upper bounds $\mathbf{x}_{\mathrm{lb}}, \mathbf{x}_{\mathrm{ub}} \in \mathbb{R}^n$ the optimization problem reads

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \\ &\text{s.t. } \mathbf{x}_{\mathbf{lb}} \leq \mathbf{x} \leq \mathbf{x}_{\mathbf{ub}}. \end{aligned} \tag{1}$$

We assume that the derivatives with respect to some directions x_i, $i \in \mathcal{I} = \{1, \dots, n\}$ are known, others are not. We denote the index set of known first order derivative directions by $\mathcal{I}_\mathrm{d} \subseteq \mathcal{I}$, such that the set of available first order derivatives is defined by

$$\mathcal{D} := \left\{ \frac{\partial}{\partial x_i} f \right\}_{i \in \mathcal{I}_\mathrm{d}}. \tag{2}$$

In order to define the set of known second order derivatives, we introduce the tuple set $\mathcal{I}_{2\mathrm{d}} \subseteq \mathcal{I} \times \mathcal{I}$. Then, the set of available second order derivatives is given by

$$\mathcal{D}_2 := \left\{ \frac{\partial^2}{\partial x_i \partial x_j} f \right\}_{(i,j) \in \mathcal{I}_{2\mathrm{d}}}. \tag{3}$$

For the sake of simplicity we will focus on the practically relevant case of first order derivatives. However, the proposed method can be straightforwardly adjusted for using the second order derivatives, cf. Sect. 5.3 and Appendix A. Since we build a quadratic approximation, higher order derivatives are not of concern. In the remainder of this paper, for better readability and without limitation of generality we assume that the x_i are ordered such that we can define

$$\mathcal{I}_\mathrm{d} = \{1, \dots, n_\mathrm{kd}\}, \; n_\mathrm{kd} \leq n \tag{4}$$

as the index set of directions for which we consider the first partial derivative to be known.

3 Model-based search derivative-free optimization

Powell's BOBYQA algorithm is a widely used algorithm in the field of DFO (Powell 2009). The original implementation is in Fortran. Cartis et al. published a Python implementation called PyBOBYQA (Cartis et al. 2019, 2021). It contains some simplifications and several modifications (e.g. for noisy data and global optimization), but Powell's main ideas remain unchanged. In this work, on the programming side, we use PyBOBYQA as a basis and add some new features to it. While the original BOBYQA method (and also PyBOBYQA) are efficient in practice, it cannot be proven that they converge globally, i.e., that they converge from an arbitrary starting point to a stationary point (Conn et al. 2009, Chap. 10.3). Conn et al. reformulated the BOBYQA method in (Conn et al. 2009, Chap. 11.3), maintaining the main concept, but enabling a proof of convergence—on the cost of practical efficiency and of bound constraints. For the theoretical considerations in this work, we take Conn's reformulation as a basis. Before we come to our modifications for mixed gradient information, we recall the basics of DFO methods and BOBYQA.

3.1 Notation

Let $m(\mathbf{x})$ be a polynomial of degree d with $\mathbf{x} \in \mathbb{R}^n$ and let $\Phi = \{\phi_0(\mathbf{x}), \ldots, \phi_q(\mathbf{x})\}$ be a basis in $\mathcal{P}_n^d$ and $q_1 = q + 1$. Further, we define the vector of basis polynomials evaluated in $\mathbf{x}$ by $\mathbf{\Phi}(\mathbf{x}) := (\phi_0(\mathbf{x}), \ldots, \phi_q(\mathbf{x}))^\top$. The training data set is denoted by

$$\mathcal{T} = \{(\mathbf{y}^0, f(\mathbf{y}^0)), \ldots, (\mathbf{y}^p, f(\mathbf{y}^p))\} \tag{5}$$

and $p_1 = p + 1$. The original BOBYQA method is based on interpolation, however, we formulate the problem more generally as interpolation or least squares regression problem. The system matrix and the right hand side of the interpolation or regression problem are then given by

$$\mathbf{M} \equiv \mathbf{M}(\Phi, \mathcal{T}) \text{ with the entries } M_{i,j} = \phi_{j-1}(\mathbf{y}^{i-1}) \tag{6}$$

and

$$\mathbf{b} \equiv \mathbf{b}(\mathcal{T}) \text{ with the entries } b_i = f(\mathbf{y}^{i-1}) \tag{7}$$

If $p_1 = q_1$ the system matrix $\mathbf{M}$ is quadratic and $\mathbf{v} \in \mathbb{R}^{q_1 = p_1}$ solves the interpolation problem

$$\mathbf{M}\mathbf{v} = \mathbf{b}. \tag{8}$$

If $p_1 > q_1$ the system matrix $\mathbf{M}$ is in $\mathbb{R}^{p_1 \times q_1}$ and $\mathbf{v} \in \mathbb{R}^{q_1}$, this leads to an overdetermined interpolation problem and can be solved with least squares regression

$$\mathbf{M}\mathbf{v} \overset{\text{l.s}}{=} \mathbf{b} \quad \Leftrightarrow \quad \min_{\mathbf{v} \in \mathbb{R}^{q_1}} ||\mathbf{M}\mathbf{v} - \mathbf{b}||^2 \quad \Leftrightarrow \quad \mathbf{M}^\top \mathbf{M}\mathbf{v} = \mathbf{M}^\top \mathbf{b}. \tag{9}$$

If the matrix $\mathbf{M}$ in (8) is non-singular, the linear system (8) has a unique solution. Then, following (Conn et al. 2009, Chap. 3), the corresponding training data set is said to be *poised*. Analogously, if the matrix $\mathbf{M}$ in (9) has full column rank, the linear system (9) has a unique solution. And, following (Conn et al. 2009, Chap. 4), the corresponding training data set is said to be *poised for polynomial least-squares regression*. When talking about training data sets in the following, we always assume them to be poised, if not specifically noted otherwise.

Although many of the results hold for any choice of a basis, in the following we use the monomial basis of degree $d = 2$. Thus, if not mentioned differently, for the remainder of this paper, Φ is defined by the $(n + 1)(n + 2)/2$-dimensional basis

$$\Phi = \left\{1, x_1, \ldots, x_n, \frac{1}{2}x_1^2, x_1x_2, x_1x_3, \ldots, x_{n-1}x_n, \frac{1}{2}x_n^2\right\}. \tag{10}$$

3.2 Λ-poisedness

Many DFO algorithms are model based. Thus, in order achieve well behavior of the optimization strategy or to even guarantee global convergence, we have to ensure that the model is *good enough*. In gradient based methods, typically some Taylor expansion error bounds are considered. In DFO methods, which are based on interpolation or regression, the quality of the model depends on the quality of the training data set. This leads us to the introduction of the term Λ-*poisedness*. We recall the definitions of Λ-poisedness from (Conn et al. 2009, Def. 3.6, 4.7, 5.3).

Definition 1 (Λ-poisedness in the interpolation sense) Given a poised interpolation problem as defined in Sect. 3.1. Let $\mathcal{B} \subset \mathbb{R}^n$ and $\Lambda > 0$. Then the training data set $\mathcal{T}$ is Λ-poised in $\mathcal{B}$ (in the interpolation sense) if and only if

$$\forall \mathbf{x} \in \mathcal{B}\, \exists \mathbf{l}(\mathbf{x}) \in \mathbb{R}^{p_1} \text{ s.t. } \sum_{i=0}^{p} l^i(\mathbf{x}) \boldsymbol{\Phi}(\mathbf{y}^i) = \boldsymbol{\Phi}(\mathbf{x}) \quad \text{with} \quad ||\mathbf{l}(\mathbf{x})||_\infty \leq \Lambda. \tag{11}$$

Remark 1 Note that in case of using the monomial basis, the equality in Def. 1 can be rewritten as $\mathbf{M}^\top \mathbf{l}(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x})$ and the $l^i(\mathbf{x})$, $i = 0, \dots, p$ are uniquely defined by the Lagrange polynomials and can be obtained by solving

$$\mathbf{M}\boldsymbol{\lambda}^i = \mathbf{e}^{i+1}, \tag{12}$$

where $\mathbf{e}^i \in \mathbb{R}^{p_1}$ denotes the i-th unit vector and the elements of $\boldsymbol{\lambda}_i$ are the coefficients of the polynomial l^i, which will be evaluated at $\mathbf{x}$.

Definition 2 (Λ-poisedness in the regression sense) Given a poised regression problem as defined in Sect. 3.1. Let $\mathcal{B} \subset \mathbb{R}^n$ and $\Lambda > 0$. Then the training data set $\mathcal{T}$ is Λ-poised in $\mathcal{B}$ (in the regression sense) if and only if

$$\forall \mathbf{x} \in \mathcal{B}\, \exists \mathbf{l}(\mathbf{x}) \in \mathbb{R}^{p_1} \text{ s.t. } \sum_{i=0}^{p} l^i(\mathbf{x}) \boldsymbol{\Phi}(\mathbf{y}^i) = \boldsymbol{\Phi}(\mathbf{x}) \quad \text{with} \quad ||\mathbf{l}(\mathbf{x})||_\infty \leq \Lambda. \tag{13}$$

Remark 2 Note that the $l^i(\mathbf{x})$, $i = 0, \dots, p$ are not uniquely defined since the system in (13) is underdetermined. However, the minimum norm solution corresponds to the Lagrange polynomials (in the regression sense), cf. (Conn et al. 2009, Def. 4.4). Analogously to remark 1 they can be computed by solving

$$\mathbf{M}\boldsymbol{\lambda}^i \overset{\text{l.s.}}{=} \mathbf{e}^{i+1} \tag{14}$$

and using the entries of $\boldsymbol{\lambda}^i$ as coefficients for the polynomial l^i.

It can be shown, that the poisedness constant Λ, or rather $1/\Lambda$ can be interpreted as the distance to singularity of the matrix $\mathbf{M}$, or as the distance to linear dependency of the vectors $\boldsymbol{\Phi}(\mathbf{y}^i)$, $i = 0, \dots, p$, respectively (Conn et al. 2009, Chap. 3).

3.3 BOBYQA

The following description of BOBYQA's main ideas follows the one in Cartis et al. (2021). The BOBYQA algorithm is a trust region method based on a (typically underdetermined) quadratic interpolation model. The training data set defined in (5) contains the objective function evaluations for each sample point $\mathbf{y}^i$, $i = 0, \ldots, p$, and the size of the training data set is given by $|\mathcal{T}| \in [n+2, (n+1)(n+2)/2]$. Note that setting $|\mathcal{T}| = n+1$ is possible in PyBOBYQA, but then a fully linear (not quadratic) interpolation model is applied.

Let $\mathbf{x}^{(k)}$ denote the solution at the k-th iteration. At each iteration a local quadratic model is built, i.e.,

$$f(\mathbf{x}) \approx m^{(k)}(\mathbf{x}) = c^{(k)} + \mathbf{g}^{(k)\top}(\mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^\top \mathbf{H}^{(k)}(\mathbf{x} - \mathbf{x}^{(k)}), \quad (15)$$

fulfilling the interpolation conditions

$$f(\mathbf{y}^j) = m^{(k)}(\mathbf{y}^j) \;\; \forall \mathbf{y}^j \in \mathcal{T}. \quad (16)$$

For $|\mathcal{T}| = (n+1)(n+2)/2$ the interpolation problem is fully determined. For $|\mathcal{T}| < (n+1)(n+2)/2$ the remaining degrees of freedom are set by minimizing the distance between the current and the last approximation of the Hessian $\mathbf{H}$ in the matrix Frobenius norm, i.e.,

$$\min_{c^{(k)}, \mathbf{g}^{(k)}, \mathbf{H}^{(k)}} ||\mathbf{H}^{(k)} - \mathbf{H}^{(k-1)}||_{\mathrm{F}}^2 \quad \text{s.t. (16) holds}, \quad (17)$$

where typically $\mathbf{H}^{(-1)} = \mathbf{0}^{n \times n}$. Once the quadratic model is built, the trust region subproblem

$$\begin{aligned} &\min_{\mathbf{x} \in \mathbb{R}^n} m^{(k)}(\mathbf{x}) \\ &\text{s.t. } ||\mathbf{x} - \mathbf{x}^{(k)}||_2 \leq \Delta^{(k)} \\ &\mathbf{x_{lb}} \leq \mathbf{x} \leq \mathbf{x_{ub}} \end{aligned} \quad (18)$$

is solved, where $\Delta^{(k)} > 0$ denotes the trust region radius. Then, having the optimal solution $\mathbf{x}^{\mathrm{opt}}$ of (18) in the k-th iteration calculated, we check if the decrease in the objective function is sufficient. Therefore, the ratio between actual decrease and expected decrease

$$r^{(k)} = \frac{\text{actual decrease}}{\text{expected decrease}} = \frac{f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{\mathrm{opt}})}{m^{(k)}(\mathbf{x}^{(k)}) - m^{(k)}(\mathbf{x}^{\mathrm{opt}})} \quad (19)$$

is calculated. If the ratio $r^{(k)}$ is sufficiently large, the step is accepted ($\mathbf{x}^{(k+1)} = \mathbf{x}^{\mathrm{opt}}$) and the trust region radius increased ($\Delta^{(k+1)} > \Delta^{(k)}$). Otherwise, the step is rejected ($\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$) and the trust region radius decreased ($\Delta^{(k+1)} < \Delta^{(k)}$).

An important question is now, how to maintain the training data set. An accepted solution is added to $\mathcal{T}$, i.e., $\mathcal{T} = \mathcal{T} \cup \{(\mathbf{y}^{\text{add}}, f(\mathbf{y}^{\text{add}}))\}$ with $\mathbf{y}^{\text{add}} = \mathbf{x}^{\text{opt}}$, and since $|\mathcal{T}|$ is fixed, another data point has to leave the training data set. Hereby, the aim is to achieve the best possible quality of the model. We know from Sect. 3.2 that the quality of the model depends on the training data set and can be expressed by the poisedness constant Λ. Thus, the decision which point is replaced depends on its impact on the Λ-poisedness. Let l^i, $i = 0, \ldots, p$ be the Lagrange polynomials obtained by evaluating (12) and let

$$i^{\text{go}} = \underset{i=0,\ldots,p}{\arg\max} \left(|l^i(\mathbf{y}^{\text{add}})| \max\left\{1, \frac{||\mathbf{y}^i - \mathbf{y}^{\text{add}}||_2^4}{||\Delta^{(k)}||^4}\right\}\right). \tag{20}$$

Then, the point $\mathbf{y}^{i^{\text{go}}}$ is replaced by the new iterate $\mathbf{y}^{\text{add}}$. This means, that the point with the worst (largest) value of the corresponding Lagrange polynomial, evaluated at the new iterate solution, is going to be replaced, i.e., the updated training data set is built by $\mathcal{T} = \mathcal{T} \cup \{(\mathbf{y}^{\text{add}}, f(\mathbf{y}^{\text{add}}))\} \backslash \{(\mathbf{y}^{i^{\text{go}}}, f(\mathbf{y}^{i^{\text{go}}}))\}$. Please note that (20) is the formula used in Powell's and Cartis' implementations, however, in Powell's work (Powell 2009, eq. (6.1)) in numerator and denominator exponent 2 is used instead of exponent 4.

Regardless of the subproblem's optimal solution, sometimes points are exchanged, only in order to improve the training data set. Let $\mathbf{y}^i$ be the training data point which shall be replaced, e.g. the point furthest from the current optimal solution. Then we consider the corresponding Lagrange polynomial l^i and choose a new point by solving

$$\begin{aligned} \mathbf{y}^{\text{new}} &= \max_{\mathbf{y}\in\mathcal{B}} |l^i(\mathbf{y})| \\ \text{s.t. } & \mathbf{x_{lb}} \le \mathbf{y} \le \mathbf{x_{ub}} \\ & ||\mathbf{y} - \mathbf{x}^{\text{opt}}|| \le \Delta^{(k)} \end{aligned} \tag{21}$$

For more details we refer to the original work by Powell (2009).

3.3.1 Solving the linear system

For the Hermite least squares method we modify BOBYQA's linear system resulting from the interpolation conditions (16). From this solution, the coefficients $c^{(k)}$, $\mathbf{g}^{(k)}$ and $\mathbf{H}^{(k)}$ of the quadratic subproblem (15) are determined. Therefore, we recall how the linear system is build in PyBOBYQA (Cartis et al. (2019). We denote the current optimal solution by $\mathbf{x}^{\text{opt}} \in \mathcal{T}$. Without limitation of generality we assume that $\mathbf{x}^{\text{opt}} = \mathbf{y}^p$. The uniquely solvable system (i.e. $p_1 = q_1 = (n+1)(n+2)/2$) reads as follows

$$\mathbf{M}_{\text{I}}\mathbf{v}^{(k)} = \mathbf{b}_{\text{I}} \tag{22}$$

with

$$\mathbf{M}_{\text{I}} \in \mathbb{R}^{p\times q} \text{ with the entries } M_{\text{I},i,j} = \phi_j(\mathbf{y}^{i-1} - \mathbf{x}^{\text{opt}}), \tag{23}$$

$$\mathbf{b}_{\text{I}} \in \mathbb{R}^{p} \text{ with the entries } b_{\text{I},i} = f(\mathbf{y}^{i-1}) - f(\mathbf{x}^{\text{opt}}), \tag{24}$$

$$\text{and } \mathbf{v}^{(k)} = \begin{pmatrix} \mathbf{g}^{(k)} \\ \mathbf{H}^{(k)\star} \end{pmatrix} \in \mathbb{R}^q, \tag{25}$$

where $\mathbf{H}^{(k)\star}$ is a vector in $\mathbb{R}^{(n^2+n)/2}$ containing the lower triangular and the diagonal elements of the diagonal matrix $\mathbf{H}^{(k)}$. The constant part is set to $c^{(k)} = f(\mathbf{x}^{\text{opt}})$.

The case $n+2 \leq p_1 < (n+1)(n+2)/2$ is not of further interest for the construction of the Hermite least squares system. Hence, we refrain from a detailed description here and refer to Cartis et al. (2019).

3.3.2 Convergence

The convergence theory for gradient based optimization algorithms like SQP is typically based on error bounds of the Taylor expansion, in order to show the decreasing error between the model $m^{(k)}(\mathbf{x})$ and the function $f(\mathbf{x})$, between the corresponding derivatives. In DFO the poisedness constant Λ can be used to formulate a Taylor type error bound. In Conn et al. (2008a) an error bound is given by

$$||\nabla m^{(k)}(\mathbf{x}) - \nabla f(\mathbf{x})|| \leq \frac{1}{(d+1)!} G \Lambda \sum_{i=0}^{p} ||\mathbf{y}^i - \mathbf{x}||^{d+1}, \tag{26}$$

where G is a constant depending only on the function f and d is the polynomial degree of the approximation model (i.e. here $d = 2$). Thus, in order to apply the convergence theory of gradient based methods to DFO methods, it is required to keep Λ uniformly bounded for all training data sets used within the algorithm.

The algorithm in (Conn et al. 2009, Algo. 11.2) is a modified version of Powell's DFO method, such that global convergence can be proven. However, in contrast to the original BOBYQA method, bound constraints are not considered in Conn's version. Hence, Conn's algorithm is rather an adaptation of Powell's UOBYQA (Unconstrained Optimization BY Quadratic Approximation) method (Powell 2002). Since UOBYQA is similar to the BOBYQA method described in this section (but without considering constraints), we refrain from a full description. For more details, we refer to the original work by Powell (Powell 2002), or the convergent modification by Conn (Conn et al. 2009, Chap. 11.3).

For the global convergence proof of Conn's version, the Λ-poisedness plays an important role. One key aspect is the fact, that the interpolation set is Λ-poised in each step of the optimization algorithm – or, within a final number of steps, it can be transformed into a Λ-poised set (using the so-called model improvement algorithm (Conn et al. 2009, Algo. 6.3)). Since Powell's and Cartis' versions allow bound constraints, the convergence proof from (Conn et al. 2009) cannot be simply applied. Even if bound constraints were not provided in these algorithms, global convergence could not be proven, since the Λ-poisedness of the interpolation set is not always guaranteed. They apply strategies to update the training data set, which *hopefully* reduce the poisedness constant Λ, but they do not provide bounds (Conn et al. 2008a). Therefore, they would require Λ-poisedness checks more often and re-evaluations of the whole training data set in situations of poorly balanced training data sets, i.e., an usage of the model improvement algorithm. However, to feature bound constraints and for the benefit of less computational effort and thus, efficiency, they abdicate on provable convergence and rely on heuristics, when and how to check and improve Λ-poisedness.

4 Hermite least squares method

In *Hermite interpolatio*n a linear system is solved in order to find a polynomial approximation of a function, considering function values and partial derivative values in given training data points, cf. (Hermann 2011, Chap. 6.6) or Sauer and Xu (1995). In the following we will build such a system, but with more information than required for a uniquely solvable Hermite interpolation and solve it with least squares regression. Thus, we call the optimization approach based on this technique *Hermite least squares* optimization.

As mentioned in Sect. 2 we assume that we know some partial derivatives of the objective function $f : \mathbb{R}^n \to \mathbb{R}$, i.e., we can calculate them with negligible computational effort compared to the effort of evaluating the objective function itself. We focus on the information in (2), i.e., we neglect the second order derivatives. Our *Hermite* training data set is then given by

$$\mathcal{T}_{\mathrm{H}} = \left\{ \left(\mathbf{y}^0, f(\mathbf{y}^0), \frac{\partial}{\partial y_1} f(\mathbf{y}^0), \dots, \frac{\partial}{\partial y_{n_{\mathrm{kd}}}} f(\mathbf{y}^0) \right), \dots \right.$$
$$\left. \dots, \left(\mathbf{y}^p, f(\mathbf{y}^p), \frac{\partial}{\partial y_1} f(\mathbf{y}^p), \dots, \frac{\partial}{\partial y_{n_{\mathrm{kd}}}} f(\mathbf{y}^p) \right) \right\}. \tag{27}$$

We want to use this additional information in order to improve the quadratic model. BOBYQA's simple interpolation (22) is extended with derivative information yielding least squares regression. First we introduce this approach starting with a uniquely solvable interpolation problem, i.e., the number of training data points p_1 coincides with the dimension of the basis q_1. Then, we allow to reduce the number of training data points such that the initial interpolation problem is underdetermined, i.e., $p_1 < q_1$. While for $p_1 = q_1$ global convergence can be ensured (in the unconstrained case), we observe superior performance for $p_1 < q_1$, see numerical results in Sect. 5.1.

4.1 Build upon interpolation ($p_1 = q_1$)

First, we consider a training data set with $|\mathcal{T}_{\mathrm{I}}| = p_1 \equiv q_1$, i.e., we could solve an interpolation problem as in (22) based only on function evaluations in order to obtain the quadratic model in (15). Instead, we provide additionally derivative information for the first n_{kd} partial derivatives of each training data point, i.e., we consider $\mathcal{T}_{\mathrm{H}}$ with $|\mathcal{T}_{\mathrm{H}}| = p_1(1+n_{\mathrm{kd}})$, where p_1 is the number of data points and $|\mathcal{T}_{\mathrm{H}}|$ denotes the *number of information*. We extend the system with the gradient information available in form of additional lines for the system matrix and the right hand side and obtain

$$\mathbf{M}_\mathrm{H} = \begin{pmatrix} \mathbf{M}_\mathrm{I} \\ \mathbf{M}_\mathrm{H}^{(1)} \\ \vdots \\ \mathbf{M}_\mathrm{H}^{(n_\mathrm{kd})} \end{pmatrix} \quad \text{and} \quad \mathbf{b}_\mathrm{H} = \begin{pmatrix} \mathbf{b}_\mathrm{I} \\ \mathbf{b}_\mathrm{H}^{(1)} \\ \vdots \\ \mathbf{b}_\mathrm{H}^{(n_\mathrm{kd})} \end{pmatrix}, \tag{28}$$

where $\mathbf{M}_\mathrm{I}$ and $\mathbf{b}_\mathrm{I}$ are defined in (23) and (24), respectively, and the entries of the submatrices $\mathbf{M}_\mathrm{H}^{(k)}$ and $\mathbf{b}_\mathrm{H}^{(k)}$, $k = 1, \ldots, n_\mathrm{kd}$, are given by

$$M_{\mathrm{H},i,j}^{(k)} = \frac{\partial}{\partial y_k} \phi_j (\mathbf{y}^{i-1} - \mathbf{x}^\mathrm{opt}) \tag{29}$$

and

$$b_{\mathrm{H},i}^{(k)} = \frac{\partial}{\partial y_k} f(\mathbf{y}^{i-1}). \tag{30}$$

Solving the overdetermined linear system

$$\mathbf{M}_\mathrm{H} \mathbf{v}^{(k)} \overset{\text{l.s.}}{=} \mathbf{b}_\mathrm{H} \tag{31}$$

using least squares regression yields a quadratic model for the trust region subproblem ($\mathbf{v}^{(k)}$ defined as in (25)). The formulation of the system matrix $\mathbf{M}_\mathrm{H}$ and the right hand side $\mathbf{b}_\mathrm{H}$ in case of second order derivatives is given in Appendix A.

An optional step is the weighting of the least squares information, cf. weighted regression (Björck 1996). Information belonging to a training data point close to the current solution could be given more weight, information belonging to a training data point far from the current solution could be given less weight. However, since we could not observe significant improvements in the numerical tests, weighting has not been further investigated in this work.

In the following, we will discuss how good the quadratic model resulting from solving (31) is. Therefore, we state the following theorem, which generalizes theorem 4.1 in Conn et al. (2008b).

Theorem 1 *Given a poised training data set $\mathcal{T}_\mathrm{I}$ and the monomial basis Φ with $|\mathcal{T}_I| = |\Phi|$, and $\mathcal{B} \subset \mathbb{R}^n$. Let $\mathbf{M}_I$ be the corresponding system matrix of the interpolation problem and $\mathbf{b}_\mathrm{I}$ the right hand side, respectively. Let $\mathcal{T}_\mathrm{R} \supset \mathcal{T}_\mathrm{I}$ be a training set containing further information. If $\mathcal{T}_\mathrm{I}$ is Λ-poised in $\mathcal{B}$ in the interpolation sense, then $\mathcal{T}_\mathrm{R}$ is at least Λ-poised in $\mathcal{B}$ in the regression sense.*

Proof The additional information can be added in form of additional lines to the system matrix and the right hand side. Thus, we can set the system matrix and the right hand side of the regression problem corresponding to $\mathcal{T}_\mathrm{R}$ to

$$\mathbf{M}_\mathrm{R} = \begin{pmatrix} \mathbf{M}_\mathrm{I} \\ \mathbf{M}_\mathrm{add} \end{pmatrix} \quad \text{and} \quad \mathbf{b}_\mathrm{R} = \begin{pmatrix} \mathbf{b}_\mathrm{I} \\ \mathbf{b}_\mathrm{add} \end{pmatrix}. \tag{32}$$

Since $\mathcal{T}_\mathrm{I}$ is Λ-poised in the interpolation sense, by Definition 1 holds

$$\forall \mathbf{x} \in \mathcal{B}\ \exists \mathbf{l}_\mathrm{I}(\mathbf{x}) \in \mathbb{R}^{|\mathcal{T}_\mathrm{I}|} \text{ s.t. } \sum_{\mathbf{y}^i \in \mathcal{T}_\mathrm{I}} l_\mathrm{I}^i(\mathbf{x})\mathbf{m}_\mathrm{I}^i = \boldsymbol{\Phi}(\mathbf{x}) \quad \text{with} \quad \|\mathbf{l}_\mathrm{I}(\mathbf{x})\|_\infty \leq \Lambda, \tag{33}$$

where $\mathbf{m}_\mathrm{I}^i$ is the i-th column of $\mathbf{M}_\mathrm{I}^\top$. We define $\mathbf{l}_\mathrm{R}(\mathbf{x}) = (\mathbf{l}_\mathrm{I}(\mathbf{x}), \mathbf{0})^\top \in \mathbb{R}^{|\mathcal{T}_\mathrm{R}|}$. Then

$$\sum_{\mathbf{y}^i \in \mathcal{T}_\mathrm{R}} l_\mathrm{R}^i(\mathbf{x})\mathbf{m}_\mathrm{R}^i = \boldsymbol{\Phi}(\mathbf{x}) \tag{34}$$

holds and $\|\mathbf{l}_\mathrm{R}(\mathbf{x})\|_\infty$ is bounded by Λ, since

$$\|\mathbf{l}_\mathrm{R}(\mathbf{x})\|_\infty = \max_{i=0,\ldots,|\mathcal{T}_\mathrm{R}|} |l_\mathrm{R}^i(\mathbf{x})| \overset{\text{Def. } \mathbf{l}_\mathrm{R}}{=} \max_{i=0,\ldots,|\mathcal{T}_\mathrm{I}|} |l_\mathrm{I}^i(\mathbf{x})| = \|\mathbf{l}_\mathrm{I}(\mathbf{x})\|_\infty \overset{(33)}{\leq} \Lambda. \tag{35}$$

□

Theorem 1 shows, that it is enough to ensure that a subset of the regression data set is Λ-poised in the interpolation sense, and then we can deduce that the regression data set is at least Λ-poised in the regression sense. Thus, although there is no analogue of the model improvement algorithm for the regression case (Conn et al. 2009, Chap. 6), we can apply the model improvement algorithm (Conn et al. 2009, Algo. 6.3) for interpolation and the optimization algorithm (Conn et al. 2009, Algo. 11.2). This ensures Λ-poisedness in the interpolation sense of a subset with $|\Phi| = (n+1)(n+2)/2$ points, and then we build the quadratic model with least squares regression. And since $l_\mathrm{R}^i(\mathbf{x}) = 0$ for $i > |\mathcal{T}_\mathrm{I}|$, the type of additional information in $\mathcal{T}_\mathrm{R}$ has no impact on the proof – as long as the matrix $\mathbf{M}_\mathrm{R}$ has full column rank. This implies, that instead of additional data points and their function evaluations, we can also add derivative information according to (28), and our training data set remains at least Λ-poised. The proof of convergence from (Conn et al. 2009) (holding for the unconstrained case) remains unaffected. In practice we expect faster convergence due to better quadratic models.

4.2 Build upon underdetermined interpolation ($p_1 < q_1$)

For fully determined quadratic interpolation, a large set of training data points is required ($p_1 = |\mathcal{T}_\mathrm{I}| = (n+1)(n+2)/2$), such that the linear system is uniquely solvable. Since in Hermite least squares we have additional gradient information we can reduce the number of training data points and still have a determined or overdetermined regression system. The number of rows in the Hermite least squares system is given by $p_1(1 + n_\mathrm{kd}) - 1$ and has to be larger than the number of columns, i.e., $q = |\Phi| - 1 = (n+1)(n+2)/2 - 1$, cf. (28). Thus, the required number of training data points in the Hermite least squares is only

$$p_1 \geq \left\lceil \frac{(n+1)\,(n+2)}{2\,(1+n_\mathrm{kd})} \right\rceil. \tag{36}$$

This allows the Hermite least squares system to be built as in (28–31), with the only difference that $p_1 < q_1$. Since the regression data set does not contain a subset of Λ-poised interpolation points anymore, the model improvement algorithm cannot be applied to a subset. Thus, even in the unconstrained case, the scheme of the formal convergence proof from (Conn et al. 2009) cannot be transferred. In the next subsection we will discuss how to build and maintain the training data set, taking into account the derivative information.

4.3 Λ-poisedness for Hermite least squares

We aim to include the derivative information into the updating procedure of the training data set. We start with the Hermite interpolation setting introduced in Sect. 4.1, i.e., initially we set

$$p_1 = \frac{(n+1)(n+2)}{2(1+n_{\text{kd}})} \tag{37}$$

s.t. $|\mathcal{T}_\text{H}| = q_1$, implying that the system matrix $\mathbf{M}_\text{H}$ is quadratic and hence, (31) is a uniquely solvable Hermite interpolation problem. We adapt the definition of Λ-poisedness to the Hermite interpolation case. However, the following definition does not guarantee the required error bounds for provable convergence (cf. (Conn et al. 2009, Chap. 6.1)). This leads to an approach, without formal convergence proof such as the common BOBYQA implementations.

Definition 3 (Λ-poisedness in the Hermite interpolation sense) Given a poised Hermite interpolation problem as defined above with p_1 training data points, the training data set $\mathcal{T}_\text{H}$ and the monomial basis Φ with $|\mathcal{T}_\text{H}| = p_1(1+n_{\text{kd}}) = q_1 = |\Phi|$. Let $\mathcal{B} \subset \mathbb{R}^n$ and $\Lambda > 0$. Then the training data set $\mathcal{T}_\text{H}$ is Λ-poised in $\mathcal{B}$ (in the Hermite interpolation sense) if and only if

$$\forall \mathbf{x} \in \mathcal{B}\ \exists \mathbf{l}(\mathbf{x}) \in \mathbb{R}^{q_1} \text{ s.t. } \quad \mathbf{M}_\text{H}^\top \mathbf{l}(\mathbf{x}) = \boldsymbol{\Phi}(\mathbf{x}) \quad \text{with} \quad ||\mathbf{l}(\mathbf{x})||_\infty \leq \Lambda. \tag{38}$$

We will define Lagrange-type polynomials for Hermite interpolation and show that they solve (38).

Definition 4 (Lagrange-type polynomial for Hermite interpolation) Let $\mathbf{M}_\text{H} \in \mathbb{R}^{q_1 \times q_1}$ be a Hermite interpolation matrix with respect to the basis Φ as defined in (28) and $\mathbf{e}^i \in \mathbb{R}^{q_1}$ the i-th unit vector. Let $\boldsymbol{\lambda}^i$ solve

$$\mathbf{M}_\text{H} \boldsymbol{\lambda}^i = \mathbf{e}^{i+1}. \tag{39}$$

Then, the polynomial built with the coefficients of $\boldsymbol{\lambda}^i$ and the basis Φ

$$t^i(\mathbf{x}) = \lambda_0^i \phi_0(\mathbf{x}) + \cdots + \lambda_q^i \phi_q(\mathbf{x}), \tag{40}$$

defines the i-th Lagrange-type polynomial for Hermite interpolation.

Lemma 1 *Let* $\mathbf{t}(\mathbf{x}) = \left(t^0(\mathbf{x}), \dots, t^q(\mathbf{x})\right)^\top$ *be defined as in* (40)*,* $\Phi(\mathbf{x})$ *defined as in Sect.* 3.1 *and* $\mathbf{M}_\mathrm{H}$ *from* (28)*. Then,* $\mathbf{t}(\mathbf{x})$ *solves* $\mathbf{M}_\mathrm{H}^\top \mathbf{l}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x})$*, i.e.,* $\mathbf{t}(\mathbf{x}) \equiv \mathbf{l}(\mathbf{x})$*.*

Proof We rewrite (39) into

$$\mathbf{M}_\mathrm{H}\mathbf{T} = \mathbf{I}, \tag{41}$$

where $\mathbf{I}$ denotes the $q_1 \times q_1$ identity matrix and $\mathbf{T}$ is defined by the solution vectors of (39), i.e.,

$$\mathbf{T} = \begin{pmatrix} | & & | \\ \boldsymbol{\lambda}^0 & \dots & \boldsymbol{\lambda}^q \\ | & & | \end{pmatrix}. \tag{42}$$

Starting from (41), changing the order of the matrices, we derive

$$\mathbf{T}\mathbf{M}_\mathrm{H} = \mathbf{I}.$$

Left multiplication by $\mathbf{\Phi}(\mathbf{x})^\top$ yields

$$\mathbf{\Phi}(\mathbf{x})^\top \mathbf{T}\mathbf{M}_\mathrm{H} = \mathbf{\Phi}(\mathbf{x})^\top.$$

We apply (42)

$$\left(\phi_0(\mathbf{x}), \dots, \phi_q(\mathbf{x})\right) \begin{pmatrix} | & & | \\ \boldsymbol{\lambda}^0 & \dots & \boldsymbol{\lambda}^q. \\ | & & | \end{pmatrix} \mathbf{M}_\mathrm{H} = \left(\phi_0(\mathbf{x}), \dots, \phi_q(\mathbf{x})\right)$$

and (40)

$$\left(t^0(\mathbf{x}), \dots, t^q(\mathbf{x})\right) \mathbf{M}_\mathrm{H} = \left(\phi_0(\mathbf{x}), \dots, \phi_q(\mathbf{x})\right).$$

Transposing yields

$$\mathbf{M}_\mathrm{H}^\top \begin{pmatrix} t^0(\mathbf{x}) \\ \vdots \\ t^q(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \phi_0(\mathbf{x}) \\ \vdots \\ \phi_q(\mathbf{x}) \end{pmatrix},$$

which is per definition equivalent to

$$\mathbf{M}_\mathrm{H}^\top \mathbf{t}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x}).$$

Thus, $\mathbf{t}$ solves (38). $\square$

It holds that the (uniquely defined) polynomial solving the Hermite interpolation problem $\mathbf{M}_{\mathrm{H}}\mathbf{v} = \mathbf{b}_{\mathrm{H}}$ with $\mathbf{M}_{\mathrm{H}} \in \mathbb{R}^{q_1 \times q_1}$ and $\mathbf{b}_{\mathrm{H}} \in \mathbb{R}^{q_1}$ can be written as

$$m_{\mathrm{H}}(\mathbf{x}) = \sum_{i=0}^{p} f(\mathbf{y}^i) t^i(\mathbf{x}) + \sum_{i=0}^{p} \sum_{j=1}^{n_{\mathrm{kd}}} \frac{\partial f}{\partial x_j}(\mathbf{y}^i) t^{jp_1-1+i}(\mathbf{x}), \tag{43}$$

where $p_1 = p + 1$ is the number of training data points and $q_1 = (1 + n_{\mathrm{kd}})p_1$.

Now, let us investigate Hermite least squares as introduced in Sect. 4.2, i.e., the case $|\mathcal{T}_{\mathrm{H}}| > q_1$, implying

$$p_1 > \frac{(n+1)(n+2)}{2(1+n_{\mathrm{kd}})}. \tag{44}$$

Extending the concept above, the Lagrange-type polynomials for Hermite least squares are obtained by solving

$$\mathbf{M}_{\mathrm{H}}\boldsymbol{\lambda}^i \overset{\text{l.s.}}{=} \mathbf{e}^{i+1}. \tag{45}$$

instead of (39). We maintain the training data set based on Λ-poisedness in the Hermite least squares sense. This means, we maximize (20) over the first p_1 Lagrange polynomials and replace the chosen data point with all corresponding information (i.e. function value and derivative information) by the new data point.

4.4 Scaling

In this section we will discuss some preconditioning steps for solving the linear equations systems. In PyBOBYQA (Cartis et al. 2019) the system is scaled in the following way: instead of

$$\mathbf{M}\mathbf{v} = \mathbf{b} \tag{46}$$

the system

$$\mathbf{L}\mathbf{M}\mathbf{R}\mathbf{R}^{-1}\mathbf{v} = \mathbf{L}\mathbf{b} \tag{47}$$

is solved, where $\mathbf{L}$ and $\mathbf{R}$ are diagonal matrices of the same dimension as $\mathbf{M}$. Each training data point entry $\mathbf{y}^i$ is scaled by the factor $1/\Delta$, where Δ is the trust region radius of the current step (for simplicity of notation we omit the index k for the current iteration for both the trust region radius and the system). Thus, the scaling matrices in PyBOBYQA (with $p_1 = q_1$) are given by

$$\mathbf{L} = \mathbf{I} \;\text{ and }\; \mathbf{R} = \operatorname{diag}\Bigg(\underbrace{\frac{1}{\Delta} \;\cdots\; \frac{1}{\Delta}}_{p} \;\underbrace{\frac{1}{\Delta^2} \;\cdots\; \frac{1}{\Delta^2}}_{p-n}\Bigg), \tag{48}$$

i.e., the columns for the linear part are scaled by $1/\Delta$, the columns for the quadratic part by $1/\Delta^2$. Preserving the same scaling scheme, the following left scaling matrix is obtained for the Hermite least squares approach

$$\mathbf{L} = \mathrm{diag}(\underbrace{1 \ \dots \ 1}_{p} \ \underbrace{\Delta \ \dots \ \Delta}_{p_1 n_{\mathrm{kd}}}), \tag{49}$$

while the right scaling matrix remains unchanged as in (48).

5 Numerical tests

A test set of 29 nonlinear, bound constrained optimization problems with 2, 3, 4, 5 and 10 dimensions has been evaluated and compared. The complete test set and the detailed results can be found at GitHub (Fuhrländer and Schöps 2022). As reference solution we consult the solution of PyBOBYQA (Cartis et al. 2019) using the default setting for the size of the training data set, i.e., $p_1 = 2n+1$, in the following referenced to as PyBOBYQA. For all remaining parameters we also apply the default settings. Another reference solution is SQP, where the unknown derivatives are calculated with finite differences. Therefore, the SciPy implementation of the SLSQP algorithm from (Kraft et al. 1988) has been used. Please note that this is a completely different implementation, thus a direct comparison should be handled with caution. In the Hermite least squares approach, numerical tests showed that

$$p_1 = \max\left(2n + 1 - n_{\mathrm{kd}}, \left\lceil \frac{(n+1)(n+2)}{2(1+n_{\mathrm{kd}})} \right\rceil\right). \tag{50}$$

is a reasonable choice for the number of training data points. We vary the number of known derivative directions n_{kd} and always assume that these derivative directions are then available for all training data points.

In the tests we are interested in two aspects: 1) do we find an optimal solution and 2) how much computing effort is needed. For 1) we check if we find the same solution as the reference methods. Please note that we considered only test functions for which the reference PyBOBYQA method was able to find the optimal solution. For 2) we compare the total number of objective function evaluations during the whole optimization process.

5.1 Results

Before we evaluate the complete test set, we analyze the accuracy of the quadratic model $m^{(k)}(\mathbf{x})$ which is built in each iteration. Let us consider the Rosenbrock function in $\mathbb{R}^2$, given by

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \tag{51}$$

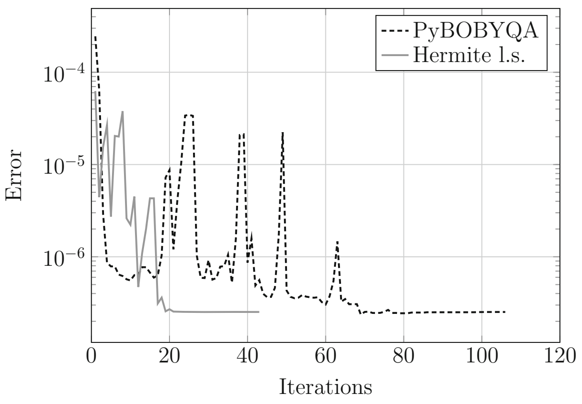


Fig. 1 Error of the quadratic model $m^{(k)}$ in each iteration compared to second order Taylor expansion, cf. (52) with $\delta = 0.01$. Comparison between proposed method Hermite least squares and reference method PyBOBYQA

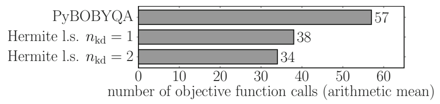
We assume $\partial f/\partial x_1$ to be unknown and $\partial f/\partial x_2$ to be available. The second order Taylor expansion $T_2 f(\mathbf{x}; \mathbf{x}^{(k)})$ in $\mathbf{x}^{(k)}$ is considered as the reference model. We investigate the error between this Taylor reference and the quadratic model of PyBOBYQA and of Hermite least squares, respectively, evaluated by using the L_2-norm

$$\|m^{(k)}(\mathbf{x}) - T_2 f(\mathbf{x}; \mathbf{x}^{(k)})\|_{L_2}^2 = \int_{\mathbf{x}^{(k)}-\delta}^{\mathbf{x}^{(k)}+\delta} |m^{(k)}(\mathbf{x}) - T_2 f(\mathbf{x}; \mathbf{x}^{(k)})|^2 \, \mathrm{d}\mathbf{x}. \tag{52}$$

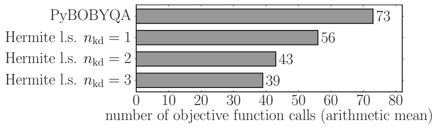
In Fig. 1, the resulting error is plotted over the number of iterations. We observe that the error of the quadratic model decreases first for the Hermite least squares model. After 20 iterations, the error remains below $2.5 \cdot 10^{-7}$. For PyBOBYQA the error is also reduced to this magnitude, but it takes 65 iterations. The errors of the quadratic models reflect the performance of the different optimization methods. Both find the same optimal solution. Hermite least squares is more efficient, it terminates after 43 iterations. The reference method PyBOBYQA terminates after 106 iterations. Here, the number of objective function calls is proportional to the number of iterations.

Test set In the following, the test set from (Fuhrländer and Schöps 2022) is evaluated. The results are consistent with the observations regarding error and performance for the Rosenbrock function, which have been described above. In Figs. 2 and 3 the numerical results for the test set are visualized. In Fig. 2 the arithmetic mean of the number of objective function calls is considered, in Fig. 3 the geometric mean, respectively. We compare PyBOBYQA with Hermite least squares and vary the number of known derivatives n_{kd}. For example, in Fig. 2b for Hermite least squares with $n_{\mathrm{kd}} = 2$ we average over all 3-dimensional test problems, solved with Hermite least squares, with three cases each, i.e., 1) $\partial f/\partial x_1$ and $\partial f/\partial x_2$ are known, 2) $\partial f/\partial x_1$ and $\partial f/\partial x_3$ are known and 3) $\partial f/\partial x_2$ and $\partial f/\partial x_3$ are known. For the 10-dimensional test problems we tested three random permutations of known derivatives per n_{kd}.

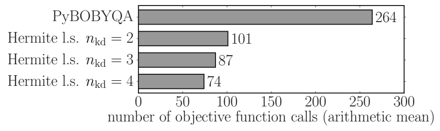
For $n = 4$ and $n_{\mathrm{kd}} = 1$ some instances did not terminate within the limit of 2000 objective function calls using Hermite least squares. Hence, this case is left out in Figs. 2c and 3c, respectively. However, the corresponding results are reported in Fuhrländer and Schöps (2022).



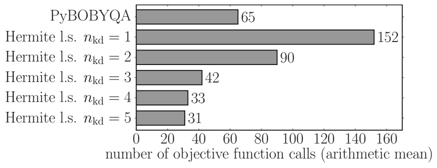
(a) 2-dimensional test problems.



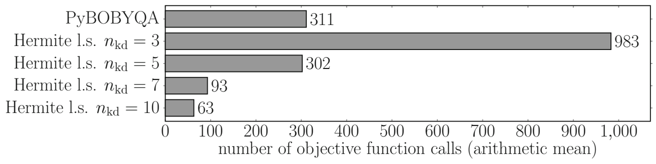
(b) 3-dimensional test problems.



(c) 4-dimensional test problems.

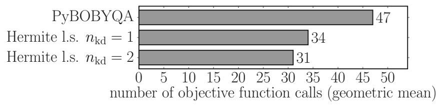


(d) 5-dimensional test problems.

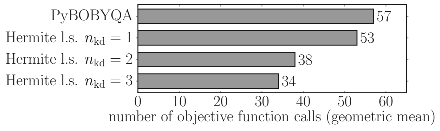


(e) 10-dimensional test problems.

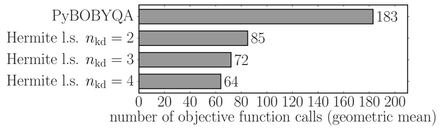
Fig. 2 Arithmetic mean of the number of function evaluations for all test problems solved with the reference method PyBOBYQA and Hermite least squares (Hermite l.s.) with varying number of known derivatives n_{kd}



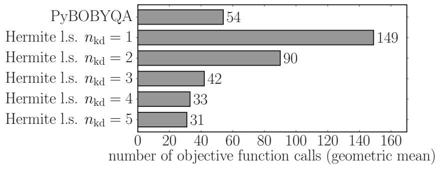
(a) 2-dimensional test problems.



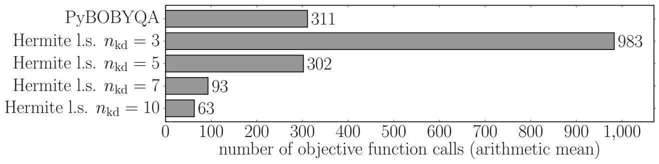
(b) 3-dimensional test problems.



(c) 4-dimensional test problems.



(d) 5-dimensional test problems.



(e) 10-dimensional test problems.

Fig. 3 Geometric mean of the number of function evaluations for all test problems solved with the reference method PyBOBYQA and Hermite least squares (Hermite l.s.) with varying number of known derivatives n_{kd}

We observe that if less than the half of the derivative directions are known, i.e., $n_{\text{kd}} < \frac{1}{2}n$, the Hermite least squares method is not reliably better than BOBYQA. However, if at least the half of the derivatives are known, we can significantly save computing effort by the proposed Hermite method. For $n_{\text{kd}} \geq \frac{1}{2}n$, with Hermite least squares the number of objective function calls can be reduced by 34 %−80 % compared to PyBOBYQA, depending on dimension n and number of known derivatives n_{kd}, see Fig. 2. One exception is the case $n = 10$ and $n_{\text{kd}} = 5$, where we can only see a reduction by 3 %. Although, we observe that there are instances of some test problems for which the number of objective function calls increase (cf. Fuhrländer and Schöps (2022)), in average the computational effort can be significantly reduced. The optimal solution has been found in all considered cases. As expected, we observe that the more gradient information we have, the less objective function evaluations are needed within one optimization run using Hermite least squares. We can conclude that, assuming about the half or more of the partial derivatives are known, using the Hermite least squares approach instead of the classic PyBOBYQA method reduces the computational effort significantly.

In the numerical tests, we also compared the reference BOBYQA and the proposed Hermite modification with SQP. While for Hermite least squares we took PyBOBYQA from Cartis et al. (2019) as a basis and included the required modifications for the Hermite approach, the SQP method from SciPy based on Kraft et al. (1988) is a different implementation, using for example different ways to solve the quadratic subproblem. However, we could observe that in almost all cases, the SQP method required a lower number of objective function calls than PyBOBYQA in order to find the optimal solution, and in most cases also less than Hermite least squares. Please note that in the SQP method we provided the known derivatives and only calculated the remaining ones with finite differences.

5.2 Noisy data

Let us consider the Rosenbrock function from (51) and investigate the performance of the different methods under noise. In accordance to Cartis et al. (2019), for that purpose we add random statistical noise to the objective function value and the derivative values by multiplying the results with the factor $1 + \xi$, where ξ is a uniformly distributed random variable, i.e., $\xi \sim \mathcal{U}(-10^{-2}, 10^{-2})$. Again, for SQP and Hermite least squares we assume $\partial f/\partial x_1$ to be unknown and $\partial f/\partial x_2$ to be available.

The optimal solution of (51) is

$$\mathbf{x}^{\text{opt}} = (1, 1) \;\text{ with }\; f\left(\mathbf{x}^{\text{opt}}\right) = 0. \tag{53}$$

We start the optimization with

$$\mathbf{x}^{\text{start}} = (1.2, 2) \;\text{ with }\; f\left(\mathbf{x}^{\text{start}}\right) = 31.4. \tag{54}$$

First we apply the Hermite least squares method. It terminates after only 37 function calls with the optimal solution

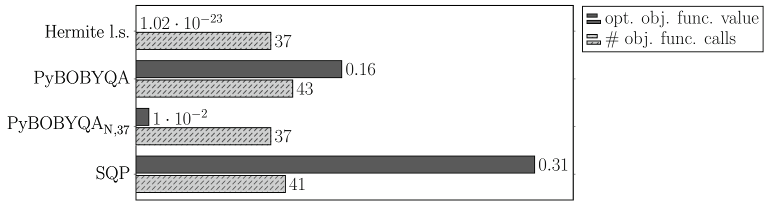


Fig. 4 Results for the optimization of the noisy Rosenbrock function (51), solved with Hermite least squares (Hermite l.s.), the reference method PyBOBYQA, the reference PyBOBYQA method for noisy data with maximum budget 37 (PyBOBYQA$_{N,37}$) and the reference SQP method

$$\mathbf{x}^{\text{H.l.s.}} = (1, 1) \text{ with } f\left(\mathbf{x}^{\text{H.l.s.}}\right) = 1.02 \cdot 10^{-23}. \tag{55}$$

Without noise, a similar number of function calls (namely 43) were needed to find the optimum. Hence, the noise did not lead to an increase in computing effort. We compare these results to the reference solution PyBOBYQA. After 43 objective function calls the algorithm terminates without reaching the optimum

$$\mathbf{x}^{\text{PyB}} = (1.41, 1.98) \text{ with } f\left(\mathbf{x}^{\text{PyB}}\right) = 0.16. \tag{56}$$

Additionally to PyBOBYQA we consider the PyBOBYQA version for noisy data from (Cartis et al. 2019, Sect. 7) as reference solution PyBOBYQA$_\text{N}$. The main differences compared to PyBOBYQA are another choice of default parameters for adjusting the trust region radius (better suited for noisy data), sample averaging and multiple restarts. Even with a high budget of 2000 objective function calls the algorithm does not terminate. In order to compare the results with Hermite least squares, we set the budget to the number of required objective function calls to terminate the Hermite least squares method, i.e., to 37, and evaluate PyBOBYQA$_\text{N}$

$$\mathbf{x}^{\text{PyBN,37}} = (1.08, 1.17) \text{ with } f\left(\mathbf{x}^{\text{PyBN,100}}\right) = 0.01. \tag{57}$$

This means, the optimum could not be sufficiently identified within this budget. Finally, we apply the gradient based SQP. It terminates after 41 objective function calls, and even though the solution has improved, the optimum could not be reached

$$\mathbf{x}^{\text{SQP}} = (0.21, -0.00) \text{ with } f\left(\mathbf{x}^{\text{SQP}}\right) = 0.31. \tag{58}$$

The results are visualized in Fig. 4. We conclude that the gradient based solver SQP fails as to be expected in optimizing the noisy Rosenbrock function. While the standard PyBOBYQA method also terminates without reaching the optimum, the noisy version PyBOBYQA$_\text{N}$ approaches the optimum, but does not terminate. The regression approach in the Hermite least squares method robustify the optimization under noisy data. It achieves the optimal solution at low computational costs (only 37 objective function calls).

Table 1 Number of objective function calls for Rosenbrock function (51) depending on the set of known derivatives and the usage of first and second order derivatives or first order derivatives only

	$\mathcal{I}_{\mathrm{d}}$ resp. $\mathcal{I}_{2\mathrm{d}}$		
	[0]	[1]	[0, 1]
1st order	67	43	40
2nd order	62	40	38

5.3 Hermite least squares with second order derivatives

Again we consider the Rosenbrock function (51) as test function and investigate the usage of second order derivatives, according to the formulation in Appendix A. In this example we observe that the usage of second derivatives additionally to first derivatives slightly reduces the computing effort. The results are given in Table 1. Since second order derivatives are rarely available in practice, so we do not further extend the numerical tests.

6 A practical example: yield optimization

In this section we discuss a practical example where the case of known and unknown gradients occur. In the field of uncertainty quantification, yield optimization is a common task (Graeb 2007). In the design process of a device, e.g. antennas, electrical machines or filters, geometry and material parameters are chosen such that predefined requirements are fulfilled. However, in the manufacturing process, there are often uncertainties which lead to a deviation in the optimized design parameters and this may cause a violation of the requirements. The aim of yield estimation is the quantification of the impact of this uncertainty. The yield defines the probability, that the device still fulfills the performance requirements, under consideration of the manufacturing uncertainties. Thus, the natural goal is to maximize the yield. Please note, the task of yield maximization is equivalent to the task of failure probability minimization. We will formally introduce the yield and discuss the task of yield optimization with an example from the field of electrical engineering: a simple dielectrical waveguide as depicted in Fig. 5. The model of the waveguide originates from (Loukrezis 2019), and was used for yield optimization previously, e.g. in Fuhrländer et al. (2020).

The waveguide has four design parameters, which shall be modified. Two uncertain geometry parameters: the length of the inlay p_1 and the length of the offset p_2. And two deterministic material parameters: d_1 with impact on the relative permittivity and d_2 with impact on the relative permeability. The uncertain parameters are modeled

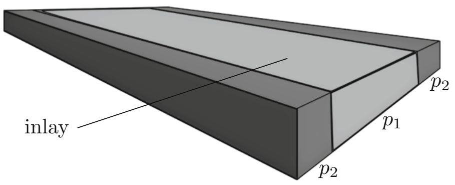


Fig. 5 Model of a simple waveguide with dielectrical inlay and two geometry parameters p_1 and p_2

as Gaussian distributed random variables. Let the mean value (for the starting point, $k = 0$) and the standard deviation been given by

$$\overline{p}_1^{(0)} = 9\,\mathrm{mm},\ \overline{p}_2^{(0)} = 5\,\mathrm{mm},\ \sigma_1 = \sigma_2 = 0.7. \tag{59}$$

The starting points for the deterministic variables are

$$d_1^{(0)} = d_2^{(0)} = 1. \tag{60}$$

The multidimensional optimization variable is defined by

$$\mathbf{x} = (\overline{p}_1, \overline{p}_2, d_1, d_2)^\top. \tag{61}$$

As quantity of interest we consider the scattering parameter S_r (S-parameter), which gives us information about the reflection behavior of the electromagnetic wave passing the waveguide. In order to calculate the value of the S-parameter for a specific setting, the electric field formulation of Maxwell has to be solved numerically, e.g. with the finite element method (FEM). The performance requirement is defined by

$$S_r(\mathbf{x}) \leq -24\,\mathrm{dB}\ \forall r \in T_r = [2\pi 6.5, 2\pi 7.5]\text{ in GHz}, \tag{62}$$

where the so-called range parameter r is the angular frequency. The range parameter interval T_r is discretized in eleven equidistant points and (62) has to be fulfilled for each of these points. The safe domain is the set of combinations of the uncertain parameters fulfilling the requirements, and depends on the current deterministic variable, i.e.,

$$\Omega \equiv \Omega_{d_1,d_2}(p_1, p_2) := \{(p_1, p_2) : S_r(\mathbf{x}) \leq -24\,\mathrm{dB}\ \forall r \in T_r\}. \tag{63}$$

We follow the definitions from Graeb (2007). The yield, i.e., the probability of fulfilling all requirements (62) under consideration of the uncertainties (59), is defined by

$$Y(\mathbf{x}) := \mathbb{E}\left[\mathbf{1}_\Omega(p_1, p_2)\right] = \int_{\mathbb{R}}\int_{\mathbb{R}} \mathbf{1}_\Omega(p_1, p_2)\mathrm{pdf}_{\overline{p}_1,\overline{p}_2,\sigma_1,\sigma_2}(p_1, p_2)\,\mathrm{d}p_1\,\mathrm{d}p_2, \tag{64}$$

where $\mathbf{1}_\Omega(p_1, p_2)$ defines the indicator function with value 1 if (p_1, p_2) lies inside the safe domain and 0 elsewise, and pdf defines the probability density function of the two dimensional Gaussian distribution. Equation (64) can be numerically estimated by a Monte Carlo analysis, i.e.,

$$Y_{\mathrm{MC}}(\mathbf{x}) = \frac{1}{N_{\mathrm{MC}}}\sum_{i=1}^{N_{\mathrm{MC}}} \mathbf{1}_\Omega\left(p_1^{(i)}, p_2^{(i)}\right), \tag{65}$$

where $(p_1^{(i)}, p_2^{(i)})_{i=1,\ldots,N_{\mathrm{MC}}}$ are sample points according to the distribution of the uncertain design parameters. Since for each sample point the S-parameter has to be calculated (using a time consuming simulation tool), the yield estimator is a computationally expensive function. In the next step, this function shall be optimized, i.e.,

$$\max_{\mathbf{x}} Y(\mathbf{x}). \tag{66}$$

Since $\overline{p}_1$ and $\overline{p}_2$ in (64) are only contained in the probability density function the derivative with respect to the mean values of the uncertain parameters is given by

$$\frac{\partial}{\partial \overline{p}_j} Y(\mathbf{x}) = \int_{\mathbb{R}} \int_{\mathbb{R}} \mathbf{1}_\Omega(p_1, p_2) \frac{\partial}{\partial \overline{p}_j} \mathrm{pdf}_{\overline{p}_1, \overline{p}_2, \sigma_1, \sigma_2}(p_1, p_2)\, \mathrm{d}p_1\, \mathrm{d}p_2, \;\; j = 1, 2. \tag{67}$$

And since the probability density function of the Gaussian distribution is an exponential function, it is continuously differentiable, thus the derivatives with respect to $\overline{p}_1$ and $\overline{p}_2$ can be calculated easily. Further, according to Graeb (2007), the derivative of the MC yield estimator with respect to the uncertain parameters is given by

$$\frac{\partial}{\partial \overline{p}_j} Y_{\mathrm{MC}}(\mathbf{x}) = Y_{\mathrm{MC}}(\mathbf{x}) \frac{1}{\sigma_j^2} (\overline{p}_{j,\Omega} - \overline{p}_j), \;\; j = 1, 2. \tag{68}$$

where $\overline{p}_{j,\Omega}$ is the mean value of all sample points of p_j lying inside the safe domain. This implies that there are not only closed-form expressions of these derivatives, but also numerical expressions which require only the evaluation of the objective function (which is anyway necessary), but no further computational effort. On the other hand, the deterministic variables are contained in the indicator function in (64). Thus, the corresponding partial derivatives are not considered as available. This leads to the situation that two partial derivatives are available, and two are unknown. The Hermite least squares approach described above can be applied and compared with standard BOBYQA and SQP.

There are two possibilities for the generation of the Monte Carlos sample set: a) the same sample set is used in each iteration and just shifted according to the current mean value (no noise) and b) the sample set is generated newly each time the mean value is changed (noise). Depending on the size of the sample set, the accuracy can be controlled. In the following we investigate three different settings:

1. no noise: same sample set (a) and $N_{\mathrm{MC}} = 2500$
2. low noise: new sample sets (b) and $N_{\mathrm{MC}} = 2500$
3. high noise: new sample sets (b) and $N_{\mathrm{MC}} = 100$

We start with the no noise setting and compare the different optimization methods with respect to the optimal value reached and the number of objective function calls needed. The initial yield value is $Y_{\mathrm{MC}}^{(0)} = 42.8\,\%$. The results are visualized in Fig. 6.

While the optimal yield values are similar (best for Hermite l.s. and SQP), the computational effort varies significantly. SQP performs best with only 36 objective function calls, Hermite l.s. is at the second position with 50 % more, then PyBOBYQA with 100 % more. This coincides with our findings in Sect. 5. There we could also observe that Hermite least squares performs best (excluding SQP).

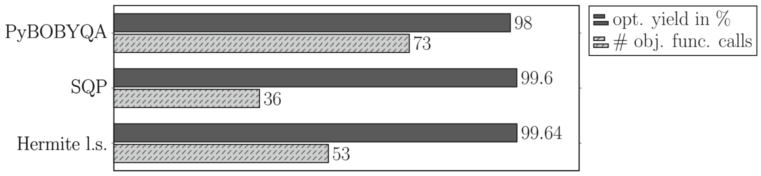
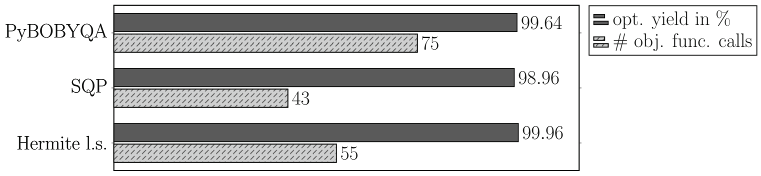
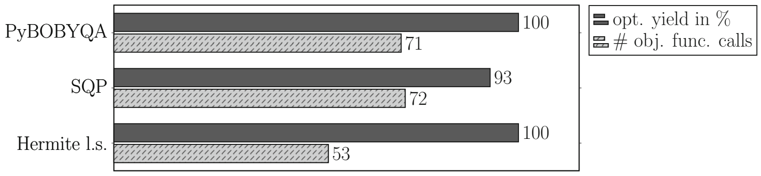


Fig. 6 Results for yield optimization in the no noise setting, solved with the reference PyBOBYQA method, the reference SQP method and Hermite least squares (Hermite l.s.)



(a) Low noise setting.



(b) High noise setting.

Fig. 7 Results for yield optimization for noisy settings, solved with the reference PyBOBYQA method, the reference SQP method and Hermite least squares (Hermite l.s.)

In the next step we evaluate the noisy settings. The results for the low noise setting are visualized in Fig. 7a, and for the high noise setting in Fig. 7b, respectively.

As in the no noise setting, PyBOBYQA and the Hermite least squares find the optimal solution and the number of objective function calls does not change significantly when noise is added. While in the low noise setting SQP finds a good optimal solution with the lowest number of objective function calls, in the high noise setting SQP loses its advantage in terms of computational effort, and at the same time breaks down in finding an optimum.

In summary, for this practical example, the Hermite least squares method performs best in terms of solution quality and computational effort. Further we observe that the interpolation and regression based methods can handle the noise, while SQP based on finite differences may not find the optimum anymore.

7 Conclusion

In this paper, we address the issue that in an optimization problem, some partial derivatives of the objective function are available and others are not. Based on Powell's derivative-free solver BOBYQA, we have developed two the Hermite least squares optimization method. Besides function evaluations, there we use the available first and second order derivatives of a training data set to build a quadratic approximation of the original objective function. In each iteration, this quadratic subproblem is solved in a trust region by least squares regression, and the training data set is updated.

Global convergence of the Hermite least squares method can be proven under the same assumptions as in Conn's BOBYQA version, i.e., for problems without bound constraints. In the Hermite least squares method, additionally a comparatively high number of interpolation points ($p_1 = q_1$) is required for the proof. However, in practice, decreasing the number of interpolation points leads to higher performance regarding the computational effort and thus to higher practical applicability. Numerical tests on 30 test problems including a practical example in the field of yield optimization have been performed. If half or more partial derivatives are available, the Hermite least squares approach outperforms (Py)BOBYQA in terms of computational effort by maintaining the ability of finding the optimal solution. Depending on the dimension and the amount of known derivative directions the number of objective function calls can be reduced by a factor up to five. Further, the proposed method is particularly stable with respect to noisy objective functions. In case of noisy data Hermite least squares finds the optimal solution more reliably and quickly than (Py)BOBYQA or gradient based solvers as sequential quadratic programming (SQP) using finite differences for calculating missing derivatives.

Acknowledgements This work has been supported by the European High-Performance Computing Joint Undertaking *TIME parallelisation: for eXascale computing and beyond* under the grant agreement No 955701, which receives support from the European Union's Horizon 2020 research and innovation programme and Belgium, France, Germany, Switzerland.

Funding Open Access funding enabled and organized by Projekt DEAL.

Appendix

Hermite least squares with second order derivatives

The usage of second order derivatives is straightforward. For the sake of completeness, we formulate the Hermite least squares system analog to (29–31). We denote the matrix with the second order derivative information corresponding to the i-th training data point by $\mathbf{M}_{2\mathrm{d}}^{i}$, the corresponding right hand side $\mathbf{b}_{2\mathrm{d}}^{i}$, respectively. They are given by

$$\mathbf{M}_{2\mathrm{d}}^{i} = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1}\phi_1(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_1}\phi_q(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) \\ \frac{\partial^2}{\partial x_1 \partial x_2}\phi_1(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_2}\phi_q(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) \\ \vdots & & \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_{n_{2\mathrm{kd}}}}\phi_1(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_{n_{2\mathrm{kd}}}}\phi_q(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) \\ \frac{\partial^2}{\partial x_2 \partial x_2}\phi_1(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) & \dots & \frac{\partial^2}{\partial x_2 \partial x_2}\phi_q(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) \\ \vdots & & \vdots \\ \frac{\partial^2}{\partial x_{n_{2\mathrm{kd}}} \partial x_{n_{2\mathrm{kd}}}}\phi_1(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) & \dots & \frac{\partial^2}{\partial x_{n_{2\mathrm{kd}}} \partial x_{n_{2\mathrm{kd}}}}\phi_q(\mathbf{y}^i - \mathbf{x}^{\mathrm{opt}}) \end{pmatrix} \tag{69}$$

with $\mathbf{M}_{2\mathrm{d}}^{i} \in \mathbb{R}^{(n_{2\mathrm{kd}}^2 + n_{2\mathrm{kd}})/2 \times q}$ and

$$\mathbf{b}_{2\mathrm{d}}^{i} = \begin{pmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f(\mathbf{y}^i) \\ \frac{\partial^2}{\partial x_1 \partial x_2} f(\mathbf{y}^i) \\ \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_{n_{2\mathrm{kd}}}} f(\mathbf{y}^i) \\ \frac{\partial^2}{\partial x_2 \partial x_2} f(\mathbf{y}^i) \\ \vdots \\ \frac{\partial^2}{\partial x_{n_{2\mathrm{kd}}} \partial x_{n_{2\mathrm{kd}}}} f(\mathbf{y}^i) \end{pmatrix} \in \mathbb{R}^{(n_{2\mathrm{kd}}^2 + n_{2\mathrm{kd}})/2}. \tag{70}$$

Utilizing that the basis Φ is defined by (10) and assuming the second order derivatives are available for all directions, i.e., $n_{2\mathrm{kd}} = n$, $\mathbf{M}_{2\mathrm{d}}^{i}$ can be simplified to

$$\mathbf{M}_{2\mathrm{d}}^{i,\mathrm{simpl.}} = \begin{pmatrix} 0 & \dots & 0 & 1 & \\ \vdots & & \vdots & & \ddots & \\ 0 & \dots & 0 & & & 1 \end{pmatrix}, \tag{71}$$

i.e., the linear part vanishes and the quadratic part is given by the identity matrix.

References

Audet C, Hare W (2017) Derivative-free and blackbox optimization. Springer. https://doi.org/10.1007/978-3-319-68913-5

Billups SC, Larson J, Graf P (2013) Derivative-free optimization of expensive functions with computational error using weighted regression. SIAM J Optim 23:27–53

Björck Å (1996) Numerical methods for least squares problems. SIAM

Cartis C, Fiala J, Marteau B, Roberts L (2019) Improving the flexibility and robustness of model-based derivative-free optimization solvers. ACM Trans Math Softw. https://doi.org/10.1145/3338517

Cartis C, Roberts L, Sheridan-Methven O (2021) Escaping local minima with local derivative-free methods: a numerical investigation. Optimization. https://doi.org/10.1080/02331934.2021.1883015

Conn AR, Scheinberg K, Vicente LN (2009) Introduction to derivative-free optimization. SIAM

Conn AR, Scheinberg K, Vicente LN (2008) Geometry of interpolation sets in derivative free optimization. Math Program 111:141–172

Conn AR, Scheinberg K, Vicente LN (2008) Geometry of sample sets in derivative-free optimization: polynomial regression and underdetermined interpolation. IMA J Numer Anal 28:721–748

Currin C, Mitchell T, Morris M, Ylvisaker D (1988) A Bayesian approach to the design and analysis of computer experiments. Tech Report ORNL–6498, Oak Ridge National Laboratory

Fuhrländer M, Schöps S (2022) Hermite least squares based on BOBYQA, Archived Version 1.0. Zenodo. https://doi.org/10.5281/zenodo.7473363

Fuhrländer M, Georg N, Römer U, Schöps S (2020) Yield optimization based on adaptive Newton-Monte Carlo and polynomial surrogates. Int J Uncert Quant 10:351–373. https://doi.org/10.1615/Int.J.UncertaintyQuantification.2020033344

Graeb HE (2007) Analog design centering and sizing. Springer, Dordrecht

Hermann M (2011) Numerische Mathematik. Oldenbourg Wissenschaftsverlag

Jones DR, Perttunen CD, Stuckman BE (1993) Lipschitzian optimization without the Lipschitz constant. J Optim Theory Appl 79:157–181

Kennedy J, Eberhart R (1995) Particle swarm optimization. Proc ICNN'95 Int Conf Neural Netw 4:1942–1948

Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Kraft D et al. (1988) A software package for sequential quadratic programming

Loukrezis D (2019) Benchmark models for uncertainty quantification, https://github.com/dlouk/UQ_benchmark_models/tree/master/rectangular_waveguides/debye1.py

Menhorn F, Augustin F, Bungartz HJ, Marzouk YM (2022) A trust-region method for derivative-free nonlinear constrained stochastic optimization, arXiv preprint arXiv:1703.04156

Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7:308–313

Powell MJD (1994) A direct search optimization method that models the objective and constraint functions by linear interpolation. Adv Optim Numer Anal, pp 51–67

Powell MJD (2002) UOBYQA: unconstrained optimization by quadratic approximation. Math Program 92:555–582

Powell MJD (2009) The BOBYQA algorithm for bound constrained optimization without derivatives, Cambridge NA Report NA2009/06, vol 26. University of Cambridge, Cambridge

Powell MJD (2015) On fast trust region methods for quadratic models with linear constraints. Math Program Comput 7:237–267

Rios LM, Sahinidis NV (2013) Derivative-free optimization: a review of algorithms and comparison of software implementations. J Global Optim 56:1247–1293

Sauer T, Xu Y (1995) On multivariate hermite interpolation. Adv Comput Math 4:207–259

Ulbrich M, Ulbrich S (2012) Nichtlineare Optimierung. Birkhäuser