MPC manuscript No.
(will be inserted by the editor)

Effective matrix adaptation strategy for noisy derivative-free optimization

Morteza Kimiaei · Arnold Neumaier

Received: date / Accepted: date

Abstract In this paper, we introduce a new effective matrix adaptation evolution strategy (`MADFO`) for noisy derivative-free optimization problems. Like every `MAES` solver, `MADFO` consists of three phases: mutation, selection and recombination. `MADFO` improves the mutation phase by generating good step sizes, neither too small not too large, that increase the probability of selecting mutation points with small inexact function values in the selection phase. In the recombination phase, a recombination point with lowest inexact function value found among all evaluated points so far may be found by a new randomized non-monotone line search method and accepted as the best point. If no best point is found, a heuristic point may be accepted as the best point. We compare `MADFO` with state-of-the-art DFO solvers on noisy test problems obtained by adding various kinds and levels of noise to all unconstrained `CUTEst` test problems with dimensions $n \leq 20$, and find that `MADFO` has the highest number of solved problems

Keywords Noisy derivative-free optimization · evolution strategy · heuristic optimization · stochastic optimization

Mathematics Subject Classification (2020) 90C15 · 90C30 · 90C56

M. Kimiaei
Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090, Wien, Austria
E-mail: kimiaeim83@univie.ac.at

A. Neumaier
Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090, Wien, Austria
E-mail: Arnold.Neumaier@univie.ac.at

1 Introduction

In this paper, we address the problem of minimizing the *noisy unconstrained derivative-free optimization* (DFO) problem

$$\begin{array}{l} \min f(x) \\ \text{s.t. } x \in \mathbb{R}^n, \end{array} \tag{1}$$

assuming that the smooth real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ is available only through a noisy oracle that takes $x \in \mathbb{R}^n$ and gives an approximated function value $\tilde{f}(x)$ of $f(x)$. The noise may be *deterministic* or *stochastic*. Sources of deterministic noise may be modelling, truncation, and/or discretization errors, and sources of stochastic noise may be rounding errors, simulation noise, or inaccurate measurements.

DFO algorithms do not assume any knowledge of the structure of the objective function, the true gradient or its Lipschitz constant, or the statistical properties of the noise. In particular, repeated evaluation of the function at the same point may or may not provide the same function value.

In finite precision arithmetic, the goal is to find an *ε-approximate stationary point* of the noisy unconstrained DFO problem, i.e., a point whose unknown exact gradient is below a given threshold $\varepsilon > 0$. In practice, we cannot know if such a point has been found since exact gradients are not available, and one usually stops when some heuristic test is satisfied. However, for problems, where the true gradient is available (though not used in the DFO algorithm), this can be checked aposteriori.

There are many solvers that can be used to solve the noisy unconstrained DFO problem (1), such as derivative-free line search solvers (cf. Larson et al. [36, Section 2.3.4]), derivative-free trust region solvers ([36, Section 2.4]), direct search solvers (see e.g., [36, Section 2.1]), matrix adaptation evolution strategies (see e.g., Auger and Hansen [9], Loshchilov et al. [39], Beyer [11], and Beyer and Sendhoff [12]). Two books with some historical references for DFO are Audet and Hare [8] and Conn et al. [15]. For the behaviour of these solvers, for the noiseless case see Rios and Sahinidis [48] and Kimiaei and Neumaier [32] and for the noisy case see Kimiaei [30]. Other useful references for noisy DFO are Berahas et al. [10], Chen [14], Elster and Neumaier [18], Gratton et al. [20,21], Gratton et al. [22], Huyer and Neumaier [27], Lucidi and Sciandrone [40], Moré and Wild [44], Powell [46,47], Shi et al. [49], and Wild et al. [52].

In the following, we only discuss line search methods and evolution strategies in more details since our method is an improved evolution strategy enriched by a new randomized non-monotone line search method and several new heuristic techniques.

1.1 Derivative-free line searches

Derivative-free line search solvers are among the fastest and most robust solvers for noisy DFO problems, e.g., the deterministic solver `SDBOX` by Lucidi and Sciandrone [40] and the randomized solvers `VRBBO` by Kimiaei and Neumaier [32] and `VRDFON` by Kimiaei [30]. For other methods, e.g., see Larson et al. [36, Section 2.3.4] and Shi et al. [49]. Derivative-free line search methods compute trial points and their inexact function values along a fixed direction and accept them if they satisfy a decrease condition. They can be classified into the following three classes:

In the first class, the gradients are approximated by finite differences and line search conditions are Armijo or Wolfe with approximate gradients and directional derivatives, e.g., `FMINUNC` by Matlab Optimization Toolbox [41] and `SSDFO` by Kimiaei et al. [34]. These solvers are robust and efficient for noiseless DFO problems. But, as shown in [30], `FMINUNC` is numerically very poor in the noisy case because the approximate gradient is not accurate, leading to poor quasi-Newton directions and line search failures.

In the second class, noisy variants of finite differences or Armijo or Wolfe line search and or both are used; e.g., see Moré and Wild [44], Berahas et al. [10], and Shi et al. [49]. The efficiency and robustness of derivative-free line search with the finite difference technique for the gradient approximation strongly depends on whether or not noise can be estimated in an efficient way. Applying noise estimation methods [49] to the L-BFGS algorithm [38] is useful. These methods require the knowledge of the noise level and n function evaluations for approximating the gradient in each iteration.

In the third class, neither the Armijo condition nor the Wolfe condition with approximate directional derivatives is used as a decrease condition to accept points with low inexact function values. Instead, approximate directional derivatives are replaced by forcing functions, non-decreasing and positive, like `SDBOX`, `VRBBO`, and `VRDFON`. However, `VRBBO` and `VRDFON` use gradient estimation to generate some heuristic techniques. As shown in [32,30], `SDBOX`, `VRBBO`, and `VRDFON` are robust line search solvers at low to high noise for noiseless and noisy unconstrained DFO problems in low to high dimensions and are efficient for noisy problems in medium to high dimensions.

Extrapolation is the main ingredient of the line search solvers. As long as the inexact function values are reduced, extrapolation increases the step sizes. This is particularly valuable when the slope of the function is small at the point with lowest inexact function value found so far, but no local minimizer is nearby. It helps the algorithm to quickly leave regions near a saddle point or local maximizer. By extrapolation, these line search solvers can actually find points with low inexact function values in the noiseless case, but in the noisy case they may need a randomized non-monotone line search condition. Indeed, the idea is to replace the inexact function value $f(x_{\rm old})$ at the old

point x_{old} with a randomized non-monotone term, which is at least $f(x_{\text{old}})$. Many deterministic non-monotone formulas have been proposed, which are useful when the function is flat or the valley is narrow, e.g., see Ahookhosh and Amini [1], Amini et al. [2,3], Birgin et al. [13], Diniz-Ehrhardt [16], Grippo and Rinaldi [24], Grippo et al. [23], Kimiaei [29], Kimiaei and Neumaier [33], Kimiaei and Rahpeymaii [35], Kimiaei et al. [31], Lucidi and Sciandrone [40], and Toint [50].

1.2 `MAES`, matrix adaptation evolution strategy

An *evolution strategy* is an algorithm that goes back to the principle of biological evolution (cf. [4]) and repeatedly proceeds with three phases:

- *Mutation* is a perturbation with zero mean.
- *Selection* selects some individuals to generate parents.
- *Recombination* chooses a new mean for the distribution.

Matrix adaptation evolution strategy (`MAES`) is a well-known numerical randomized DFO method used for solving noisy unconstrained DFO problems. Each mutation direction $d = Mz$ of `MAES` is chosen from a *multivariate normal distribution* $\mathcal{N}(0, C)$ with zero mean and covariance matrix $C := MM^T$, which specifies the pairwise dependencies between the variables in the distribution. Here the distribution direction z is chosen from $\mathcal{N}(0, I)$ with zero mean and variance I (an identity matrix) and M is an affine scaling matrix, which is determined adaptively and heuristically. `MAES` updates the shape of the distribution ellipsoid and is particularly useful when the objective function is ill-conditioned. There are different ways to update M or C, see e.g., Auger and Hansen [9] (the `CMAES` solver), Loshchilov et al. [39] (the `LMMAES` solver), Beyer [11] (the `FMAES` solver), and Beyer and Sendhoff [12] (the `BiPopMAES` solver).

Inspired by the approximation of the inverse Hessian matrix in quasi-Newton methods, a second order model of the objective function can be constructed whose Hessian is an adaptation of C or M. For convex-quadratic objective functions the inverse Hessian matrix is approximated by C, which is equivalent to rescaling the ellipsoid function to a spherical function (cf. Hansen [25]). Therefore, the optimal covariance matrix is assumed to be equal to the inverse Hessian matrix up to a constant factor. The final objective of covariance matrix is to approximate the contour lines of the objective function f as well as possible. For convex-quadratic functions, this amounts to an approximation of the inverse Hessian matrix, similar to a quasi-Newton method. No derivative is needed, only the ascending rank order of the inexact function values at the mutation points is used to learn the sample distribution. Thus, unlike

most traditional optimization methods, the nature of the underlying objective function requires fewer assumptions. Therefore, for problems with expensive function evaluations, `MAES` solvers are preferable to quasi-Newton methods since they require a lower number of function evaluations. However, these solvers do not care whether or not the inexact function value is reduced when step sizes are updated. Therefore, they may accept points with large inexact function values.

1.2.1 The basic mutation phase

The goal of the selection phase is to generate mutation points around the mean of the distribution.

At the ℓth iteration of `MAES`, three ingredients of the mutation phase are:

(i) Each component of distribution directions $z^{i\ell}$ $(i = 1, \ldots, \lambda)$ is chosen from the normal distribution $\mathcal{N}(0, 1)$ with zero mean and variance one. Here $\lambda \geq 2$ is the number of distribution directions.

(ii) The ith mutation direction

$$d^{i\ell} := M_\ell z^{i\ell} \tag{2}$$

is computed by multiplying the affine scaling matrix M_ℓ and the ith distribution direction $z^{i\ell}$.

(iii) The λ mutation points

$$x^{i\ell} := y^\ell + \sigma_\ell d^{i\ell} \quad \text{for } i = 1, 2, \ldots, \lambda \tag{3}$$

are computed, each of which is a perturbation of the current recombination y^ℓ. Here y^0 is an initial given point, $y^{\ell+1}$ is computed in the recombination phase below, $\sigma_\ell > 0$ is called *recombination step size* (initially given), and $M_\ell \in \mathbb{R}^{n\times n}$ is the ℓth affine scaling matrix, which is initially the identity matrix $M_0 = I$. We discuss below in the recombination phase how $\sigma_{\ell+1}$ and $M_{\ell+1}$ are updated.

1.2.2 The selection phase

The goal of the selection phase is to sort points and directions obtained from the mutation phase with respect to the ascending order of the inexact function values at the λ mutation points and then select some mutation points with low inexact function values and the corresponding directions, which are used by the recombination phase below.

At the ℓth iteration of `MAES`, the selection phase sorts the mutation points $x^{i\ell}$ $(i = 1, \ldots, \lambda)$ with

$$\tilde{f}(x^{\pi 1,\ell}) \leq \tilde{f}(x^{\pi 2,\ell}) \leq \ldots \leq \tilde{f}(x^{\pi \lambda,\ell}), \tag{4}$$

where π is a permutation of $\{1, 2, \ldots, \lambda\}$. The distribution directions $z^{i\ell}$ $(i = 1, \ldots, \lambda)$ and the mutation directions $d^{i\ell}$ $(i = 1, \ldots, \lambda)$ are sorted accordingly. Then the μ best mutation points and their inexact function values are selected:

$$X^\ell := (x^{\pi 1,\ell}, \ldots, x^{\pi \mu,\ell}) \in \mathbb{R}^{n \times \mu}, \quad \tilde{F}^\ell := (\tilde{f}^{\pi 1,\ell}, \ldots, \tilde{f}^{\pi \mu,\ell}) \in \mathbb{R}^{1 \times \mu}.$$

The corresponding distribution directions and mutation directions are:

$$Z^\ell := (z^{\pi 1,\ell}, \ldots, z^{\pi \mu,\ell}) \in \mathbb{R}^{n \times \mu}, \quad D^\ell := (d^{\pi 1,\ell}, \ldots, d^{\pi \mu,\ell}) \in \mathbb{R}^{n \times \mu}.$$

1.2.3 The basic recombination phase

Using the selected directions and points obtained from the selection phase, the recombination phase computes a new mean for the distribution (called recombination point below).

The main ingredients of the recombination phase are:

(i) The ℓth *recombination mutation* (*recm*) direction

$$d^\ell_{\text{rec}} := \sum_{i=1}^{\mu} w_i d^{\pi i,\ell}$$

and the ℓth *recombination distribution* (*recd*) direction

$$z^\ell_{\text{rec}} := \sum_{i=1}^{\mu} w_i z^{\pi i,\ell}$$

are computed, where the weights w_i of the recombination step satisfy

$$\sum_{i=1}^{\lambda} w_i = 1 \quad \text{and} \quad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0 = w_{\mu+1} = \ldots = w_\lambda. \tag{5}$$

Here $d^{\pi i,\ell}$ and $z^{\pi i,\ell}$, for $i = 1, \ldots, \mu$, are the μ mutation and distribution directions, respectively, sorted and selected in the selection phase. We discuss in Section 3 how to numerically compute the weights w_i for $i = 1, \ldots, \mu$.

(ii) The $(\ell + 1)$th recombination point

$$y^{\ell+1} := y^\ell + \sigma_\ell d^\ell_{\text{rec}}$$

and its inexact function value $\tilde{f}^{\ell+1} := \tilde{f}(y^{\ell+1})$ are computed, where the recombination step size σ_ℓ is updated below.

(iii) `MAES` updates adaptively the $(\ell+1)$th affine scaling matrix by the traditional formula (cf. Hansen [25])

$$M_{\ell+1} := M_\ell \left(I + \frac{c_1}{2}\Big(p_\sigma^\ell (p_\sigma^\ell)^T - I\Big) + \frac{c_\mu}{2}\Big(\sum_{i=1}^{\mu} w_i z^{\pi i,\ell}(z^{\pi i,\ell})^T - I\Big)\right) \tag{6}$$

with $\mathcal{O}(n^3)$ operations because of the matrix-matrix products (cf. Beyer [11]). Here $0 < c_\mu \leq 1$ is the learning rate for the rank-μ matrix of M_ℓ, $c_1 \leq 1 - c_\mu$ is the learning rate for the rank-one update of M_ℓ, and the evolution path

$$p_\sigma^0 := 0, \quad p_\sigma^{\ell+1} := (1-c_\sigma)p_\sigma^\ell + \overline{c}_\sigma z_{\text{rec}}^\ell \quad \text{with } \overline{c}_\sigma := \sqrt{c_\sigma(2-c_\sigma)\mu_{\text{rec}}}$$

is updated using the variance effective selection mass

$$\mu_{\text{rec}} := \frac{\|w\|_1^2}{\|w\|_2^2} = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \in [1, \mu]$$

by (5). Since

$$z^{\pi i,\ell}(z^{\pi i,\ell})^T = -z^{\pi i,\ell}(-z^{\pi i,\ell})^T,$$

the sign of selected distribution steps in the third term (6) is lost in the calculation of $M_{\ell+1}$. This is the motivation for using the evolution path p_σ^ℓ in the second term (6), which causes $M_{\ell+1}$ to behave better in practice when μ_{rec} is small (cf. [25]).

The formula (6) is a combination of the rank-one update

$$M_{\ell+1} = (1-c_1)M_\ell + c_1 p_\sigma^\ell (p_\sigma^\ell)^T \tag{7}$$

and the rank-μ update

$$M_{\ell+1} = (1-c_\mu)M_\ell + c_\mu \sum_{i=1}^{\mu} w_i z^{\pi i,\ell}(z^{\pi i,\ell})^T. \tag{8}$$

The weighted selection used in the third term of (6) and the second term of (8) lead to affine scaling matrices that behave better in practice. In (8), when $c_\mu = 0$, learning is not taken and when $c_\mu = 1$ prior information is not retained.

Maximum likelihood estimator (`MLE`) of the covariance matrix C differs slightly from the unbiased estimator; e.g., the empirical covariance matrix (mean of the actual realized sample) is an unbiased estimator of the original covariance matrix (the true mean value). `MLE` maximizes a likelihood function such that the observed data are most likely under the assumed statistical model.

In the rank-μ update (8), the second term is a result of maximizing a log-likelihood. Hence, the rank-one update for $M_{\ell+1}$ inserts the maximum likelihood of $z^{\pi i}$ $(i = 1, \ldots, \mu)$ into M_ℓ to increase the probability of $z^{\pi i}$ $(i = 1, \ldots, \mu)$ in the next iteration. The rank-μ update for $M_{\ell+1}$ is the mean of the estimated affine scaling matrices from all iterations, for more details see [25, Section 3].

(iv) Given the learning rate $c_\sigma \leq 1$ for the cumulation for the step size and the damping parameter $d_\sigma \approx 1$, the $(\ell+1)$th recombination step size

$$\sigma_{\ell+1} := \sigma_\ell \exp\Big(c_\sigma d_\sigma^{-1}\Big(e_\sigma^{-1}\|p_\sigma^\ell\| - 1\Big)\Big) \tag{9}$$

is updated, where e_σ is an approximation of the expected value $\mathbf{E}(\|u\|)$ of the norm of the vector $u \sim \mathcal{N}(0, I)$ (cf. [25, Section 4]).

1.2.4 `FMAES`, a fast `MAES` algorithm

We now discuss the details of the `FMAES` algorithm from Beyer [11]. `FMAES` is almost the same as `MAES`, which is a part of Loshchilov et al. [39, Algorithm 1], except that `FMAES` computes (9) using $e_\sigma = \mathcal{O}(\sqrt{n})$, while `MAES` computes it using $e_\sigma = n$.

Figure 1 is pseudocode for `FMAES`. Let us describe how `FMAES` works. The main loop of `FMAES` repeatedly performs three phases, `mutation-basic`, `selection`, and `recombination-basic`, until the maximum number of function evaluations is reached or an approximate stationary point is found.

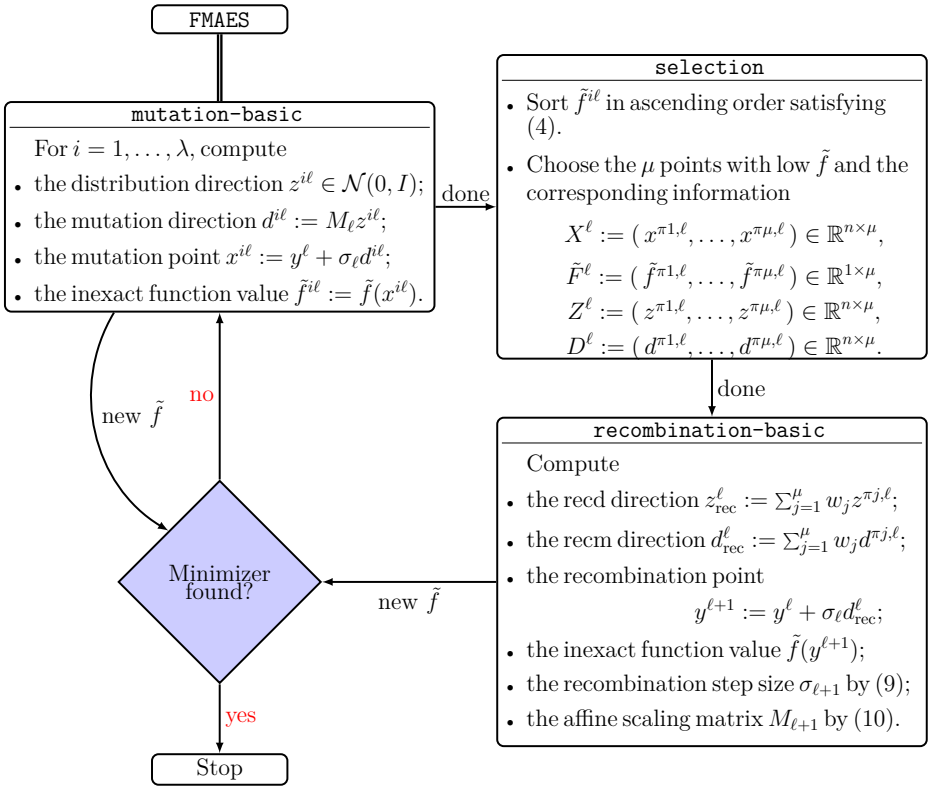

Fig. 1: Pseudocode for `FMAES`. $X_{:i}$ denotes the ith column of the matrix X.

`mutation-basic` computes distribution directions, mutation directions, mutation points, and their inexact function values. Then `selection` sorts these function values in ascending order and accordingly the distribution directions, mutation directions, and mutation points. Then finally it selects μ selected distribution directions, mutation directions, μ selected mutation points and their inexact function values. Using the selected information from the mutation phase, `recombination-basic` then computes two new recm and recd directions, a new recombination point and its inexact function value, and updates a new recombination step size and a new affine scaling matrix.

Defining $d_\sigma^\ell := M_\ell p_\sigma^\ell$, `FMAES` computes the $(\ell+1)$th affine scaling matrix

$$M_{\ell+1} := \left(1 - \frac{c_1}{2} - \frac{c_\mu}{2}\right) M_\ell + \frac{c_1}{2} d_\sigma^\ell (p_\sigma^\ell)^T + \frac{c_\mu}{2} \sum_{i=1}^{\mu} w_i d^{\pi i,\ell} (z^{\pi i,\ell})^T. \qquad (10)$$

Since d_σ^ℓ is precomputed and $d^{\pi i,\ell} = M_\ell z^{\pi i,\ell}$ is reused (computed in the mutation phase before), `FMAES` requires $\mathcal{O}(n^2)$ operations due to the vector-matrix products, unlike the traditional formula (6) that requires $\mathcal{O}(n^3)$ operations due to the matrix-matrix products.

Two sources for poor recm directions d_{rec}^ℓ and recd directions z_{rec}^ℓ by the basic recombination `recombination-basic` are:

(i) Since the previous recombination step size is used as the fixed step size (without any safeguard) to generate the current λ mutation points, these points are at the boundary of a neighborhood of the current recombination point and cannot be well-distributed. In this case, if the step size is large, many candidate points in this neighborhood that could be mutation points are ignored. If these mutation points are far from an approximate stationary point (this happens because there is no decrease condition for $\tilde{f}$ to accept points with low inexact function values), then the μ selected mutation points obtained from the selection phase are points possibly with large inexact function values; therefore, the recm direction and recd direction become poor directions. Therefore, this inadequacy of `mutation-basic` affects the ability of `recombination-basic` to find recombination points with low inexact function values.

(ii) For high noise, sorting the inexact function values at the mutation points may be not a successful process. Therefore, the ordering of the weights may not be correct, resulting in two poor recd and recm directions.

When noise is not large, the main shortcoming of `recombination-basic` is that there is no a descent condition, like the line search condition, to check whether or not a new recombination point with low inexact function value is found.

2 Our new algorithm

This section introduces MADFO and describes how it works.

2.1 The MADFO algorithm

This subsection describes briefly the MADFO algorithm and its main ingredients.

MADFO improves the two phases, mutation and recombination and preserves the selection phase. The main loop of MADFO repeatedly performs three phases, mutation, selection, and recombination, until the maximum number of function evaluations is reached or an approximate stationary point is found.

A recombination point with lowest inexact function value found so far by MADFO is called **best point**, denoted by y_{best}. Figure 2 is pseudocode for MADFO, showing a simple structure of MADFO and its functions. For $\ell = 1, 2, \ldots$, MADFO includes the following three functions:
• goodStepSize computes good mutation and recombination step sizes.
• recomSubDir computes recm subspace directions $d_{\text{rec}}^{\ell s}$.
• getBestPoint updates y_{best} and $\tilde{f}_{\text{best}} := \tilde{f}(y_{\text{best}})$.

We discuss goodStepSize in Subsection 2.3, recomSubDir in Subsection 2.4.1, and getBestPoint below.

The data structure $\tilde{F}_{\text{nm}}^{\ell}$ contains the list of at least λ inexact function values at the mutation points. It uses for the computation of non-monotone terms f_{nm}^{ℓ}. The vertices of the first triangle $\triangle_1 := \triangle(y_{\text{best}}, y_{\text{sbest}}, y_{\text{tbest}})$ are the three best points found so far by MADFO, called *best vertex* (y_{best}), *second best vertex* (y_{sbest}), and *third best vertex* (y_{tbest}), respectively. The best vertex of the second triangle $\triangle_2 := \triangle(y_{\text{best}}, y_{1,2}, y_{1,3})$ is the best vertex of $\triangle_1$. The other two vertices of $\triangle_2$ are $y_{1,2}$ (the mean of y_{best} and y_{sbest}) and $y_{1,3}$ (the mean of y_{best} and y_{tbest}).

getBestPoint uses the following functions:
• updateNM updates $\tilde{F}_{\text{nm}}^{\ell}$.
• randNM computes f_{nm}^{ℓ}.
• randomNMLS performs extrapolation along one of $\pm d_{\text{rec}}^{\ell s}$.
• triUpdate1 replaces in $\triangle_1$ the vertex with largest $\tilde{f}$ with y_{best}.
• triUpdate2 sorts the vertices of $\triangle_1$, so that their function values are in ascending order and then computes the vertices of $\triangle_2$.
• triSubPoint generates random triangle subspace points within $\triangle_2$.

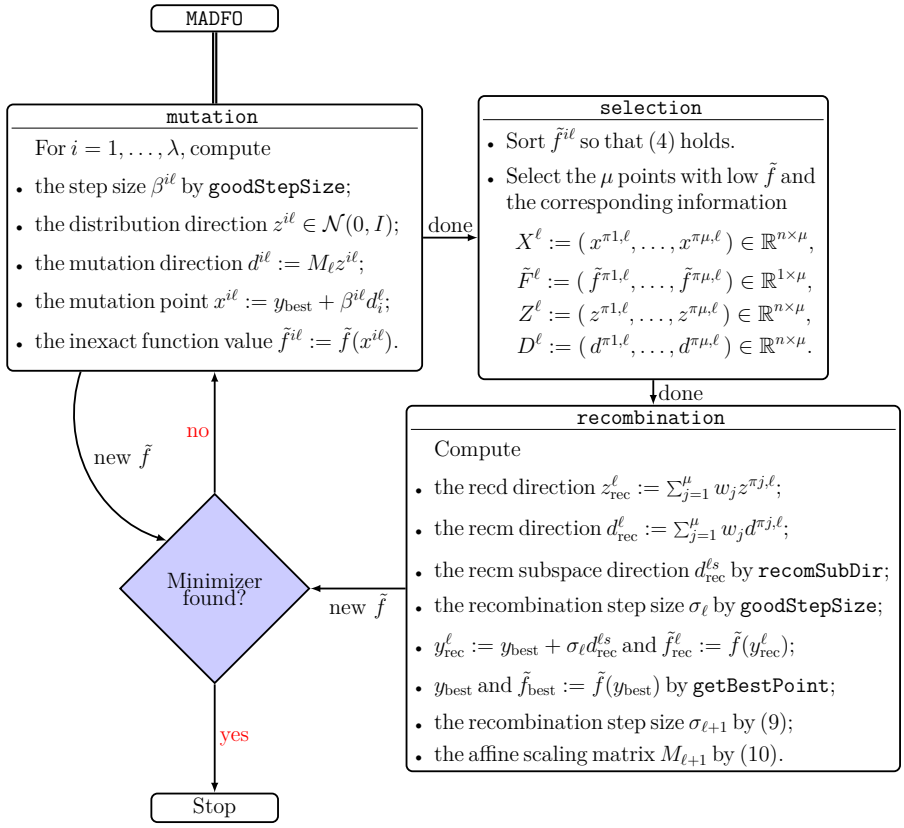

Fig. 2: Pseudocode for MADFO.

Figure 3 is pseudocode for getBestPoint, which tries to find the new $y_{\rm best}$ and its inexact function value $\tilde{f}_{\rm best}$, while calling the functions updateNM, randNM, randomNMLS (discussed in Subsection 2.4.3), triUpdate1, triUpdate2, and triSubPoint (discussed in Subsection 2.4.4).

getBestPoint uses the two decrease conditions on $\tilde{f}$ to accept the best point. The first condition, $\tilde{f}^\ell_{\rm nm} > \tilde{f}^\ell_{\rm rec} + \gamma\sigma^2_\ell$ with a given $0 < \gamma < 1$, is our new randomized line search condition (discussed in Subsection 2.4.3) and the second condition, $\tilde{f}^\ell_{\rm rec} < \tilde{f}_{\rm best}$, is used to check whether or not random triangle subspace points can be accepted as the new best point (see Subsection 2.4.4).

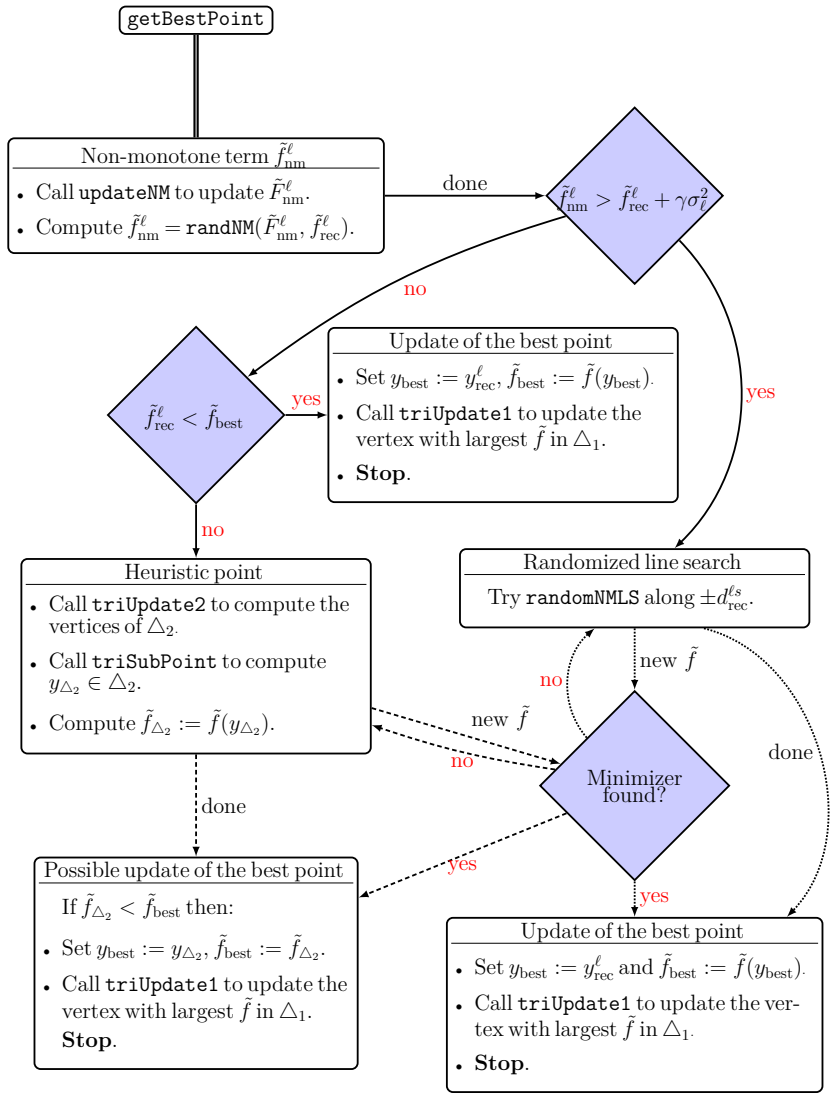

Fig. 3: Pseudocode for `getBestPoint`. Here $0 < \gamma < 1$ is a tuning parameter for line search.

2.2 An overview of our method

This paper describes the design and implementation of a robust solver, called *matrix adaptation evolution strategy* (`MADFO`) for noisy unconstrained DFO problems in low dimensions, an extended version of `FMAES` [11].

The main features of `MADFO` are:

(i) Unlike `MAES` solvers that use the previous recombination step size as the fixed mutation step sizes, the new good mutation step sizes are computed heuristically so that the mutation points are well-distributed; hence the chance of finding selected mutation points with low inexact function values increases in the selection phase, affecting the ordering of recombination weights and leading to finding the new best point in the recombination phase.

(ii) Extrapolation is performed using a new randomized non-monotone line search condition to enhance the chance of quickly leaving regions near a saddle point or local maximizer.

(iii) To avoid line search failures, recombination step sizes are controlled so that they are neither too small nor too large.

(iv) In each recombination phase, the recm subspace is spanned by the previous and current recm directions. Then, recm subspace directions within the recm subspace are chosen. Since the old recm directions are the result of the various sorting processes in the previous selection phases, there is a good chance that there are some safe sorting procedure in the cases where not much noise has accumulated yet. This property contributes to `MADFO` producing useful recm subspace directions even if the current sorting process fails due to noise.

(v) DFO algorithms use decrease conditions to find the best point. In the presence of noise, DFO algorithms may accept point whose function values are *spuriously good*, which means that the approximate function value is significantly smaller than the unknown exact function value. If only the best approximate point is kept, this may cause DFO algorithms to get stuck before finding an ε-approximate stationary point. If our line search cannot find the new best point, a heuristic procedure is used to find a random triangle subspace point that can be accepted as the new best point, a point whose inexact function value is in practice not spuriously good. After a few iterations, `MADFO` forms $\triangle_1$ and $\triangle_2$ while updating their vertices. This random triangle subspace point is inside $\triangle_2$ because $\triangle_2$ is near the best vertex of $\triangle_1$ and $\triangle_2$ and there is a high probability of finding the new best point in the neighborhood of the previous best point, leaving points with large inexact function values.

None of the above features can be found in previous `MAES` solvers. Although extrapolation has been used by other derivative-free line search solvers, such as `SDBOX`, `VRBBO`, and `VRDFON`, its use in a `MAES` solver is new.

In this subsection, we discuss how, as opposed to the basic mutation, the function `goodStepSize` heuristically computes in our improved mutation phase good mutation step sizes that are neither too small nor too large in the hope of finding mutation points with low inexact function values. The goal of `goodStepSize` is to generate good mutation step sizes based on the previous best point, so that the components of mutation points never become too small or too large, and mutation points are well-distributed around the previous best point, useful in both noisy and noiseless cases.

Let us discuss our motivation to generate good mutation step sizes. For fixed mutation step sizes that match the previous recombination step size, all λ mutation points evaluated are likely to be within a ball whose center is the best point and whose radius is either too large or too small if the fixed step size is too large or too small. If this fixed step size is too large and there is an approximate stationary point near the current recombination point, mutation points are far from an approximate stationary point. On the other hand, if this fixed step size is too small and an approximate stationary point is far from the current recombination point, mutation points are far from a local minimizer. In both cases, the new recombination point is generated far from an approximate stationary point, resulting in slow convergence or failure of `MAES` solvers to find an approximate stationary point.

We now discuss the effects of good mutation step sizes on the selection and recombination phases. Due to the good mutation step sizes computed by `goodStepSize` in the mutation phase, λ mutation points are in a neighborhood of the current best point whose radius is equal to the maximum value of the good step sizes generated in the mutation phase. In such a case, this neighborhood contains some well-distributed points with low or large inexact function values that can be selected as mutation points. Although some mutation points may have large inexact function values, μ mutation points are selected in the selection phase, most of which may have low inexact function values. By using sorted weights and μ selected mutation and distribution steps, more useful recm directions and recd directions can be generated. As a consequence, the new best point can more often be found and the new affine scaling matrix behaves well in practice. Therefore in the next iteration mutation directions with this matrix will be more useful directions.

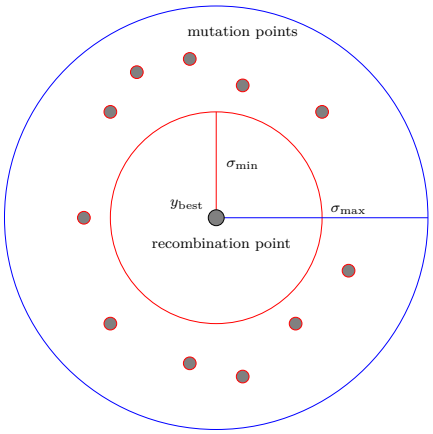

Fig. 4: Well-distributed mutation points.

Given the tuning parameters $0 < \sigma_{\min} < \sigma_{\max} < \infty$, to calculate the well-distributed mutation points

$$x^{j\ell} = y_{\text{best}} + \beta^{j\ell} d^{j\ell} \quad \text{for } j = 1, \ldots, \lambda$$

within a neighborhood of the best point y_{best}, `goodStepSize` calculates the good mutation step sizes $\beta^{j\ell}$, for $j = 1, \ldots, \lambda$, so that

$$\sigma_{\min} < \beta^{j\ell} < \sigma_{\max} \quad \text{for } j = 1, \ldots, \lambda.$$

This restriction is independent of the noise level, but is recommended for noisy problems and problems with highly nonconvex objective functions because it avoids too small step sizes (the source of null steps) and too large step sizes (a source of line search failure); cf. Figure 4. Hence, each component of the mutation points $x^{j\ell}$ for $j = 1, \ldots, \lambda$ is constrained to be neither too small nor too large, and these points are well-distributed around the best point y_{best}. This is the reason why y_{best} is included in the calculation of each mutation step size.

At the first iteration, $\beta^{j\ell} := \sigma_0$ is chosen and continuing until $y_{\text{best}} = 0$. This case happens if the initial point is 0 and the algorithm cannot update y_{best}. `goodStepSize` performs the following steps:
(S1) Compute the vector $\mathbf{a}^{j\ell} = |y_{\text{best}}|//|d^{j\ell}|$, where // denotes componentwise division.
(S2) Eliminate from the vector $\mathbf{a}^{j\ell}$ the components that are zero, NaN, infinity, or outside of $(\sigma_{\min}, \sigma_{\max})$ if any.
(S3) If $\mathbf{a}^{j\ell}$ is empty, set $\beta^{j\ell} = \sigma_{\ell-1}$. Otherwise, project $\sigma_{\ell-1}$ into $[\sigma_{\min}, \sigma_{\max}]$ and then compute

$$\mathbf{a}^{j\ell}_{\text{med}} := \operatorname*{median}_i \{\mathbf{a}^{j\ell}_i\} \quad \text{and} \quad \beta^{j\ell} := \sqrt{\sigma_{\ell-1} \mathbf{a}^{j\ell}_{\text{med}}}. \tag{11}$$

We found that, in practice, the proper behavior under scaling is more important than translation invariance. The recipe used for the $\beta^{j\ell}$ ensures that the magnitudes of the components of the corrections match those of the current best point.

One of the causes of line search failure is too small or too large step sizes. If the vector $\mathbf{a}^{j\ell}$ has components larger than $\sigma_{\max}$, they are removed to avoid large steps, and if the vector $\mathbf{a}^{j\ell}$ has components smaller than $\sigma_{\min}$, they are removed to avoid small steps. The more robust medians $\mathbf{a}^{j\ell}_{\text{med}}$ for $j = 1, \ldots, \lambda$ are obtained by restricting $\mathbf{a}^{j\ell}$ to $(\sigma_{\min}, \sigma_{\max})$. Hence $\beta^{j\ell}$ are forced to lie $(\sigma_{\min}, \sigma_{\max})$ and their corresponding mutation points are well-distributed around the previous best point.

2.4 Improved recombination

This subsection discusses new heuristic techniques that improve the recombination phase to update the best point.

2.4.1 `recomSubDir` *– recm subspace direction*

This subsection describes how `recomSubDir` chooses the recm subspace directions inside the recm subspace, spanned by the old and current recm directions.

The purpose of `recomSubDir` is to prevent `MADFO` from getting stuck in cases where the sorting process fails in the selection phase when noise is large. This goal can be achieved by using the old and current recm directions. Both the old and current recm directions are scaled by two different random factors to determine what percentage of these directions can be used to construct the recm subspace direction. This is necessary to construct the recm subspace direction based on the different sorting procedures of the current and previous selection phases, because if the current sorting procedure fails, the recombination weights may be wrong, so both recm and recd directions and a new recombination point may not be useful.

`recomSubDir` computes the ℓth recm subspace direction

$$d^{\ell s}_{\text{rec}} := \begin{cases} s \circ \left(\theta d^{\ell}_{\text{rec}} + q^{-\ell}\sqrt{1-\theta^2}\, d^{\ell-1}_{\text{rec}}\right) & \text{if } \theta \geq 0.5 \\ s \circ \left(\sqrt{1-\theta^2}\, d^{\ell}_{\text{rec}} + \theta q^{-\ell} d^{\ell-1}_{\text{rec}}\right) & \text{otherwise,} \end{cases} \tag{12}$$

where $\circ$ denotes the componentwise product, $1 < q < \infty$ is a tuning parameter, $\theta \in (0, 1)$ is a uniformly distributed random value, and $s \in \mathbb{R}^n$ is a scaling vector:

• Optimization problems frequently have variables of different magnitudes. These are estimated by the scaling vector

$$s := \sup\left\{y_{\text{best}} - y_{\text{sbest}}, y_{\text{best}} - y_{\text{tbest}}\right\},$$

where the two vectors $y_{\text{best}} - y_{\text{sbest}}$ and $y_{\text{best}} - y_{\text{tbest}}$ give for each component sensible componentwise step lengths in the direction of the current valley, unless they vanish; in that case the default value of one (i.e., $s_i = 1$ for $i \in J = \{i \mid s_i = 0\}$) is used. Here y_{best}, y_{sbest}, and y_{tbest} are the best point, second best point, and third best point, respectively, which are the vertices of $\triangle_1$ and are sorted so that their function values are in ascending order. Until these three best points are found, $s_i = 1$ for $i = 1, \ldots, n$.

• We recommend no restart process to compute d^{ℓ}_{rec} because a restart process eliminates the old good information that was saved in cases where noise was not significant.

• When the iterations are near an ε-approximate stationary point, $q^{-\ell}$ may become too small. In this case, the recm subspace direction mainly uses the first component d^{ℓ}_{rec} of the recm subspace, since the use of small steps is preferable to the use of large steps, and the use of both terms may not be beneficial in the presence of much accumulated noise.

• The contribution of d^{ℓ}_{rec} is preferably larger than that of $d^{\ell-1}_{\text{rec}}$ at the ℓth iteration; therefore, for $\theta \geq 0.5$, the factor of d^{ℓ}_{rec} is θ and otherwise $\sqrt{1-\theta^2}$.

2.4.2 Updating and adjusting recombination step sizes

This subsection discusses how `goodStepSize` updates and possibly adjusts the recombination step size in each iteration. The first goal of `goodStepSize` was discussed in Section 2.3.

The second goal of `goodStepSize` is to calculate the recombination step size, so that it is neither too small nor too large. Therefore, large steps and null steps, which are sources of line search failure, never occur. To achieve this goal, `goodStepSize` computes the recombination step size by a heuristic formula using the components of the old best point so that the components of the new recombination point are neither too small nor too large. As explained in Section 2.3, this constraint on the recombination step size results in good mutation step sizes that are neither too large nor too small, leading to a good distribution of mutation points around the current best point.

Since in the recombination phase the recm direction $d^{\ell s}_{\text{rec}}$ is used, `goodStepSize` performs the following steps:
(S1) Compute the vector $\mathbf{c}^{\ell} = |y_{\text{best}}|//|d^{\ell s}_{\text{rec}}|$, where // denotes componentwise

division.
(S2) Eliminate from the vector $\mathbf{c}^\ell$ the components that are zero, NaN, infinity, or outside of $(\sigma_{\min}, \sigma_{\max})$ if any.
(S3) If $\mathbf{c}^\ell$ is empty, set $\sigma_\ell = \sigma_{\ell-1}$. Otherwise, project $\sigma_{\ell-1}$ into $[\sigma_{\min}, \sigma_{\max}]$ and then compute

$$\mathbf{c}^\ell_{\text{med}} := \operatorname*{median}_i\{\mathbf{c}^\ell_i\} \quad \text{and} \quad \sigma_\ell := \sqrt{\sigma_{\ell-1}\mathbf{c}^\ell_{\text{med}}}.$$

In practice, we found that translation invariance is not as important as appropriate scaling behavior. The recipe used for the $\mathbf{c}^\ell$ ensures that the magnitudes of the components of the corrections match those of the current best point. At the end of each iteration, `MADFO` computes σ_ℓ by the traditional formula (9). This step size is used as the initial extrapolation step size. Therefore, this choice affects the ability of extrapolation; because if the initial step size is too large, extrapolation cannot be done, and if it is too small, extrapolation can be expensive.

The reason why y_{best} is used to calculate $\mathbf{c}^\ell$ is that the components of the best point y_{best} become neither too small nor too large. On the other hand, the reason why the median was taken to calculate $\mathbf{c}^\ell_{\text{med}}$ is that there is no guarantee that the data used are symmetrically distributed. Since large and small components of $\mathbf{c}^\ell$ were removed, the more robust $\mathbf{c}^\ell_{\text{med}}$ is obtained.

2.4.3 `randomNMLS` *– randomized non-monotone line search*

This subsection describes how to compute our new randomized non-monotone term and insert it into a line search condition whose goal is in practice to discard recombination points whose function values are spuriously good and perform extrapolation to quickly leave regions near a saddle point or local maximizer.

Figure 5 plots several heuristic approximations to the exact function values $f^\ell := f(x^\ell)$. It shows that the approximation $\tilde{f}^\ell_{\text{nm}}$ to f^ℓ tends to be better than the three noisy function values, namely the minimum $\tilde{f}^\ell_{\min}$, the maximum $\tilde{f}^\ell_{\max}$, and the median $\tilde{f}^\ell_{\text{med}}$ of the list

$$\tilde{F}^\ell_{\text{nm}} = \big(\, \tilde{f}(x^{1\ell}), \ldots, \tilde{f}(x^{\overline{\lambda}\ell}) \,\big)$$

of $\overline{\lambda}$ noisy function values $\tilde{f}(x^{i\ell})$ $(i = 1, \ldots, \lambda)$ at the current and previous mutation points $x^{i\ell}$ (here $\overline{\lambda} = \kappa\lambda$ is a multiple of λ, where $\kappa \geq 1$ is an integer tuning parameter). This goal can be achieved by defining $\tilde{f}^\ell_{\text{nm}}$ as a convex combination of the two pairs

$$(\tilde{f}^\ell_{\text{med}}, \tilde{f}^\ell_{\max}) \quad \text{or} \quad (\tilde{f}^\ell_{\min}, \tilde{f}^\ell_{\text{med}}). \tag{13}$$

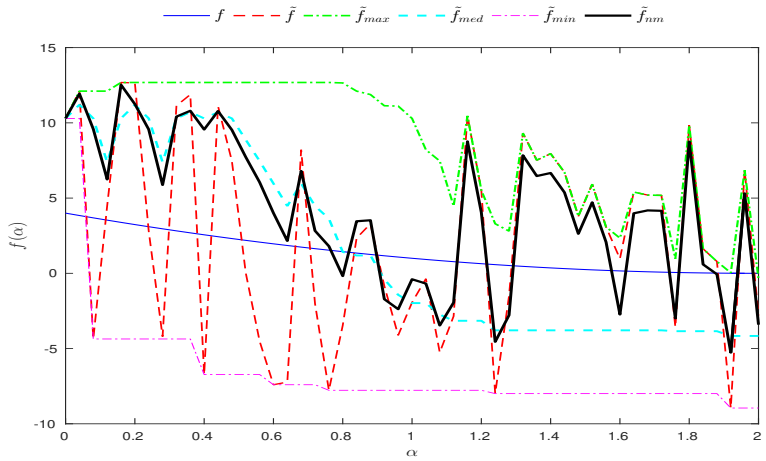

Fig. 5: A plot of the exact function values f, the noisy function values $\tilde{f}$, the values $\tilde{f}_{\max}$, $\tilde{f}_{\min}$, and $\tilde{f}_{\text{med}}$ from (14), and the new noisy non-monotone term values $\tilde{f}_{\text{nm}}$ of the one-dimensional objective function $f(\alpha) = (\alpha - 2)^2$ for $\alpha \in [0, 2]$. Absolute uniform noise with noise level 1 was used.

Since the data used to calculate the non-monotone term does not need to be symmetrically distributed, the more robust median $\tilde{f}^{\ell}_{\text{med}}$ is used rather than the mean $\tilde{f}^{\ell}_{\text{mean}}$. Using the three values $\tilde{f}^{\ell}_{\min}$, $\tilde{f}^{\ell}_{\max}$, and $\tilde{f}^{\ell}_{\text{med}}$, the parameter $\tilde{\eta}$ of this convex combination is heuristically calculated to lie in $]\frac{1}{2}, 1[$, while identifying that $\tilde{f}^{\ell}_{\text{nm}}$ becomes close to $\tilde{f}^{\ell}_{\text{med}}$ or $\tilde{f}^{\ell}_{\min}$.

To get the more robust median $\tilde{f}^{\ell}_{\text{med}}$, $\tilde{F}^{\ell}_{\text{nm}}$ must have more inexact function values at the current and previous mutation points. If the length of $\tilde{F}^{\ell}_{\text{nm}}$ is below $\overline{\lambda}$, `updateNM` adds the λ inexact function values at the current mutation points to $\tilde{F}^{\ell}_{\text{nm}}$. Otherwise, `updateNM` selects a random subset J of $\{1, 2, \ldots, \overline{\lambda}\}$ with the λ members and then only replaces randomly the λ members of the list $\tilde{F}^{\ell}_{\text{nm}}$ by the λ inexact function values $\tilde{f}(x^{j\ell})$ $(j = 1, 2, \ldots, \lambda)$ at the mutation points $x^{j\ell}$, i.e.,

$$(\tilde{F}^{\ell}_{\text{nm}})_{J(j)} = \tilde{f}(x^{j\ell}) \quad \text{for } j = 1, 2, \ldots, \lambda.$$

`randNM` takes as input the list $\tilde{F}^{\ell}_{\text{nm}}$ of inexact function values at the $\overline{\lambda}$ mutation points, the inexact function value $\tilde{f}_{\text{best}} := \tilde{f}(y_{\text{best}})$ at the best point y_{best}, and the inexact function value $\tilde{f}^{\ell}_{\text{rec}} = \tilde{f}(y^{\ell}_{\text{rec}})$ at the recombination point

$$y^{\ell}_{\text{rec}} := y_{\text{best}} + \sigma_{\ell} d^{\ell s}_{\text{rec}}.$$

It returns the ℓth non-monotone term $\tilde{f}^{\ell}_{\text{nm}}$.

`randNM` uses the following steps:
(S1) Compute three values

$$\tilde{f}^\ell_{\min} := \min\big(\tilde{f}_{\rm best}, \min(\tilde{F}^\ell_{\rm nm})\big), \quad \tilde{f}^\ell_{\max} := \max\big(\tilde{F}^\ell_{\rm nm}\big), \quad \tilde{f}^\ell_{\rm med} := \operatorname{median}\big(\tilde{F}^\ell_{\rm nm}\big). \tag{14}$$

(S2) Compute two adaptive parameters

$$\eta^\ell_1 := (\tilde{f}^\ell_{\rm med} - \tilde{f}^\ell_{\min})/(\tilde{f}^\ell_{\max} - \tilde{f}^\ell_{\min}), \quad \eta^\ell_2 := (\tilde{f}^\ell_{\max} - \tilde{f}^\ell_{\rm med})/(\tilde{f}^\ell_{\max} - \tilde{f}^\ell_{\min}).$$

(S3) Using η^ℓ_1 and η^ℓ_2, compute heuristically the parameter

$$\eta^\ell := \begin{cases} \min(\eta^\ell_1, \eta^\ell_2) & \text{if } \eta^\ell_1 \neq 0 \text{ and } \eta^\ell_2 \neq 0, \\ \eta^\ell_1 & \text{if } \eta^\ell_1 \text{ is nonzero,} \\ \eta^\ell_2 & \text{if } \eta^\ell_2 \text{ is nonzero,} \\ r & \text{otherwise} \end{cases}$$

and restrict it to $\tilde{\eta}^\ell = \max(r, \eta^\ell) \in \,]\frac{1}{2}, 1[$, where $r \in \,]\frac{1}{2}, 1[$ is a uniformly distributed random value.
(S4) Compute the randomized non-monotone term

$$\tilde{f}^\ell_{\rm nm} := \begin{cases} \tilde{\eta}^\ell \tilde{f}^\ell_{\rm med} + (1-\tilde{\eta}^\ell)\tilde{f}^\ell_{\max} & \text{if } \tilde{f}^\ell_{\rm rec} \geq \tilde{f}^\ell_{\max}, \\ (1-\tilde{\eta}^\ell)\tilde{f}^\ell_{\rm med} + \tilde{\eta}^\ell \tilde{f}^\ell_{\min} & \text{otherwise.} \end{cases} \tag{15}$$

The parameter $\tilde{\eta}^\ell$ heuristically computed in (S3) can be computed by any heuristic formula, provided it belongs to $]\frac{1}{2}, 1[$. This parameter controls the non-monotone term $\tilde{f}^\ell_{\rm nm}$ to be close to $\tilde{f}^\ell_{\rm med}$ or $\tilde{f}^\ell_{\min}$ according to whether or not $\tilde{f}^\ell_{\rm rec} \geq f^\ell_{\max}$ holds:

• If $\tilde{f}^\ell_{\rm rec} \geq \tilde{f}^\ell_{\max}$, $\tilde{f}^\ell_{\rm nm}$ is chosen to be in $(\tilde{f}^\ell_{\rm med}, \tilde{f}^\ell_{\max})$, closer to $\tilde{f}^\ell_{\rm med}$ than to $\tilde{f}^\ell_{\max}$, since the weight $\tilde{\eta}^\ell \in \,]\frac{1}{2}, 1[$ of $\tilde{f}^\ell_{\rm med}$ is greater than the weight $1 - \tilde{\eta}^\ell$ of $\tilde{f}^\ell_{\max}$.

• The condition $\tilde{f}^\ell_{\rm rec} \geq \tilde{f}^\ell_{\max}$ tells us that either the recombination point is far from an approximate stationary point or the noise is strong; hence there is a possibility to find points whose inexact function values (in practice not spuriously good) belong to $(\tilde{f}^\ell_{\rm med}, \tilde{f}^\ell_{\max})$.

• If $\tilde{f}^\ell_{\rm rec} < \tilde{f}^\ell_{\max}$ holds, $\tilde{f}^\ell_{\rm nm}$ is chosen to be within $(\tilde{f}^\ell_{\min}, \tilde{f}^\ell_{\rm med})$, closer to $\tilde{f}^\ell_{\min}$ than to $\tilde{f}^\ell_{\rm med}$, since the weight $\tilde{\eta}^\ell \in \,]\frac{1}{2}, 1[$ of $\tilde{f}^\ell_{\min}$ is greater than the weight $1 - \tilde{\eta}^\ell$ of $\tilde{f}^\ell_{\rm med}$.

• The condition $\tilde{f}^\ell_{\rm rec} < \tilde{f}^\ell_{\max}$ tells us that the recombination point is close to the current best point; so the inexact function value (in practice not spuriously good) at this point belong to $(\tilde{f}^\ell_{\rm med}, \tilde{f}^\ell_{\max})$.

In the presence of strong noise or highly nonconvex functions, if extrapolation uses the monotone line search condition

$$\tilde{f}_{\text{best}} > \tilde{f}(y_{\text{best}} + \sigma_\ell d_{\text{rec}}^{\ell s}) + \gamma\sigma_\ell^2 \quad \text{with } 0 < \gamma < 1, \tag{16}$$

this condition has a small chance of finding a point whose inexact function value is not spuriously good, since

$$\tilde{f}_{\text{nm}}^{\ell} \geq \tilde{f}_{\text{min}}^{\ell} \geq \tilde{f}_{\text{best}}. \tag{17}$$

In (17), the first inequality is obtained from (15) and the second inequality is obtained from the definition of $\tilde{f}_{\text{min}}^{\ell}$ and selecting a random subset J of $\{1, 2. \ldots, \overline{\lambda}\}$. The condition (16) contains no directional derivative unlike the first and second classes of line search methods discussed in Section 1.1; instead, the forcing function $\gamma\sigma_\ell^2$ is used. As a useful substitute for $\tilde{f}_{\text{best}}$ in (16), we use the non-monotone term $\tilde{f}_{\text{nm}}^{\ell}$ computed by (15) in the randomized non-monotone line search condition

$$\tilde{f}_{\text{nm}}^{\ell} > \tilde{f}(y_{\text{best}} + \sigma_\ell d_{\text{rec}}^{\ell s}) + \gamma\sigma_\ell^2. \tag{18}$$

`MADFO` calls `randomNMLS` checking whether or not the line search condition (18) holds. If this condition holds, it performs extrapolation along one of $\pm d_{\text{rec}}^{\ell s}$ with the goal of quickly leaving regions near a saddle point or local maximizer in cases where the slope of the function at the current best point is small, finding recombination points whose function value is not spuriously good, and accepting one of these points with lowest inexact function value found so far as the new best point.

Using the tuning parameters $0 < \gamma < 1$ (parameter for the line search condition) and $\gamma_e > 1$ (parameter for expanding the step size), at the ℓth iteration of `MADFO`, `randomNMLS` takes as input the ℓth recombination step size σ_ℓ, the inexact function value $\tilde{f}_{\text{rec}}^{\ell} := \tilde{f}(y_{\text{rec}}^{\ell})$ at the ℓth recombination point $y_{\text{rec}}^{\ell} := y_{\text{best}} + \sigma_\ell d_{\text{rec}}^{\ell s}$, the current best point y_{best}, and its inexact function value $\tilde{f}_{\text{best}}$ and returns as output the new best point y_{best} and its inexact function value $\tilde{f}_{\text{best}}$. It includes the following steps:
(R1) If the non-monotone condition (18) holds, go to (R3) to perform extrapolation along $d_{\text{rec}}^{\ell s}$.
(R2) If there is no reduction in $\tilde{f}$, the opposite direction $d_{\text{rec}}^{\ell s} = -d_{\text{rec}}^{\ell s}$ is chosen and go to (R3) to perform extrapolation along $d_{\text{rec}}^{\ell s}$.
(R3) Initialize the two values $\tilde{f}_b := \tilde{f}_{\text{rec}}^{\ell}$ and $\sigma_b := \sigma_\ell$.
(R4) Expand the step size $\sigma_\ell := \gamma_e \sigma_\ell$.
(R5) Compute the recombination point $y_{\text{rec}} := y_{\text{best}} + \sigma_\ell d_{\text{rec}}^{\ell s}$ and its inexact function value $\tilde{f}_{\text{rec}} := \tilde{f}(y_{\text{rec}})$. Then, if $\tilde{f}_{\text{rec}} < \tilde{f}_b$, set $\tilde{f}_b := \tilde{f}_{\text{rec}}$ and $\sigma_b := \sigma_\ell$.
(R6) Once the non-monotone condition $\tilde{f}_{\text{nm}}^{\ell} > \tilde{f}_{\text{rec}} + \gamma\sigma_\ell^2$ is violated, set

$y_{\text{best}} := y_{\text{best}} + \sigma_b d_{\text{rec}}^{\ell s}$ and $\tilde{f}_{\text{best}} := \tilde{f}_b$ and terminate the extrapolation (indeed, during the extrapolation, many recombination points with low inexact function values may be found, hence a point with the lowest inexact function value is accepted as the new best point). Otherwise, go back (R4).

As stated in (R6), extrapolation is terminated when the non-monotone line search condition (18) does not hold. But when the objective function is unbounded below, extrapolation is terminated when the function value reaches -10^{12}.

2.4.4 `triSubPoint` *– random triangle subspace point*

This subsection discusses how `triSubPoint` finds random triangle subspace points, which may be accepted as the new best point.

The goal of `triSubPoint` is to find the new best point when `randomNMLS` cannot find such a best point due to noise. This is in contrast to any `MAES` solver, which accepts any evaluated recombination point, not necessarily the new best point.

To achieve this goal, `triUpdate1` forms $\triangle_1 := \triangle(y_{\text{best}}, y_{\text{sbest}}, y_{\text{tbest}})$ and updates its vertices y_{best}, y_{sbest}, and y_{tbest}, while `triUpdate2` forms $\triangle_2 := \triangle(y_{\text{best}}, y_{1,2}, y_{1,3})$ and updates its vertices y_{best}, $y_{1,2}$ (the mean of y_{best} and y_{sbest}), and $y_{1,3}$ (the mean of y_{best} and y_{tbest}); cf. Figure 6. These three vertices are the three best points found from a list of $N \geq 3$ previous best points found by `MADFO` so far, because some such points may be found by our non-monotone line search method. Then, `triSubPoint` selects one *random triangle subspace point* within $\triangle_2$ which is closest neighborhood to the best vertex y_{best}. If the inexact function value at this point is decreased, this point is accepted as the new best point.

The goal of `triSubPoint` is to compute random triangle subspace points within $\triangle_2$ by the following four steps:
(S1) Form the matrix $\overline{X} := (\, y_{\text{best}} \quad y_{1,2} \quad y_{1,3} \,)$.
(S2) Choose $v := (\, v_1 \quad v_2 \quad v_3 \,) \in \mathcal{N}(0, I)$.
(S3) Scale the random vector $v := v/\|v\|_\infty$.
(S4) Compute the random triangle subspace point $y_{\triangle_2} := \sum_{i=1}^{3} v_i \overline{X}_{:i}$.

If $y_{\triangle_2}$ satisfies the condition $\tilde{f}(y_{\triangle_2}) < \tilde{f}_{\text{best}}$, then $y_{\text{best}} := y_{\triangle_2}$ and $\tilde{f}_{\text{best}} := \tilde{f}(y_{\triangle_2})$ are chosen.

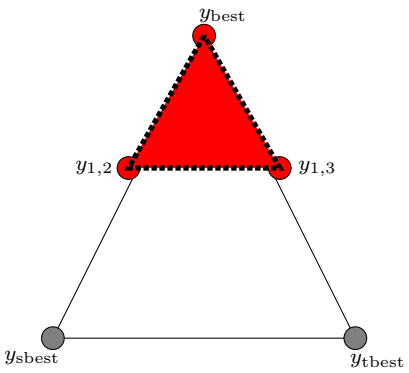

Fig. 6: The first vertex y_{best}, the second vertex y_{sbest}, and the third vertex y_{tbest} of $\triangle_1$ are the three best points found so far by `MADFO`, sorted in ascending order. $y_{1,2}$ is the mean of the points y_{best} and y_{sbest} and $y_{1,3}$ is the mean of the points y_{best} and y_{tbest}. They are the two vertices of $\triangle_2$, which are not necessarily best points.

3 Numerical results

In this section, we compare `MADFO` with the state-of-the-art solvers on noisy test problems obtained from all 157 noiseless unconstrained `CUTEst` test functions in dimensions $2 \leq n \leq 20$ from the collection of Gould et al. [19]. 17 of these functions are quadratic, 80 more are a sum of squares, and the remaining 60 are neither quadratic nor a sum of squares. For each test function, we use absolute and relative, uniform and Gaussian noise with the 8 noise levels $\omega = 10^k$ for $k = -5, -4, \ldots, 2$ and the 3 target accuracies $\varepsilon \in \{10^{-4}, 10^{-3}, 10^{-2}\}$. This produces a total of $157 \times 2 \times 2 \times 8 \times 3 = 15072$ noisy test problems, including:

- $17 \times 2 \times 2 \times 8 \times 3 = 1632$ noisy test problems with quadratic objective functions;

- $80 \times 2 \times 2 \times 8 \times 3 = 7680$ noisy test problems whose objective functions are a sum of squares;

- $60 \times 2 \times 2 \times 8 \times 3 = 5760$ other noisy test problems whose objective functions are none of quadratic and a sum of squares.

For all compared solvers, the default values of the tuning parameters are used.

`MAES` solvers: `MADFO` and `FMAES`

• `MADFO` is a Matlab implemented version of the new techniques discussed in the paper. The source code is obtainable from

`https://github.com/GS1400/MADFO`.

Following [11], we selected as defaults

$$\lambda := 4 + \lfloor 3 \ln n \rfloor, \quad \mu := \lfloor \lambda/2 \rfloor, \quad w_i^0 := \ln\Big(\mu + \frac{1}{2}\Big) - \ln i,$$

$$w_i := \frac{w_i^0}{\sum_{j=1}^{\mu} w_j^0} \quad \text{for } i = 1, \ldots, \mu, \quad \mu_{\rm rec} := \frac{1}{\sum_{j=1}^{\mu} w_j^2},$$

$$c_\sigma := \min\Big(1.999, \frac{\mu_{\rm rec} + 2}{n + \mu_{\rm rec} + 5}\Big), \quad \overline{c}_\sigma := \sqrt{c_\sigma(2 - c_\sigma)\mu_{\rm rec}},$$

$$e_\sigma := \sqrt{n}\big(1 - 1/(4n) - 1/(21n^2)\big), \quad c_1 := 2/\big((n + 1.3)^2 + \mu_{\rm rec}\big),$$

$$c_\mu := \min\left\{1 - c_1, \frac{2(\mu_{\rm rec} - 2 + 1)/\mu_{\rm rec}}{(n+2)^2 + \mu_{\rm rec}}\right\},$$

$$d_\sigma := 1 + c_\sigma + 2\max\left\{0, \sqrt{\frac{\mu_{\rm rec} - 1}{n+1}} - 1\right\}.$$

In contrast to [11], we added the upper bound 1.999 on c_σ since $\overline{c}_\sigma$ has to be a real value. Other tuning parameters for `MADFO` are:

$\gamma = 10^{-12}$, $\sigma_0 = 1$, $\gamma_e = 2$, $\sigma_{\min} = 10^{-2}$, $\sigma_{\max} = 0.5$, $\kappa = 5$, $N = 10$, and $q = e$.

• `FMAES` by Beyer [11], obtained from

`https://homepages.fhv.at/hgb/downloads.html`,

uses an effective matrix adaptation evolution strategy.

Line search solvers: `SDBOX`, `VRBBO`, `VRDFON`

The details of the three line search solvers are as follows:

• `SDBOX` is a derivative-free algorithm for bound-constrained optimization problems discussed in [40], downloaded from

`http://www.iasi.cnr.it/~liuzzi/DFL/index.php/list3`.

• VRBBO is a randomized algorithm by Kimiaei and Neumaier [32]; it can be downloaded from

https://doi.org/10.5281/zenodo.5648165.

• VRDFON is a randomized algorithm by Kimiaei [30]; it can be downloaded from

https://github.com/GS1400/VRDFON.

Direct search solvers: NMSMAX and BFO

• NMSMAX by Higham [26], obtained from

http://www.ma.man.ac.uk/~higham/mctoolbox/

is a version of Nelder–Mead simplex method for direct search optimization algorithms.

• BFO, available at

https://github.com/m01marpor/BFO,

is a trainable stochastic derivative-free solver for mixed integer bound-constrained optimization by Porcelli and Toint [45].

Model-based solvers: UOBYQA and NOMAD

UOBYQA by Powell [46] is obtained from

https://www.pdfo.net/docs.html

and NOMAD [5,7,37] is available at [6].

Other codes such as BCDFO [22] and NELDER [28] were not used because BCDFO is a model-based solver, comparable to UOBYQA and NOMAD, while NELDER is a Nelder–Mead algorithm comparable to NMSMAX.

3.2 Starting and stopping

The initial point: We used for each test problem the standard initial point provided by the CUTEst collection.

Stopping tests: The convergence speed of the solver s to reach a minimum of the smooth true function f is measured by the quotients

$$q_s := (f_s - f_{\text{opt}})/(f_0 - f_{\text{opt}}) \quad \text{for } s \in \mathcal{S}, \tag{19}$$

where $\mathcal{S}$ denotes the list of compared solvers, f_s denotes the best function value found by the solver s, f_0 denotes the function value at the starting point

(common for all solvers), and f_{opt} denotes the function value at the best known point (in most cases a global minimizer or at least a better local minimizer) found by running a sequence of gradient-based and local/global gradient free solvers; see Appendix B in [32].

For problems with several local minimizers this gives an advantage to solvers that find a better minimizer.

Note that q_s is not available in real applications but can be computed on the test problems since the noise free version is available. We consider a problem *solved* by the solver s if $q_s \leq \varepsilon$ and neither the maximum number `nfmax` of function evaluations nor the maximum allowed time `secmax` in seconds is reached, and *unsolved* otherwise.

$$\varepsilon \in \{10^{-4}, 10^{-3}, 10^{-2}\}, \quad \texttt{secmax} := 360, \quad \texttt{nfmax} := 2000n + 5000$$

were chosen so that the best solver can solve at least half of the problems, unless due to high noise increasing `secmax` and `nfmax` cannot change the efficiency and robustness.

3.3 Profiles

Efficiency and robustness: To find approximate stationary points of the noisy unconstrained DFO problems, we say that a DFO solver is most *efficient* if it has a lowest cost of function evaluations and is most *robust* if it has a highest number of solved problems compared to the other solvers.

Cost measure: We denote the list of compared solvers by $\mathcal{S}$ and the list of problems by $\mathcal{P}$. $c_{p,s}$ denotes a *cost measure* of the solver $s \in \mathcal{S}$ to solve the problem $p \in \mathcal{P}$. Our cost measure is the number `nf` of function evaluations. The efficiency with respect to the cost measure `nf` is called `nf` *efficiency*.

Using this cost measure, the efficiency and robustness of DFO solvers can be identified by four different profiles: Box plot, performance profile (Dolan and Moré [17]), data profile (Moré and Wild [43]), and Morales profile (Morales and Nocedal [42]).

Box plot: A box plot shows a robust statistical summary of five numbers, minimum, first quartile, median (second quartile), third quartile, maximum, in a given data set. In fact, this plot is not the actual number for each of these five numbers, but the number closest to these numbers in the data set, which is a good tool for comparing distributions between groups.

Data profile: The data profile of the solver s, the fraction of problems that the solver s can solve with κ groups of $n_p + 1$ function evaluations, is

$$\delta_s(\kappa) := \frac{1}{|\mathcal{P}|} \left| \left\{ p \in \mathcal{P} \;\middle|\; cr_{p,s} := \frac{c_{p,s}}{n_p + 1} \leq \kappa \right\} \right|. \tag{20}$$

Here n_p is the dimension of the problem p and $cr_{p,s}$ is the *cost ratio* of the solver s to solve the problem p.

Performance profile: The performance profile of the solver s

$$\rho_s(\tau) := \frac{1}{|\mathcal{P}|}\Big|\Big\{p \in \mathcal{P} \;\Big|\; pr_{p,s} := \frac{c_{p,s}}{\min(c_{p,\overline{s}} \mid \overline{s} \in S)} \leq \tau\Big\}\Big|. \tag{21}$$

is the fraction of problems that the performance ratio $pr_{p,s}$ is at most τ. In particular, the fraction of problems that the solver s wins compared to the other solvers is $\rho_s(1)$ and the fraction of problems for sufficiently large τ (or κ) that the solver s can solve is $\rho_s(\tau)$ (or $\delta_s(\kappa)$).

Morales profile: Morales profiles of the two most robust solvers with respect to the relative cost for function evaluations measure the efficiency of each solver by the area of the graphs on its side of the half-space.

Noise profiles: Two noise profiles identify the robustness and efficiency of all compared solvers with respect to the noise levels.

Box plots, data profiles, and performance profiles are based on the problem scales but not on the noise levels, while two noise profiles are based on the noise levels. Hence, these five plots are used to identify the robustness and efficiency of the compared solvers with respect to the problem scales and the noise levels.

3.4 Testing for features of `MADFO`

We compare, on the $157 \times 2 \times 2 \times 8 = 5024$ noisy problems with the target accuracy $\varepsilon = 10^{-4}$, the default version of `MADFO` with the four versions obtained by disabling the following features:

- `goodStepSize`, finding good mutation and recombination step sizes.
- `recomSubDir`, computing recm subspace directions.
- `randomNMLS`, performing randomized non-monotone line search method.
- `triSubPoint`, computing random triangle subspace points.

Table 1 compares the default version of `MADFO` and the versions where one of these four features is dropped.

We see that turning off each feature of `MADFO` reduces the number of solved problems, but turning off the randomized non-monotone line search and random triangle subspace points reduces the number of solved much more than turning off the two other features.

Table 1: Cumulative number of solved problems (# solved) out of 5024 noisy problems for the noise levels $\omega = 10^k$ with $k = -5, -4, \ldots, 2$ and the target accuracy $\varepsilon = 10^{-4}$.

solver	version	# solved
`MADFO`	default	**3316**
`MADFO2`	no `recomSubDir`	3285
`MADFO1`	no `goodStepSize`	3277
`MADFO4`	no `triSubPoint`	2654
`MADFO3`	no `randomNMLS`	2615

The box plots, data profiles and performance profiles in Figure 7 show that `MADFO` and `MADFO2` are more robust and efficient than the other versions. Moreover, the noise profiles show that `MADFO` and `MADFO2` are more robust and efficient than the other versions with the respect to the noise levels.

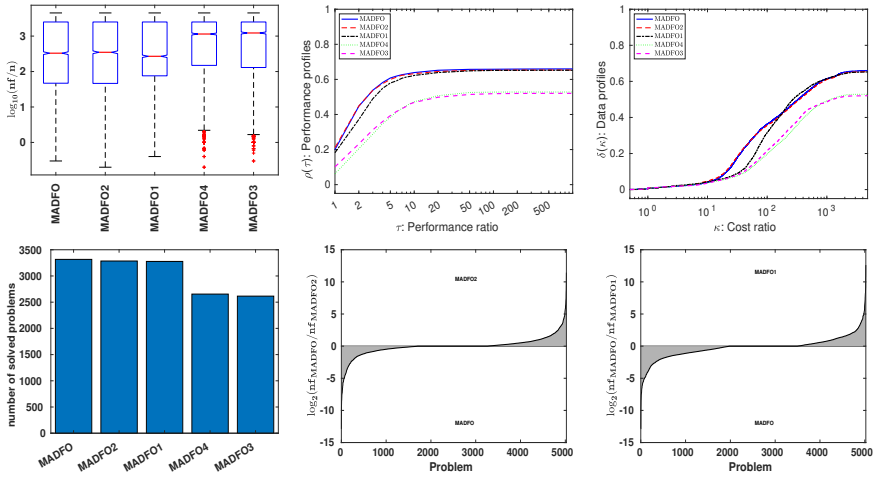

Fig. 7: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with all types of f and small dimensions $1 < n \leq 20$, generated by the absolute/relative uniform and Gaussian noises, the target accuracy $\varepsilon = 10^{-4}$, and the noise levels $\omega = 10^k$ for $k = -5, -4, \ldots, 2$. Problems solved by no solver are ignored. `MADFO` and its four versions used the budgets `secmax` $= 360$ and `nfmax` $= 2000n + 5000$.

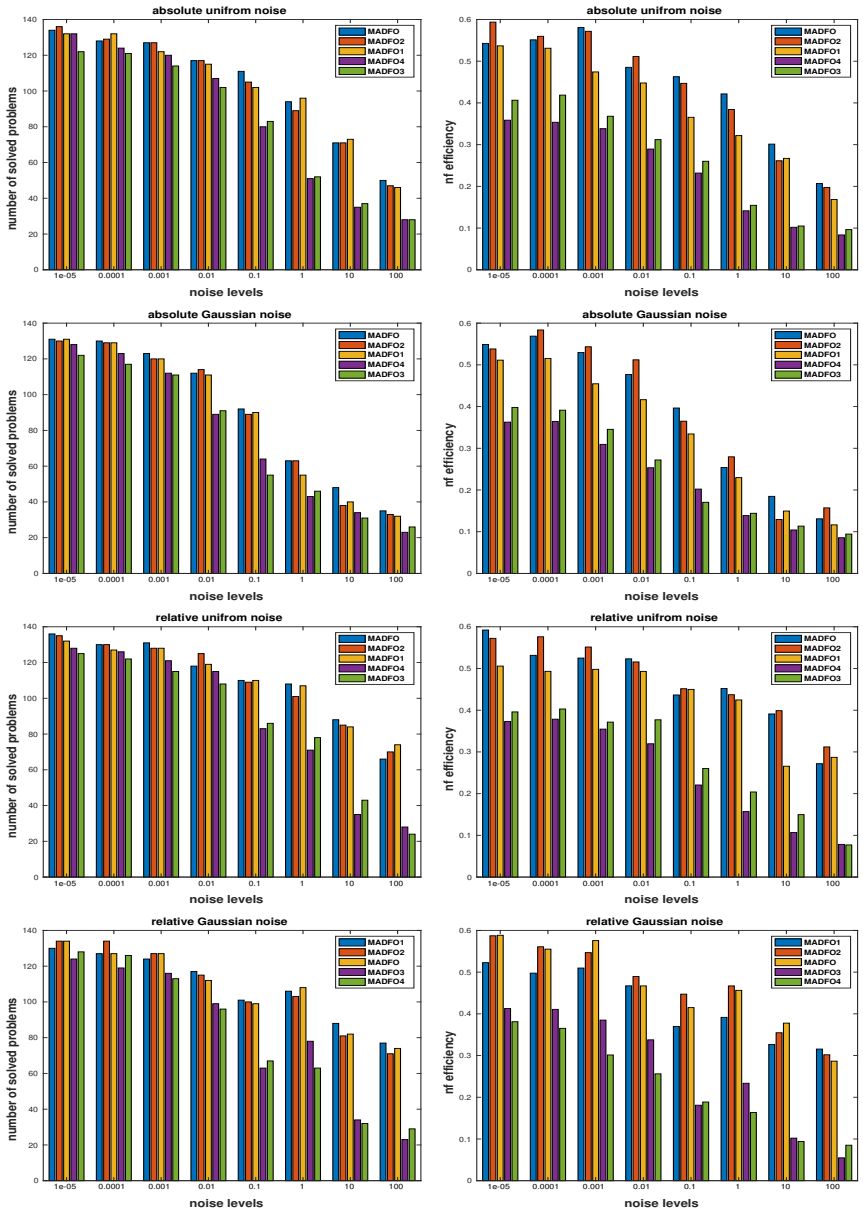

Fig. 8: Noise profiles with respect to the noise levels for the robustness (left) and the `nf` efficiency (right) of `MADFO` and its four versions. Other details as Figure 7.

3.5 Results for problems with the various kinds and levels of noise

Table 2: Cumulative number of solved problems for all noise levels $\omega = 10^{-i}$ with $i = -2, \ldots, 5$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	VRBBO	NMSMAX	BFO
all ϵ	number of solved problems								
$\sum$	11052	9252	9160	8403	8153	7498	7478	7447	7150
	absolute uniform noise								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	VRBBO	NMSMAX	BFO
ϵ	number of solved problems								
10^{-2}	1014	850	897	881	824	810	807	812	767
10^{-3}	948	737	782	762	714	693	687	667	608
10^{-4}	832	610	661	626	600	537	574	513	477
$\sum$	2794	2197	2340	2269	2138	2040	2068	1992	1852
	absolute Gaussian noise								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	VRBBO	NMSMAX	BFO
ϵ	number of solved problems								
10^{-2}	969	821	851	803	797	770	778	778	733
10^{-3}	875	704	754	682	686	649	667	637	574
10^{-4}	734	576	637	553	575	508	528	473	434
$\sum$	2578	2101	2242	2038	2058	1927	1973	1888	1741
	relative uniform noise								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	VRBBO	NMSMAX	BFO
ϵ	number of solved problems								
10^{-2}	1014	927	862	809	755	696	665	708	725
10^{-3}	958	845	785	727	675	617	591	616	606
10^{-4}	887	734	694	619	599	511	523	501	501
$\sum$	2859	2506	2341	2155	2029	1824	1779	1825	1832
	relative Gaussian noise								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	VRBBO	NMSMAX	BFO
ϵ	number of solved problems								
10^{-2}	1011	914	834	752	718	661	624	681	686
10^{-3}	947	823	740	641	644	559	553	597	587
10^{-4}	863	711	663	548	566	487	481	464	452
$\sum$	2821	2448	2237	1941	1928	1707	1658	1742	1725

We summarize our results in Tables 2 and Figure 9. We can see that the new heuristic techniques in MADFO added to FMAES makes MADFO more robust than the other solvers.

From Table 2, we conclude that MADFO, FMAES, NOMAD, VRDFON are more robust solvers than the other compared solvers. In fact, MADFO can solve 1800

more problems than FMAES, 1892 more problems than NOMAD, and 2649 more problems than VRDFON.

Table 4 contains failure diagnostics, indicating how the compared solvers terminate due to reaching secmax, reaching nfmax, and failure for algorithmic reasons. The latter are primarily step failures and decrease failures.

As discussed in Subsection 2.3, we found that, in practice, the proper behavior under scaling is more important than translation invariance. To numerically verify this, we generate the new problems by the change of coordinates $x_{\text{new}} = x + v$ (e.g. $v_i = \pm 1$ with the sign drawn randomly) and transformation to x_{new} ($f(x)$ becomes $f(x_{\text{new}} - v)$). Then, we solve the new problems. Table 3 shows that the results of MADFO are nearly unaffected by a shift of the points, while the results of the most other solvers are somewhat more affected. It is unclear to us why the results of FMAES are strongly affected by the shift.

Table 3: Test for translation invariance: Cumulative number of solved problems for all noise levels $\omega = 10^{-i}$ with $i = -2, \ldots, 5$.

	absolute uniform noise							
	problems without shifted points							
	solvers							
	MADFO	NOMAD	VRDFON	FMAES	VRBBO	SDBOX	NMSMAX	BFO
$\epsilon = 10^{-4}$	number of solved problems							
$\sum$	832	661	626	610	574	537	513	477
	problems with shifted points							
	solvers							
	MADFO	NOMAD	VRDFON	FMAES	VRBBO	SDBOX	NMSMAX	BFO
$\epsilon = 10^{-4}$	number of solved problems							
$\sum$	842	779	733	818	671	610	595	531
variation (%)	0.7962	9.3949	8.5191	16.5605	7.7229	5.8121	6.5287	4.2994

Step failures: Before an approximate stationary point is found, DFO algorithms (like trust region, line search, and direct search) may get stuck because of null steps. UOBYQA uses a trust region algorithm. In most cases, it cannot find a reduction in the inaccurate function value when noise increases or is large. Therefore, the trust region radius is reduced more and more and eventually approaches zero, which is the cause of the trust region failure. Something related is true for the line search solvers (VRBBO and VRDFON) and the direct search solvers (NOMAD, NMSMAX and BFO). In this case, the step sizes go to zero, which is the cause of line search and direct search failures. Since the step sizes are updated by (9) in the MAES solvers (FMAES and MADFO), the step sizes are reduced more slowly than the step sizes of the line search and direct search. Unlike FMAES, MADFO changes and controls the step sizes so that they are neither too small nor too large.

Decrease failures: Due to large noise ($\omega > 0.1$), decrease conditions cannot discard points whose function values are spuriously good. Therefore the noisy

function values reach -10^{12} and DFO algorithms get stuck before finding an approximate stationary point. To have a meaningful comparison for relative noise with the noise level $\omega > 0.1$, we computed the noisy function values by

$$\tilde{f} = \omega(1 + \max(0.1\,\mathrm{rand}, \mathrm{randn}))f$$

in the relative Gaussian noise and by

$$\tilde{f} = \omega(1 + \max(0.1\,\mathrm{rand}, 2\,\mathrm{rand} - 1))f$$

in the relative uniform noise, where rand $\in \mathcal{U}(0,1)$ (uniformly distributed) and randn $\in \mathcal{N}(0,1)$ (normally distributed). Although `FMAES` does not use any decrease condition on $\tilde{f}$ to accept the best point, it uses such a decrease condition to sort inexact function values at the λ mutation points and, accordingly, distribution directions and mutation directions; thus, decrease failures for `FMAES` mean that the sorting process fails. `MADFO` uses two various decrease conditions to accept the best point, and like `FMAES`, it uses the sorting process to calculate recombination points with low inexact function values that can be accepted as the new best point. Hence, for `MADFO`, decrease failures mean that two decrease conditions fail to find best point whose function values are not spuriously good or the sorting process fails and poor recombination points with spuriously good function values are accepted as the new best point.

Other bugs: Occasionally solvers fail for other algorithmic reasons.

Table 4: # `nfmax`, # `secmax`, # `alg` count number of problems (from a total of 15072) for which each solver was terminated due to reaching `secmax`, reaching `nfmax`, and failure for algorithmic reasons (step failures, decrease failures, or other bugs), respectively.

	solvers								
	`MADFO`	`FMAES`	`NOMAD`	`VRDFON`	`UOBYQA`	`SDBOX`	`VRBBO`	`NMSMAX`	`BFO`
# `nfmax`	4002	2579	0	6664	0	7571	7592	7622	123
# `secmax`	0	0	0	0	0	1	0	0	0
# `alg`	18	3241	5912	5	6919	2	2	3	7799
# `total`	4020	5820	5912	6669	6919	7574	7594	7625	7922

Table 4 contains the number of problems (out of a total of 15072) where each solver was terminated for (exactly one of) reaching `secmax`, reaching `nfmax`, and failure for algorithmic reasons. We see that the time limit chosen was quite generous; `SDBOX` was stopped due to reaching `secmax` only once.

The first row of Figure 9 contains box plots, performance profiles, and data profiles of all 9 compared solvers in terms of `nf`, while the second row contains bar graphs of all 9 compared solvers in terms of the number of solved problems and two Morales profiles of the three most robust solvers in terms of `nf`, the goal of which is to show which solver is the most efficient.

Figure 9 reflects the fact that `MADFO` is much more robust than the other solvers. `MADFO` solves 11052 out of 15072 and hence is much more robust than the other solvers, while the second to forth more robust solvers are `FMAES`, `NOMAD`, and `VRDFON`. The performance profiles of this figure in terms of `nf` show that the efficiency of `MADFO` comes at a price in terms of the cost: with respect to the `nf` efficiency, `MADFO` only takes the fourth place after `UOBYQA` (first), `VRDFON` (second), and `NOMAD` (third). Indeed, `MADFO` is more efficient than the other solvers for 10% of problems, while `UOBYQA` is more efficient than the other solvers for 20% of problems, and `VRDFON` is more efficient than the other solvers for 12% of problems.

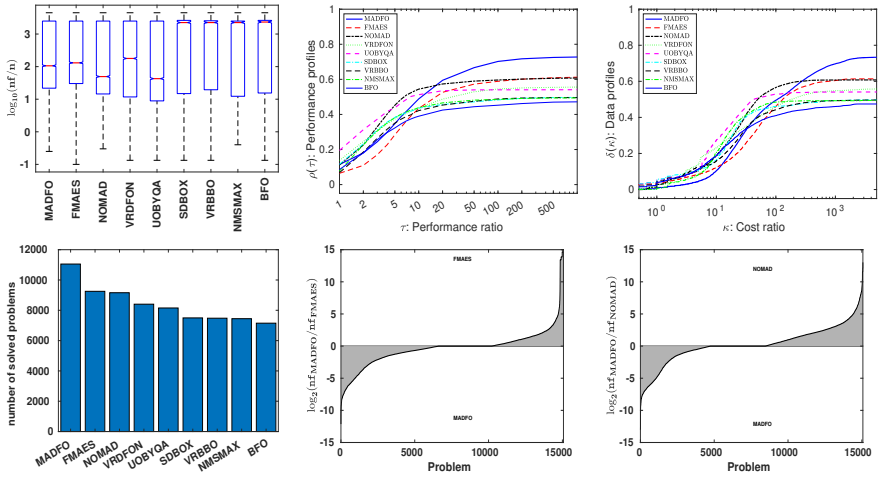

Fig. 9: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with all types of f and small dimensions $1 < n \leq 20$, generated by the absolute/relative uniform and Gaussian noises, the three target accuracies $\varepsilon \in \{10^{-4}, 10^{-3}, 10^{-2}\}$, and the noise levels $\omega = 10^k$ for $k = -5, -4, \ldots, 2$. Problems solved by no solver are ignored. All 9 compared solvers used the budgets $\texttt{secmax} = 360$ and $\texttt{nfmax} = 2000n + 5000$.

4 Recommendation

This section provides three recommendations on the type of objective function, the type of noise, and the different noise levels.

4.1 Classified by the type of objective function

This section provides some recommendations on the type of objective function: Quadratic, a sum of squares, and other (neither quadratic nor a sum of squares).

Our findings form Figures 10–12 and Tables 5–7 are here summarized:

• For problems with the quadratic f, `MADFO` was much more robust than the other solvers since it solved 1456 out of 1632 problems, while the second more robust solver `NOMAD` solved 1241 out of 1632 problems. In terms of relative cost for `nf`, `MADFO` takes the fourth place after `UOBYQA` (first), `NOMAD` (second), and `VRDFON` (third).

• For problems whose f is a sum of squares, `MADFO` was much more robust than the other solver since it solved 5436 out of 7680 problems, while the second more robust solver `FMAES` solved 4949 problems. In terms of relative cost for `nf`, `MADFO` only is more efficient than `FMAES`, while `UOBYQA` and `SDBOX` are the first and second more efficient solvers, respectively.

• For problems with the other types of f, `MADFO` was much more robust and efficient than the other solvers. `MADFO` solved 4160 out of 5760 problems, while the second robust solver `NOMAD` solved 3277 out of 5760 problems. The first more efficient solver `MADFO` for 19% of the problems and the second more efficient solver `VRDFON` for 18% of the problems had the lowest relative cost for `nf` compared to the other solvers.

Table 5: Cumulative number of solved problems with the quadratic f for all noise levels $\omega = 10^k$ with $k = -5, -4, \ldots, 2$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	MADFO	NOMAD	FMAES	UOBYQA	VRDFON	NMSMAX	VRBBO	BFO	SDBOX
all ε	number of solved problems from a total of 1632								
$\sum$	**1456**	1241	1231	1201	1113	1097	964	960	917

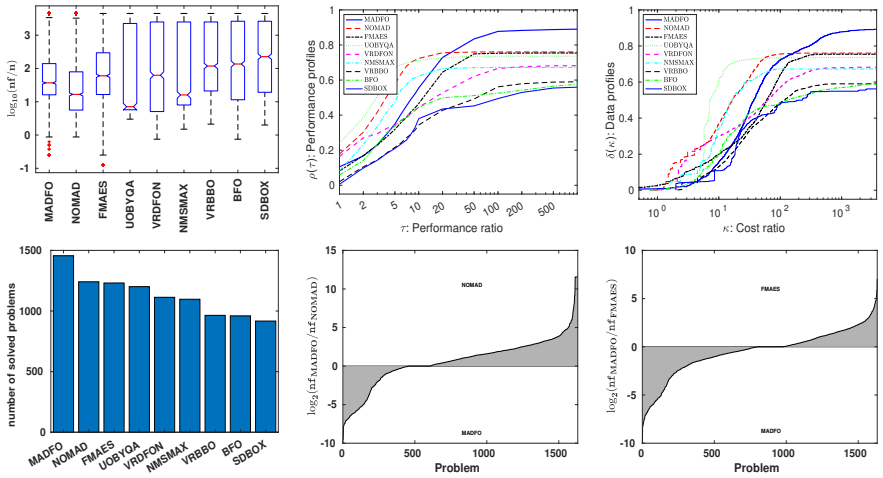

Fig. 10: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with the quadratic f. Other details as Figure 9.

Table 6: Cumulative number of solved problems with f as a sum of squares for all noise levels $\omega = 10^k$ with $i = -5, -4, \ldots, 2$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	NMSMAX	VRBBO	BFO
all ε	number of solved problems from a total of 7680								
$\sum$	**5436**	4949	4642	4302	4291	3903	3898	3867	3724

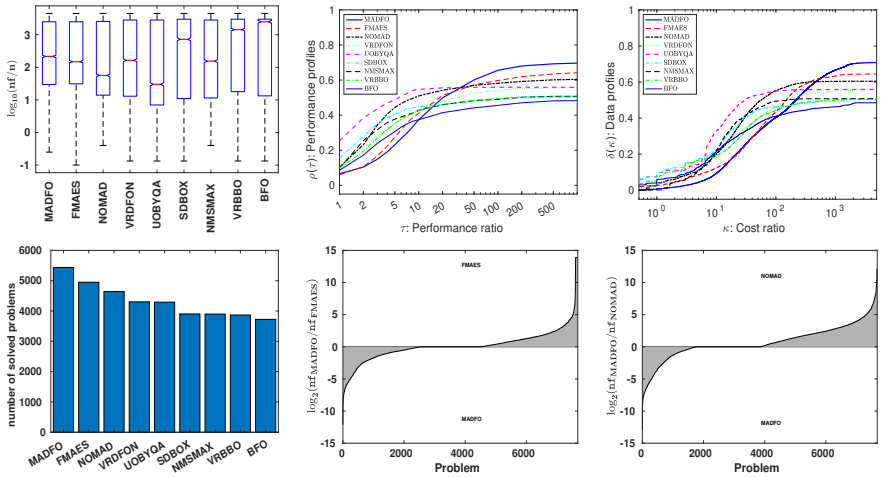

Fig. 11: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with f as a sum of squares. Other details as Figure 9.

Table 7: Cumulative number of solved problems with the other types of f for the 8 noise levels $\omega = 10^k$ with $i = -5, -4, \ldots, 2$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	MADFO	NOMAD	FMAES	VRDFON	SDBOX	UOBYQA	VRBBO	BFO	NMSMAX
all ε	number of solved problems from a total of 5760								
$\sum$	**4160**	3277	3072	2988	2678	2661	2647	2466	2452

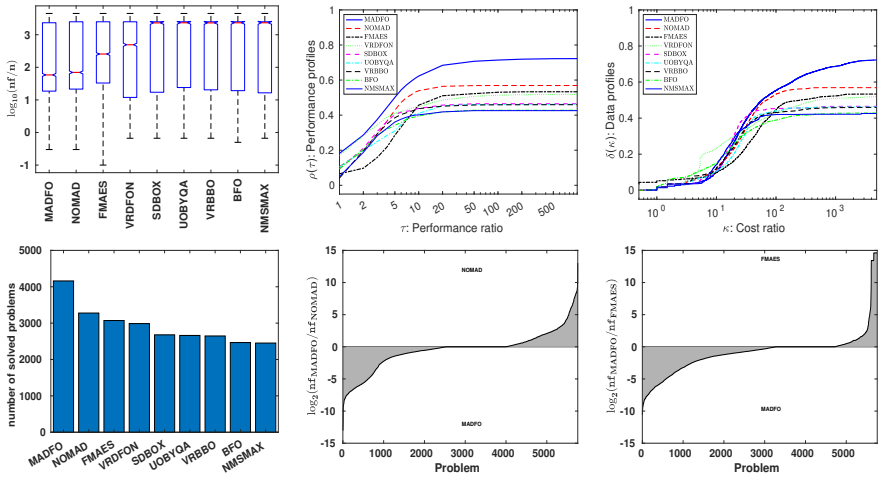

Fig. 12: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with the other types of f. Other details as Figure 9.

4.2 Classified by the noise level

This section provides a recommendation on the small ($\omega < 0.1$) and large ($\omega \geq 0.1$) levels of noise.

For all types of noise and the three different target accuracies, we summarize here our results from Figures 13 and 14 and Tables 8 and 9:

• At small noise levels `MADFO` is more robust than the others. It solved 6358 out of 7536 problems, while the second more robust solver `NOMAD` solved 6175 out of 7536 problems. `MADFO` ranks last among the 9 compared solvers in terms of relative cost for `nf`.

• At large noise levels, `MADFO` is much more robust and efficient than the others. It solved 4694 out of 7536 problems, while the second more robust solver `FMAES` solved 3330 out of 7536 problems. The first more efficient solver `MADFO` has the lowest relative cost for `nf` on 20% of problems, while the second more efficient solver `UOBYQA` has the lowest relative cost for `nf` on 10% of problems.

Table 8: Cumulative number of solved problems for the 4 small noise levels $\omega = 10^k$ with $i = -5, -4, -3, -2$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	`MADFO`	`NOMAD`	`FMAES`	`VRDFON`	`UOBYQA`	`VRBBO`	`NMSMAX`	`SDBOX`	`BFO`
all ε	number of solved problems from a total of 7536								
$\sum$	**6358**	6175	5922	5863	5824	5634	5392	5313	5131

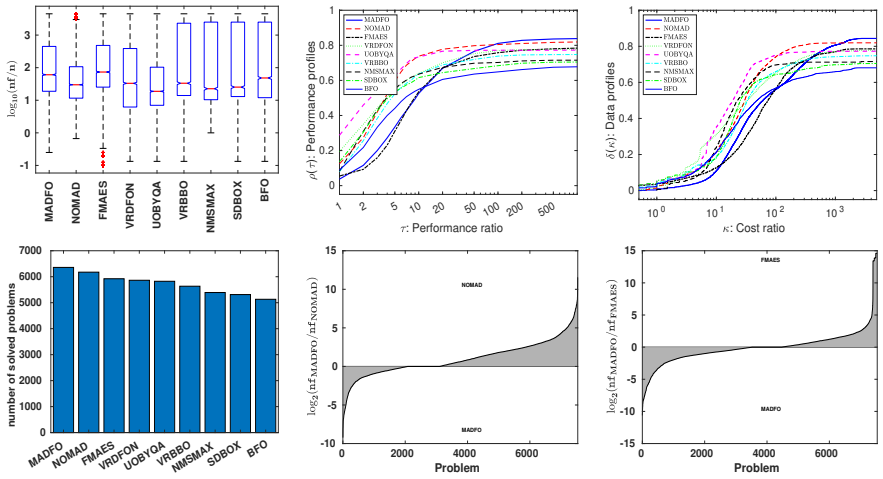

Fig. 13: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with all types of f and the 4 small noise levels $\omega = 10^k$ for $k = -5, -4, -3, -2$. Other details as Figure 9.

Table 9: Cumulative number of solved problems for the 4 large noise levels $\omega = 10^k$ with $i = -1, 0, 1, 2$.

	absolute/relative uniform and Gaussian noises								
	solvers								
	MADFO	FMAES	NOMAD	VRDFON	UOBYQA	SDBOX	NMSMAX	BFO	VRBBO
all ε	number of solved problems from a total of 7536								
$\sum$	**4694**	3330	2985	2540	2329	2185	2055	2019	1844

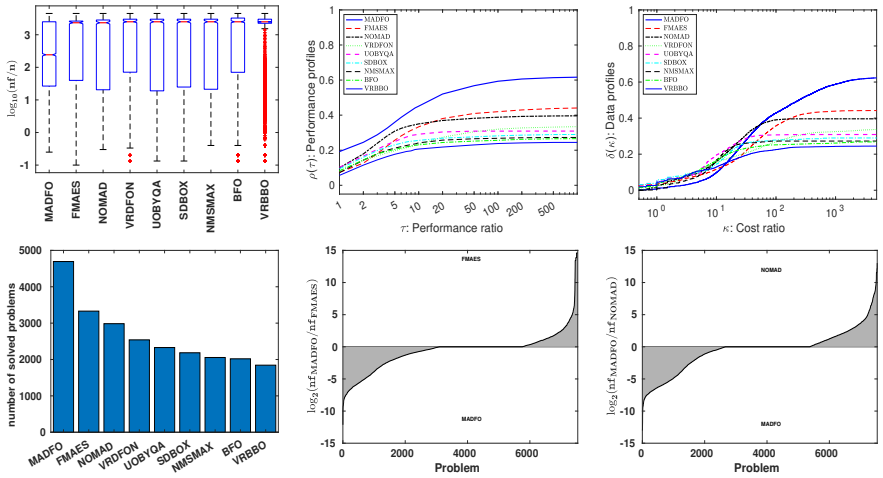

Fig. 14: The box plots, data profiles and performance profiles (first row) and the bar graphs and the Morales profiles (second row) for noisy problems with all types of f and the 4 large noise levels $\omega = 10^k$ for $k = -1, 0, 1, 2$. Other details as Figure 9.

4.3 Classified by the kind and level of noise and the target accuracy

This section provides a recommendation for the type and level of noise and the target accuracy.

We summarize here our results from Figures 15–18. For all types of noise, `MADFO` is much more robust than the other solvers. When increasing the level of noise, `MADFO` not only stays in the first rank for robustness, but also becomes the first more efficient solver for large noise.

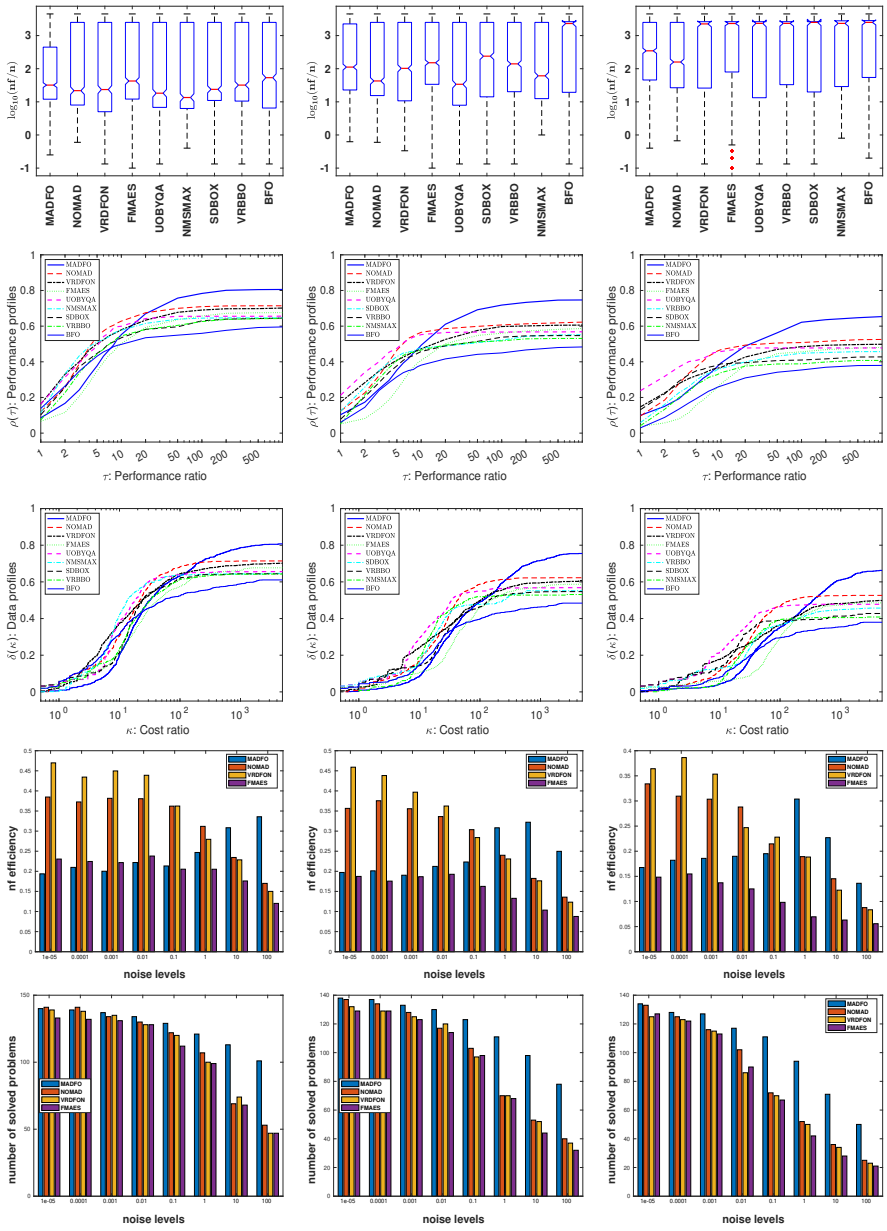

Fig. 15: For the absolute uniform noise: Box plots (first row), performance profiles (second row), data profiles (third row), and noise profiles (forth row) in terms of `nf` and noise profiles (fifth row) in terms of number of solved problems for the four more robust solvers. Other details as Figure 9.

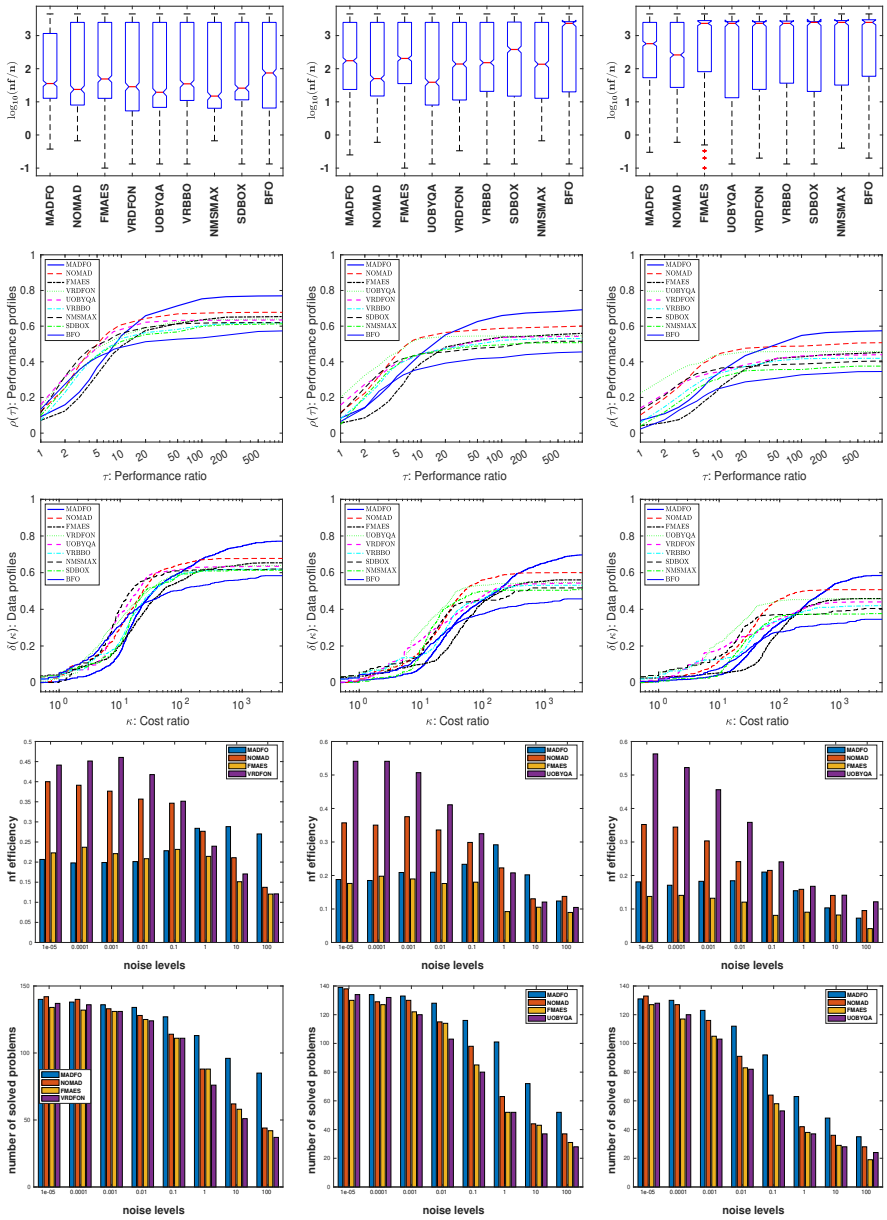

Fig. 16: For the absolute Gaussian noise: Box plots (first row), performance profiles (second row), data profiles (third row), and noise profiles (forth row) in terms of `nf` and noise profiles (fifth row) in terms of number of solved problems for the four more robust solvers. Other details as Figure 9.

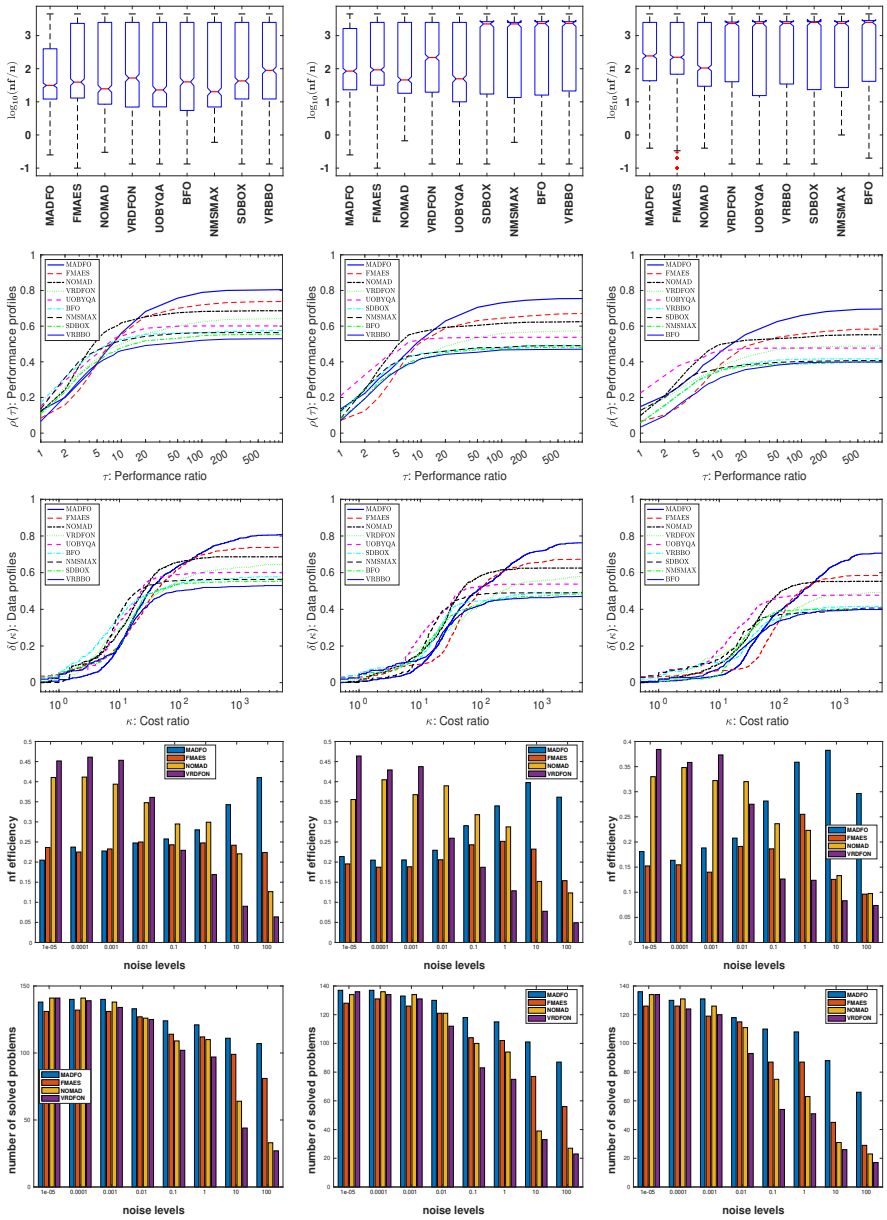

Fig. 17: For the relative uniform noise: Box plots (first row), performance profiles (second row), data profiles (third row), and noise profiles (forth row) in terms of `nf` and noise profiles (fifth row) in terms of number of solved problems for the four more robust solvers. Other details as Figure 9.

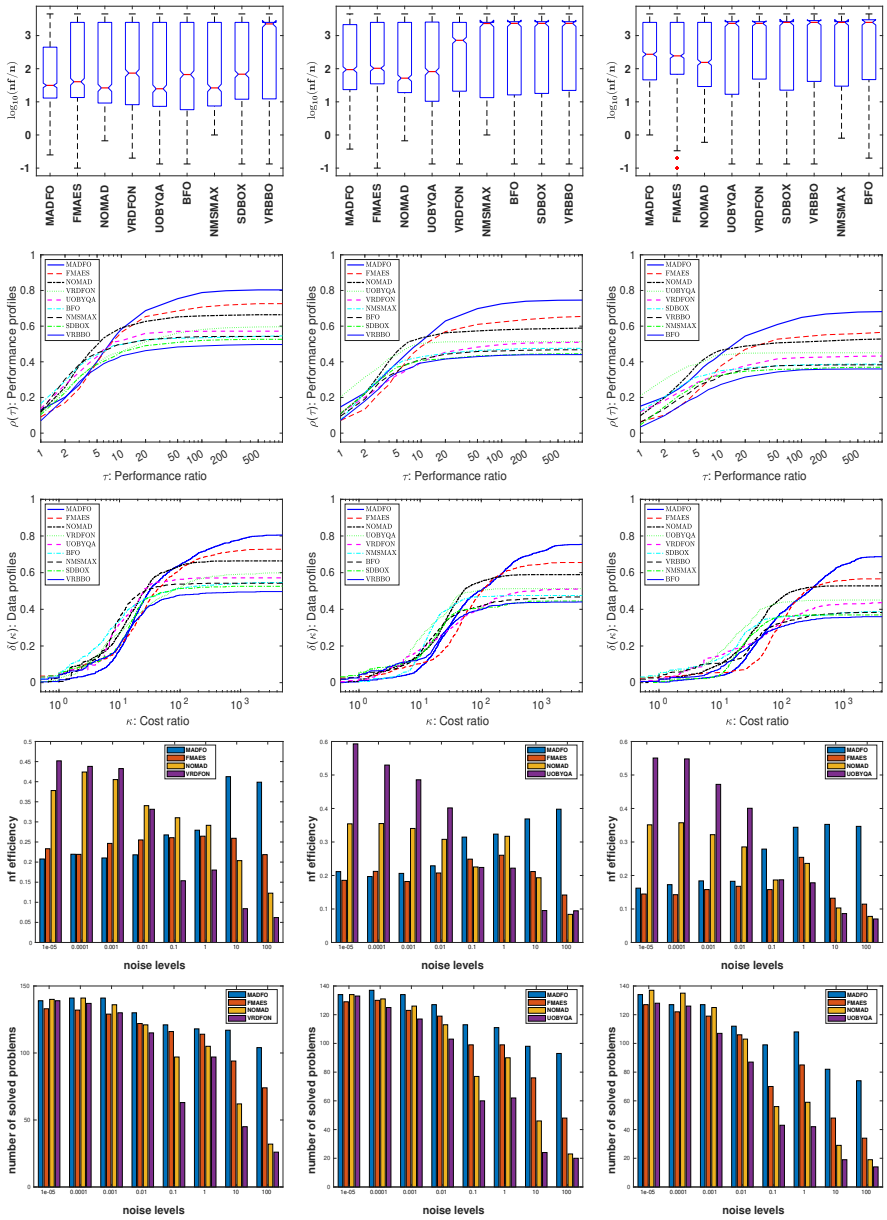

Fig. 18: For the relative Gaussian noise: Box plots (first row), performance profiles (second row), data profiles (third row), and noise profiles (forth row) in terms of `nf` and noise profiles (fifth row) in terms of number of solved problems for the four more robust solvers. Other details as Figure 9.

5 Conclusion

In this paper we discuss MADFO, an improved matrix adaptation strategy for noisy DFO problems. MADFO uses several new features, such as good (neither too small nor too large) mutation and recombination step sizes, recm subspace directions as a useful substitute (in practice against a failing sorting process in the section phase) for recm directions, a randomized non-monotone line search method to find mainly recombination points with low inexact function values, and random triangle subspace points to find a useful substitute for discarded recombination points with large inexact function values. Turning off each feature of MADFO reduces robustness, but turning off the randomized non-monotone line search method and random triangle subspace points reduces robustness much more than turning off the other two features.

As a result of our recommendations, MADFO is much more robust than the other solvers with respect to the type and level of noise and the types of objective function. When noise is large, MADFO is not only much more robust than the other solvers, but also more efficient than them in terms of the relative cost for the number of function evaluations. For problems with other types (neither a sum of squares nor quadratic) of f, MADFO is much more robust and efficient than the other solvers, while for problems whose f are as a sum of squares or quadratic MADFO is only much more robust than the other solvers.

Acknowledgment The first author acknowledges the financial support of the Austrian Science Foundation under Project No. P 34317. We would like to thank the associated editor, the technical editor and two unknown referees for their valuable comments.

References

1. M. Ahookhosh and K. Amini. An efficient nonmonotone trust-region method for unconstrained optimization. *Numer. Algorithms* **59** (2011), 523–540.
2. K. Amini, H. Esmaeili, and M. Kimiaei. A nonmonotone trust-region-approach with nonmonotone adaptive radius for solving nonlinear systems. *IJNAO* **6** (2016), 101–121.
3. K. Amini, M. Kimiaei, and H. Khotanlou A non-monotone pattern search approach for systems of nonlinear equations. *Int. J. Comput. Math.* **96** (2019), 33–50.
4. S. Araki and S. Nishizaki. Call-by-Name Evaluation of RPC and RMI Calculi. In *Theory and Practice of Computation: Proceedings of Workshop on Computation: Theory and Practice WCTP2013* (2015).
5. C. Audet, J. Dennis, Jr., Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.* **17** (2006), 188–217.
6. C. Audet, S. Le Digabel, V. Rochon Montplaisir, C. Tribes, The NOMAD project, Software available `https://www.gerad.ca/nomad/`
7. C. Audet, S. Le Digabel, C. Tribes, The Mesh Adaptive Direct Search Algorithm for Granular and Discrete Variables. *SIAM J. Optim.* **29** (2019), 1164–1189.
8. C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization.* Springer International Publishing (2017).

9. A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*. IEEE (2005).
10. A. S. Berahas, R. H. Byrd, and J. Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.* **29** (2019), 965–993.
11. H. G. Beyer. Design principles for matrix adaptation evolution strategies. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. ACM (2020).
12. H. G. Beyer and B. Sendhoff. Simplify your covariance matrix adaptation evolution strategy. *IEEE Trans. Evol. Comput.* **21** (2017), 746–759.
13. E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Optim.* **10** (1999), 1196–1211.
14. R. Chen. *Stochastic Derivative-Free Optimization of Noisy Functions*. PhD thesis, Lehigh University (2015). Theses and Dissertations. 2548.
15. A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics (2009).
16. M. A. Diniz-Ehrhardt, J. M. Martínez, and M. Raydan. A derivative-free nonmonotone line-search technique for unconstrained optimization. *J. Comput. Appl. Math.* **219** (2008), 383–397.
17. E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002), 201–213.
18. C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.* **15** (1995), 585–608.
19. N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* **60** (2015), 545–557.
20. S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim* **25** (2015), 1515–1541.
21. S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Complexity and global rates of trust-region methods based on probabilistic models. *IMA J. Numer. Anal.* **38** (2017), 1579–1597.
22. S. Gratton, Ph. L. Toint, and A. Tröltzsch. An active-set trust-region method for derivative-free nonlinear bound-constrained optimization. *Optim. Methods Softw.* **26** (2011), 873–894.
23. L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for newton's method. *SIAM J. Numer. Anal.* **23** (1986), 707–716.
24. L. Grippo and F. Rinaldi. A class of derivative-free nonmonotone optimization algorithms employing coordinate rotations and gradient approximations. *Comput. Optim. Appl.* **60** (2014), 1–33.
25. N. Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).
26. N. J. Higham. Optimization by direct search in matrix computations. *SIAM J. Matrix Anal. Appl.* **14** (1993), 317–333.
27. W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.* **35** (2008), 1–25.
28. C. T. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics (1999).
29. M. Kimiaei. A new class of nonmonotone adaptive trust-region methods for nonlinear equations with box constraints. *Calcolo* **54** (2016), 769–812.
30. M. Kimiaei. An improved randomized algorithm with noise level tuning for large-scale noisy unconstrained DFO problems. `http://www.optimization-online.org/DB_HTML/2020/09/8007.html` (2023).
31. M. Kimiaei, H. Esmaeili, and F. Rahpeymaii. A trust-region method using extended nonmonotone technique for unconstrained optimization. *Iran. J. Math. Sci.* **16** (2021), 15–33.

32. M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.* **14** (2022), 365–414.
33. M. Kimiaei and A. Neumaier. A new limited memory method for unconstrained nonlinear least squares. *Soft Comput.* **26** (2022), 465–490.
34. M. Kimiaei, A. Neumaier, and P. Faramarzi. New subspace method for unconstrained derivative-free optimization. *ACM. Trans. Math. Softw.* **49** (2023), 1–28.
35. M. Kimiaei and F. Rahpeymaii. A new nonmonotone line-search trust-region approach for nonlinear systems. *TOP* **27** (2019), 199–232.
36. J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.* **28** (2019), 287–404.
37. S. Le Digabel, Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM. Trans. Math. Softw.* **37** (2011), 1–15.
38. D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.* **45** (1989), 503–528.
39. I. Loshchilov, T. Glasmachers, and H. G. Beyer. Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Trans. Evol. Comput.* **23** (2019), 353–358.
40. S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.* **21** (2002), 119–142.
41. MATLAB Optimization Toolbox. The MathWorks, Natick, MA, USA.
42. J. L. Morales and J. Nocedal. Remark on "algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization". *ACM. Trans. Math. Softw.* **38** (2011), 1–4.
43. J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.* **20** (2009), 172–191.
44. J. J. Moré and S. M. Wild. Estimating derivatives of noisy simulations. *ACM Trans. Math. Softw.* **38** (2012), 1–21.
45. M. Porcelli and P. Toint. Global and local information in structured derivative free optimization with BFO. *arXiv: Optimization and Control* (2020). `https://doi.org/10.48550/arXiv.2001.04801`
46. M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* **92** (2002), 555–582.
47. M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA. J. Numer. Anal.* **28** (2008), 649–664.
48. L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Global. Optim.* **56** (2012), 1247–1293.
49. H. J. M. Shi, M. Q. Xuan, F. Oztoprak, and J. Nocedal. On the numerical performance of finite-difference-based methods for derivative-free optimization. *Optim. Methods Softw.* **17** (2022), 1–23.
50. Ph. L. Toint. An assessment of nonmonotone linesearch techniques for unconstrained optimization. *SIAM J. Sci. Comput.* **17** (1996), 725–739.
51. V. J. Torczon. *Multidirectional search: A direct search algorithm for parallel machines.* PhD thesis, Diss., Rice University (1989).
52. S. M. Wild, R. G. Regis, and C. A. Shoemaker. ORBIT: Optimization by radial basis function interpolation in trust-regions. *SIAM J. Sci. Comput.* **30** (2008), 3197–3219.
53. M. H. Wright. Direct search methods: Once scorned, now respectable. *Pitman Research Notes in Math. Series* (1996), 191–208.