Supplemental Material "MATRS – Heuristic methods for derivative-free bound-constrained mixed-integer optimization" Mathematical Programming Computation (2025)

Morteza Kimiaei
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: kimiaeim83@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~kimiaei`

Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: `http://www.mat.univie.ac.at/~neum`

Abstract. This supplemental material provides flowcharts for all subroutines of the `MATRS` solver [11], explains their structures, and provides a comparison between `MATRS` and state-of-the-art continuous and integer DFO solvers.

Contents

1 Flowcharts of all subroutines **2**

1.1 A continuous mutation phase 2

1.1.1 `get`α 2

1.1.2 `updatePoint` 3

1.1.3 `cLSS` 5

1.1.4 `cMutation` 7

1.2 Selection 9

1.3 A continuous recombination phase 10

1.3.1 `updateM` 10

1.3.2 `cRecom` 11

1.4 `cTRS` 13

1.5 `cMATRS` 14

1.6 An integer mutation phase 14
1.7 iRecom 16
1.8 iTRS 17
1.9 iMATRS 18
1.10 A mixed-integer MATRS 19
1.10.1 miLSS 19
1.10.2 miMATRS 19

2 Results for global, bcp, and prince **23**

3 Results for globalInt, bcpInt, and princeInt **32**

Supplementary information (suppMat_MATRS.pdf) for the paper [12] is discussed here, which is available at

https://github.com/GS1400/SuppMat_MATRS.

Section 1 presents flowcharts for all subroutines of MATRS and explains their structures. Section 2 provides a comparison between MATRS and state-of-the-art continuous DFO solvers, while Section 3 compares MATRS with state-of-the-art integer DFO solvers.

1 Flowcharts of all subroutines

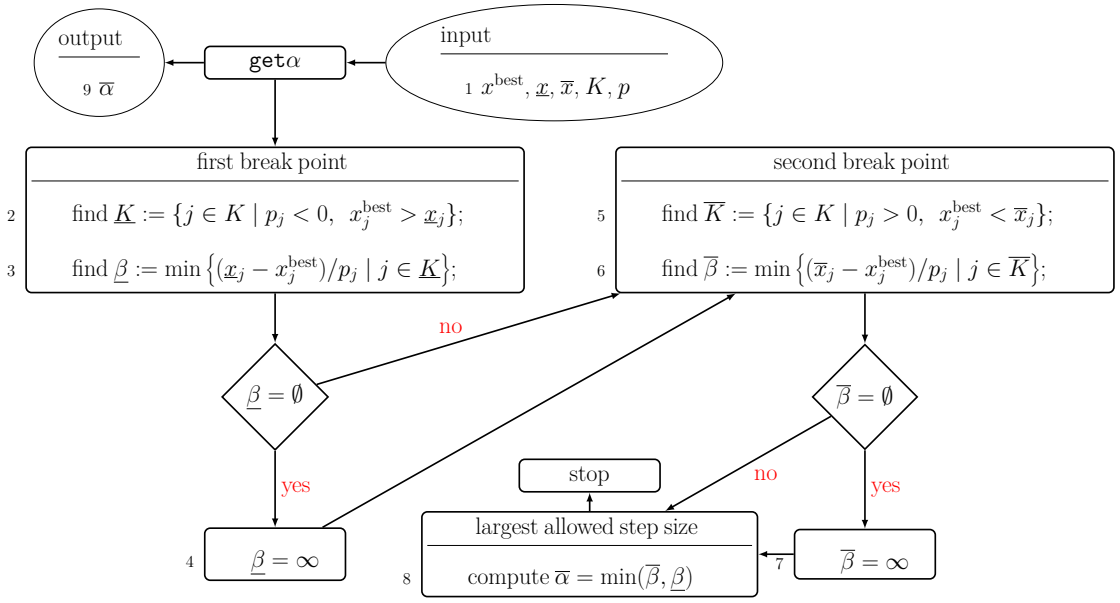
This section discusses flowcharts of several important subroutines of MATRS [12] whose Matlab codes are available in the MATRS package [11].

1.1 A continuous mutation phase

1.1.1 getα

Algorithm 1 is pseudocode for getα. getα computes the two break points $\underline{\beta}$ and $\overline{\beta}$ in lines 3 and 6 and takes their minimum $\overline{\alpha}$ in line 8. Here p is a given trial direction. If cMutation calls getα, p is a mutation direction. Otherwise, if cRecom calls getα, p is a recombination mutation direction. In addition, if miMATRS calls getα, p is a combination direction. If the first break point does not exist, it is chosen to be infinity and the same applies to the second break point. After computing $\overline{\alpha}^i$, if $\overline{\alpha}^i = 0$ for the continuous variables and $\overline{\alpha}^i < 1$ for the integer variables, then there is no feasible trial point along $\pm p_{\mathrm{md}}$; hence at most n_{dd} times the corresponding distribution direction p_{dd} and then p_{md} are recomputed

Algorithm 1 Pseudocode for `get`α



for a possible finding of some feasible trial points along new $\pm p_{\text{md}}$. Here $n_{\text{dd}} = n'_{\text{dd}}$ for the continuous search and $n_{\text{dd}} = n''_{\text{dd}}$ for the integer search, where n'_{dd} and n''_{dd} are the tuning parameters. To simplify `get`α, this improvement does not appear in Algorithm 1. In addition, in the Matlab code, `get`α also computes the initial step sizes for pseudocode of `cMutation`, `iMutation`, `cRecom`, `iRecom`, and `miMATRS`. Since there are different formulas for calculating the initial step sizes in the present paper, we calculate these step sizes after calculating $\overline{\alpha}$ by `get`α in all the mentioned pseudocode so that these pseudocode are easy to read.

1.1.2 `updatePoint`

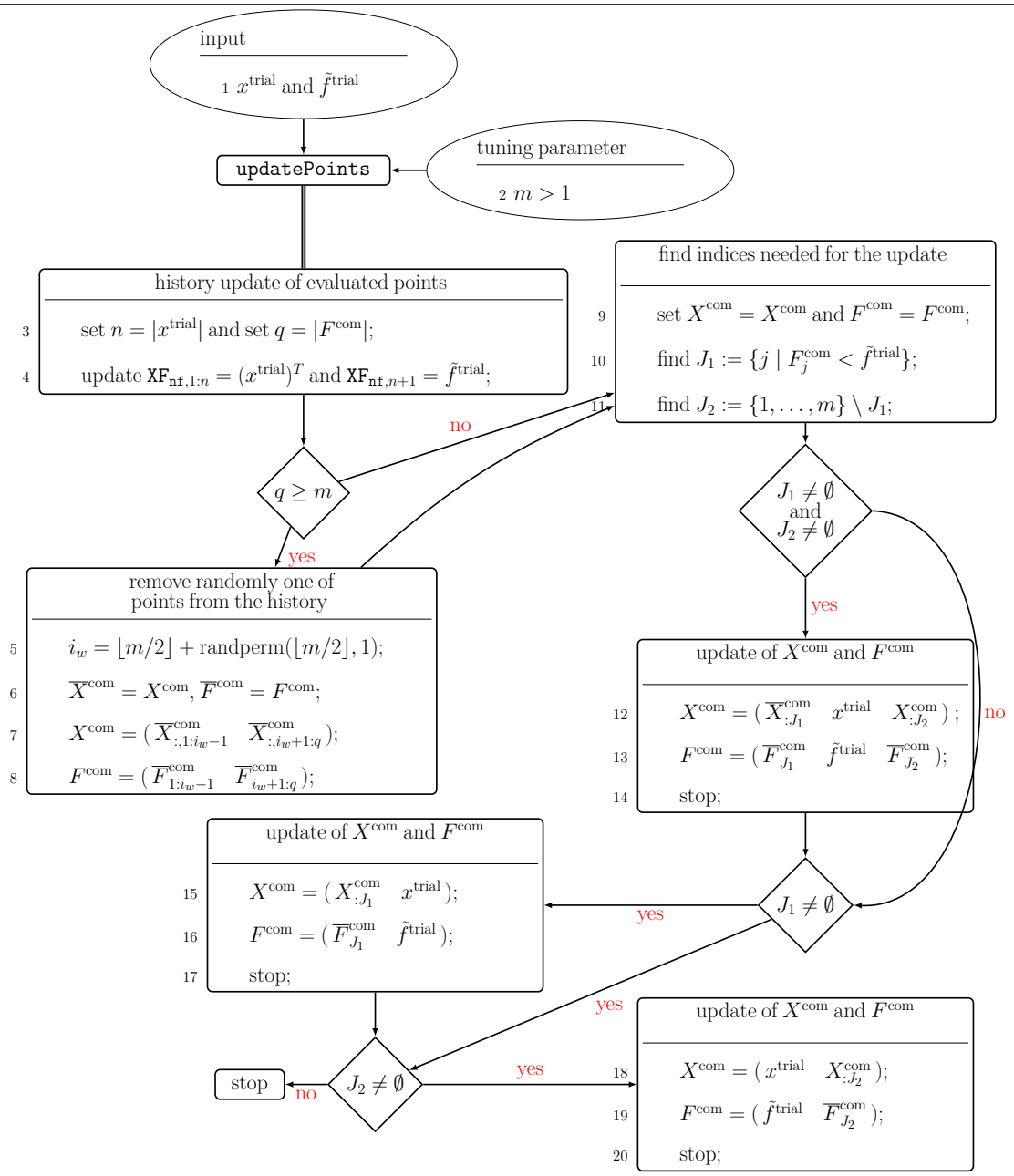
Algorithm 2 is pseudocode for `updatePoint`. `updatePoint` creates and updates the two different lists of evaluated points and their function values.

To approximate the gradients of the models in `cTRS` and `iTRS`, `updatePoint` saves any new trial point and its function value in the matrix `XF` (the first list) in line 4. To simplify pseudocode of `updatePoint` in the integer variable each trial point must be checked whether or not it has been evaluated already; this is done in the Matlab code of `updatePoint`.

To compute the combination directions in `miMATRS`, `updatePoint` saves and updates in the second list $(X)^{\text{com}}, F^{\text{com}}$ at most m best evaluated points in the matrix X^{com} and their function values in the vector F^{com} in ascending order. More precisely, `updatePoint` randomly selects an evaluated point whose place is between $m/2$ and m in line 5 by using the Matlab function randperm, removes this selected point and its inexact function value from X^{com} and F^{com} (the second list) in lines 7-8, and adds the new evaluated point and its inexact function value to X^{com} and F^{com} in lines 12-13, 15-16, 18-19, so that the ascending order of inexact function values at these points is preserved.

To simplify all algorithms, we do not mention `XF`, X^{com}, and F^{com} as input and output of each subroutine, which computes the function value. Hence, `XF`, X^{com}, F^{com}, and `nf` are persistent variables.

Algorithm 2 Pseudocode for `updatePoint`

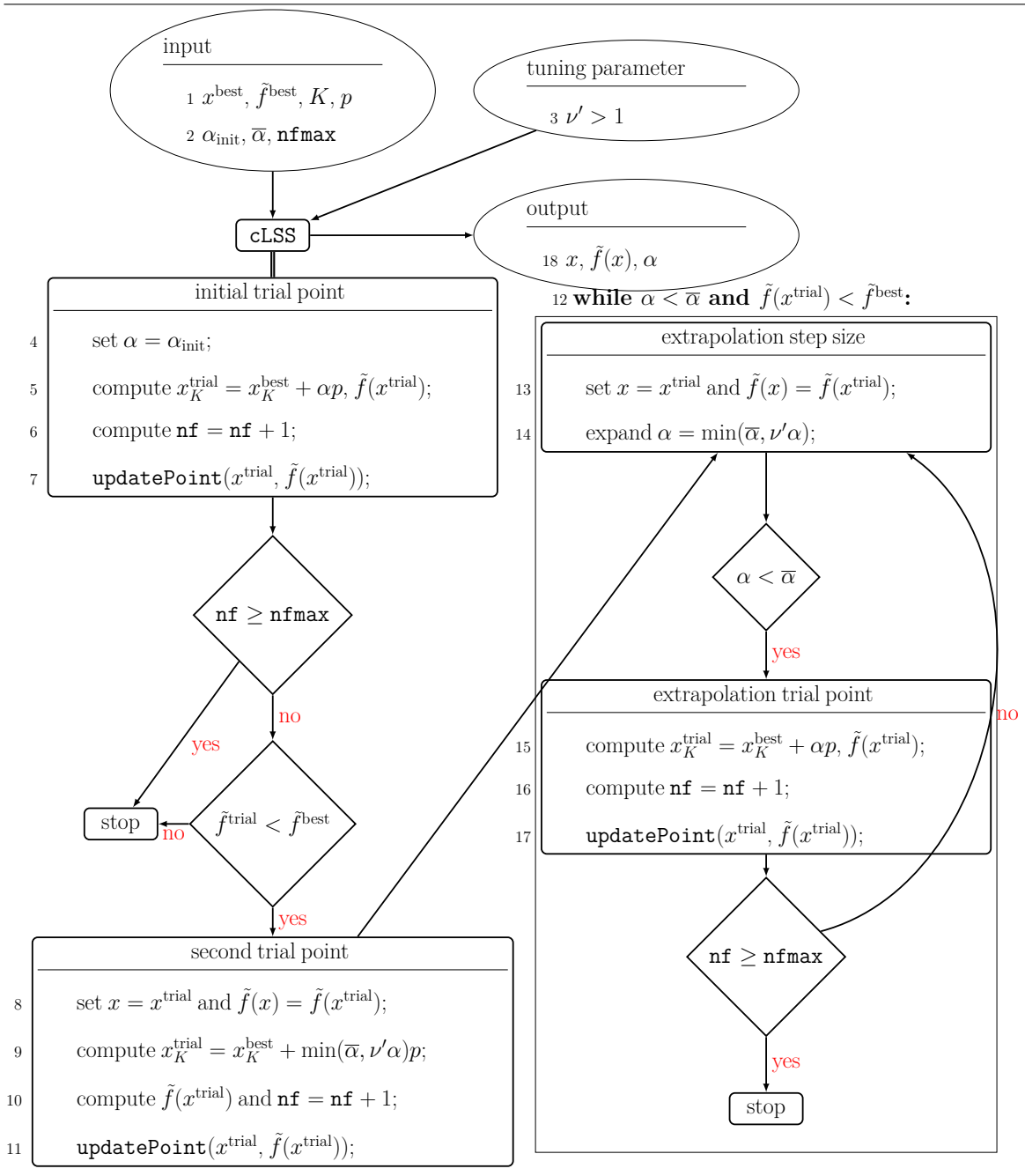


1.1.3 cLSS

Algorithm 3 is pseudocode for cLSS. In line 4 of cLSS, $\alpha = \alpha_{\text{init}}$ is chosen and the first continuous trial point x_K^{trial} and its inexact function value $\tilde{f}^{\text{trial}} := \tilde{f}(x^{\text{trial}})$ are calculated. The history of evaluated points is updated by calling updatePoint in line 7. If the descent condition $\tilde{f}^{\text{trial}} < \tilde{f}^{\text{best}}$ holds, the first trial point and its function value are saved in line 8 of cLSS and accepted as the first trial point of extrapolation. Then the new continuous trial point x_K^{trial} and its inexact function value $\tilde{f}^{\text{trial}}$ are calculated in lines 9-10 of cLSS. Otherwise, cLSS ends with the first trial point, which is accepted as either a mutation point in cMutation or a recombination point in cRecom. As long as the conditions $\alpha < \overline{\alpha}$ and $\tilde{f}^{\text{trial}} < \tilde{f}^{\text{best}}$ hold, an extrapolation step along the search direction p is continued by expanding the real step size in line 14 of cLSS and computing the new continuous trial point x_K^{trial} and its inexact function value $\tilde{f}^{\text{trial}}$ in line 15 of cLSS. Step sizes within cLSS are defined as in [8].

As in [10], after the extrapolation with at least two trial points is terminated, the trial point with the lowest inexact function values among all trial points evaluated by extrapolation is chosen as the new best point. To simplify the structure of cLSS this does not appear in pseudocode of cLSS, below. This is done in the Matlab code of cLSS. As described in lines 14 and 15 of cMutation below, the corresponding step size of the accepted trial point by extrapolation is stored in one of the components of $\mathbf{a}'$.

Algorithm 3 Pseudocode for `cLSS`

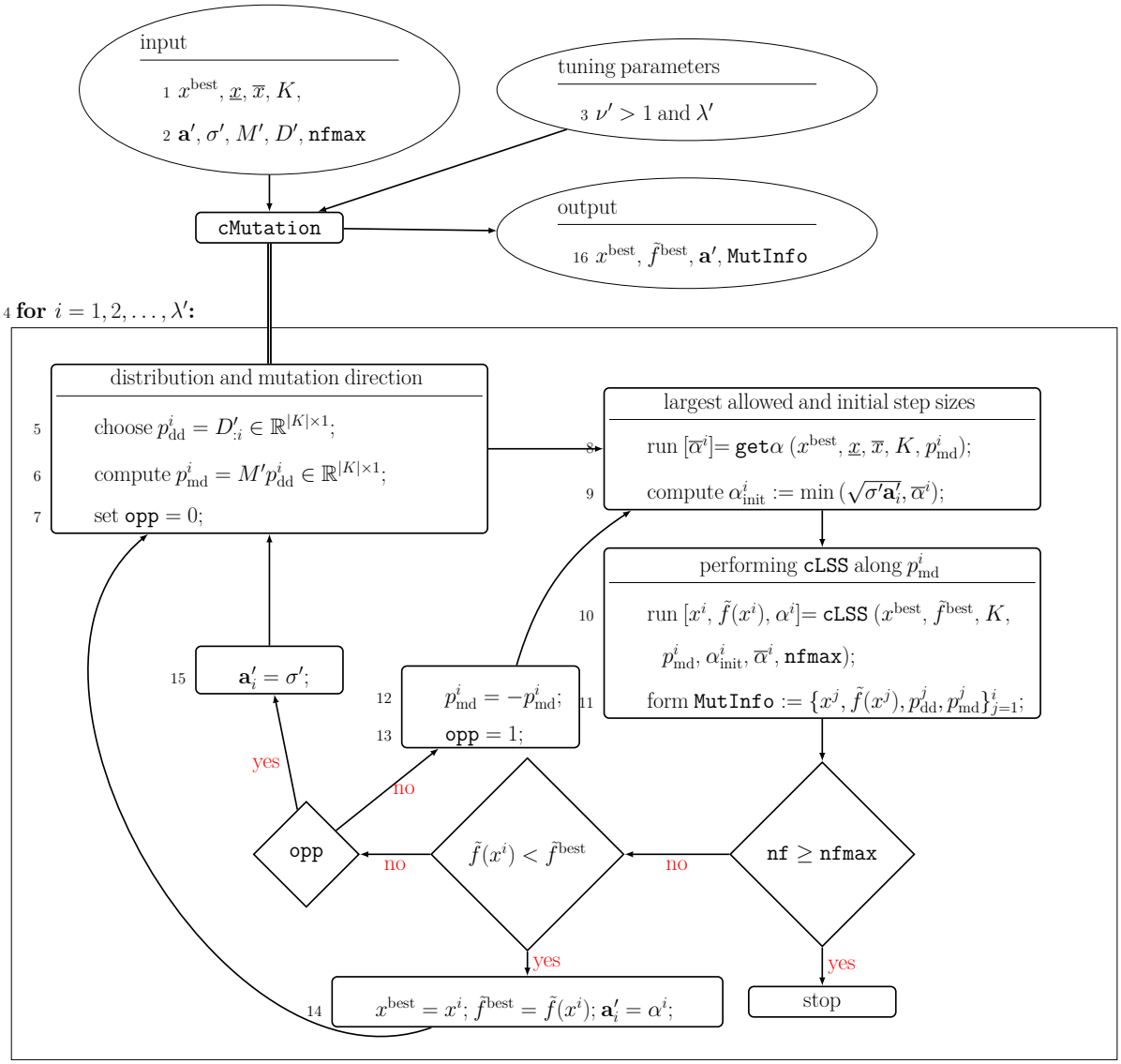


1.1.4 cMutation

Algorithm 4 is pseudocode for cMutation. The matrix $D'_{n\times\lambda'}$ denotes the set of distribution directions, each of which is chosen from the normal distribution $\mathcal{N}(0, I)$ with zero mean and variance I. In line 5 of cMutation, the ith distribution direction $p^i_{\rm dd} = D'_{:i} \in \mathbb{R}^{|K|\times 1}$ is chosen. Then, in line 6 of this algorithm, the ith mutation direction $p^i_{\rm md}$ is the product of the affine scaling matrix M' and $p^i_{\rm dd}$. Note that, in line 19 of cRecom, M' is updated by updateM. In line 7, the Boolean variable opp $= 0$ is evaluated, meaning that the opposite direction of $p^i_{\rm md}$ has not been tried yet. Before cLSS is executed, in line 8 of cMutation, the ith largest allowed real step size $\overline{\alpha}^i$ is computed by getα. In line 9 of cMutation, the ith initial real step size $\alpha^i_{\rm init}$ is chosen to be the minimum of $\overline{\alpha}^i$ and $\sqrt{\sigma'\mathbf{a}'_i}$, where σ' is the recombination step size, which is initially a positive tuning parameter and updated in lines 21-22 of cRecom below, and $\mathbf{a}'_i$ is the ith component of the mutation step size vector, which is updated in lines 14 and 15 of cMutation. The goal of this choice is to be neither too small nor too large to avoid line search failures. After computing $\alpha^i_{\rm init}$ and $\overline{\alpha}^i$, cMutation performs cLSS along the ith continuous distribution direction $p^i_{\rm md}$ to obtain the ith trial point $x^i_K = x^{\rm best}_K + \alpha^i p^i_{\rm md}$ in line 10, where α^i is found by cLSS. This trial point either is accepted (as either the continuous mutation point or the new best point) or rejected. If nf reaches nfmax, cMutation terminates. If cLSS cannot update $x^{\rm best}$ along $p^i_{\rm md}$, $p^i_{\rm md} = -p^i_{\rm md}$ is chosen in line 12 and opp $= 1$ is evaluated in line 13. After recomputing $\alpha^i_{\rm init}$ and $\overline{\alpha}^i$ in lines 8-9, respectively, cMutation performs cLSS along $p^i_{\rm md}$ and the list MutInfo is formed in line 11 of cMutation, which is used as input for selection in Section 1.2. Then, if nf reaches nfmax, cMutation terminates. Otherwise, if the descent condition $\tilde{f}(x^i) < \tilde{f}^{\rm best}$ holds, $x^{\rm best}$ and $\tilde{f}^{\rm best}$ are updated in line 14 of cMutation. In this case, x^i is one of the trial points evaluated by cLSS along $\pm p^i_{\rm md}$ with the lowest inexact function value $\tilde{f}(x^i)$ and the other trial points are rejected. If the first trial point cannot be the new best point, in this case, cLSS evaluates only one trial point and ends while accepting the first trial point as the ith mutation point.

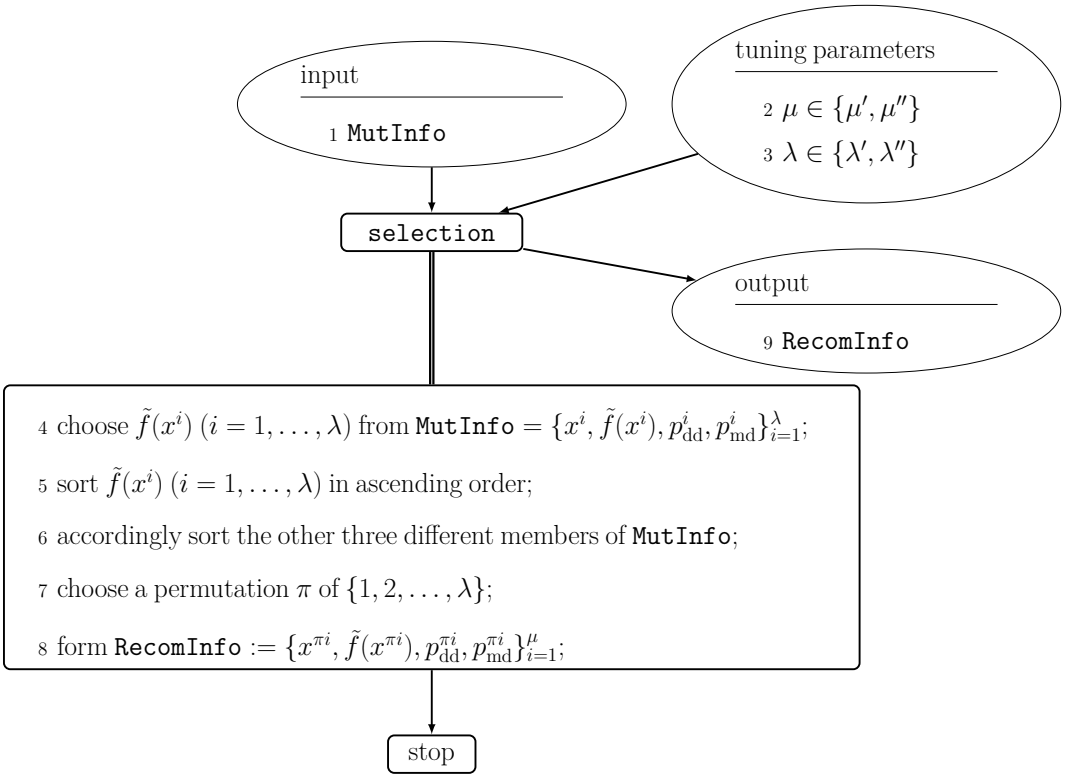
In lines 14 and 15 of cMutation, the ith component of the ith list $\mathbf{a}' \in \mathbb{R}^{\lambda'}$ of real mutation step sizes is updated, which is used to update the initial real step size $\alpha^i_{\rm init}$ for $i = 1, 2, \cdots, \lambda'$ in line 9 of cMutation. The vector $\mathbf{a}'$ is initially a tuning vector with real components ($\mathbf{a}'_i > 0$ for $i = 1, 2, \cdots, \lambda'$) and updated depending on whether or not decreases in the inexact function values are found at the trial points. After cLSS terminates to find the ith continuous mutation point, the corresponding step size of a point with the lowest inexact function value among all points evaluated in extrapolation is stored in $\mathbf{a}'_i$ in line 14 of cMutation. Otherwise, unlike [8] with $\mathbf{a}'_i = \mathbf{a}'_i/\nu'$ ($\nu' > 1$ is a given tuning parameter), $\mathbf{a}'_i = \sigma'$ is stored in line 15 of cMutation. The reason for this new choice is that σ' does not become too small, avoiding getting stuck before an approximate stationary point is found. This is a new property of our algorithm, which is against line search failures.

Algorithm 4 Pseudocode for `cMutation`



1.2 Selection

Algorithm 5 Pseudocode for `selection`



Algorithm 5 is pseudocode for `selection`. When `cMATRS` calls `selection`, $\lambda = \lambda'$ and $\mu = \mu'$ are chosen as the number of mutation points and the number of selected mutation points, respectively, but when `iMATRS` calls `selection`, $\lambda = \lambda''$ and $\mu = \mu''$ are chosen. `selection` takes the list `MutInfo` defined in line 4, where x^i $(i = 1, \ldots, \lambda)$ is the sequence of (integer and continuous) mutation points found. The sequence $\tilde{f}(x^i)$ $(i = 1, \ldots, \lambda)$ of inexact function values of the mutation points x^i $(i = 1, \ldots, \lambda)$ is sorted in ascending order

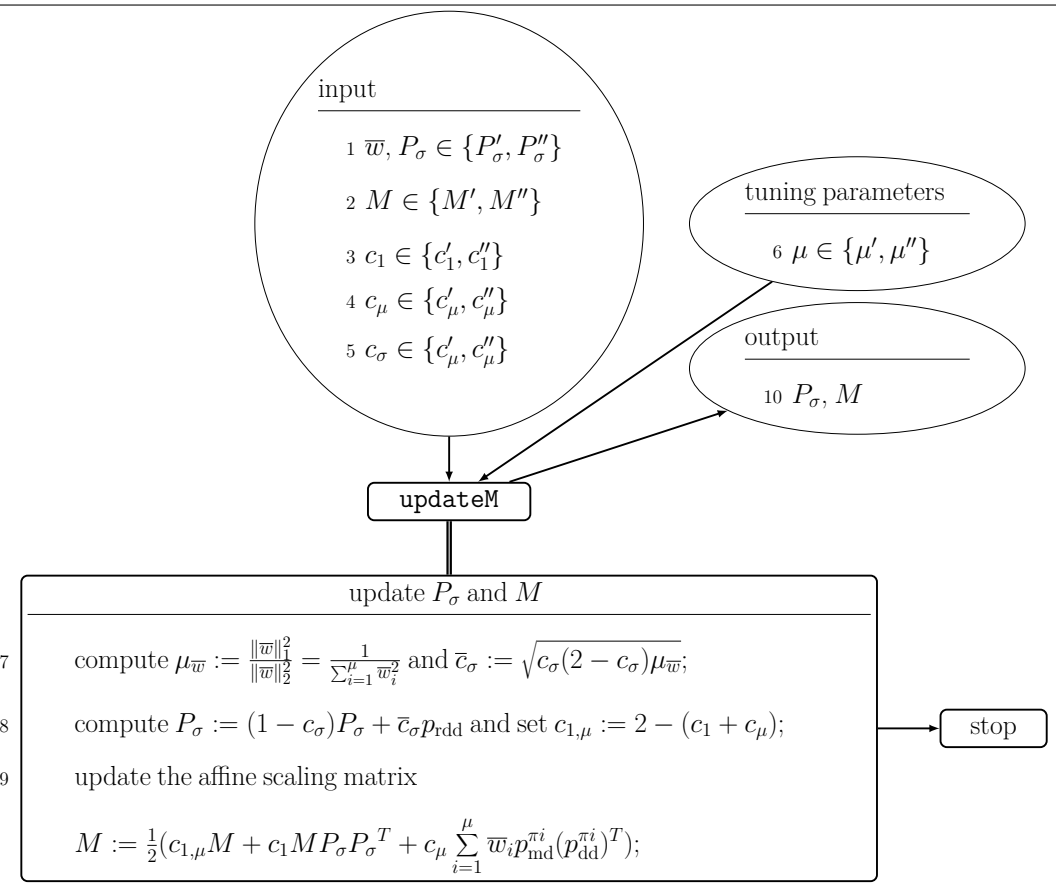
$$\tilde{f}(x^{\pi 1}) \leq \tilde{f}(x^{\pi 2}) \leq \ldots \leq \tilde{f}(x^{\pi \mu}) \leq \tilde{f}(x^{\pi(\mu+1)}) \leq \cdots \leq \tilde{f}(x^{\pi \lambda})$$

in line 5, where π is a permutation of $\{1, 2, \ldots, \lambda\}$. Then, accordingly the distribution directions $p^{\pi i}_{\mathrm{dd}}$ $(i = 1, \ldots, \lambda)$ and the mutation directions $p^{\pi i}_{\mathrm{md}}$ $(i = 1, \ldots, \lambda)$ are obtained in line 6 of `selection`. Finally, `selection` chooses π in line 7 and saves the best information in the list `RecomInfo` in line 8, which is used to compute new recombination points in the recombination phase, where μ is the number of selected points.

1.3 A continuous recombination phase

1.3.1 `updateM`

Algorithm 6 Pseudocode for `updateM`



Algorithm 6 is pseudocode for `updateM`. As can be seen from line 6 of `updateM`, for an example, $\mu = \mu'$ is chosen when `cRecom` calls `updateM`, while $\mu = \mu''$ is chosen when `iRecom` calls `updateM`. This is the same for the other parameters defined in lines 1-6. In line 7 of `updateM`, the variance effective selection mass $\mu_{\overline{w}}$ and the normalization constant $\overline{c}_\sigma$ are computed, the second which is used in line 8 of `updateM` to update the evolution path P_σ and the recombination step size in lines 20 of `Recom` below. In line 9 of `updateM`, as in [1], the affine scaling matrix M is updated, where $0 < c_\mu \leq 1$ is a learning rate for updating M and $c_1 \leq 1 - c_\mu$ is a learning rate for the rank-one-update M (Section 3 of [12] discussed the numerical formulas for c_1, c_μ, and c_σ). In this rank-one-update: (i) The first term $c_{1,\mu}M$ includes the previous information and accumulates the information. (ii) The second term $c_1 M P_\sigma P_\sigma{}^T$ is the rank-one update, whose goal is to increase the probability of $p_{\mathrm{dd}}^{\pi i}$ $(i = 1, \ldots, \mu)$ for the next iteration, by maximizing the log-likelihood of $p_{\mathrm{dd}}^{\pi i}$ $(i = 1, \ldots, \mu)$. (iii) The third term $c_\mu \sum_{i=1}^{\mu} \overline{w}_i p_{\mathrm{md}}^{\pi i}(p_{\mathrm{dd}}^{\pi i})^T$ is the rank-μ update, whose goal is to take the mean of the estimated affine scaling matrices from all iterations.

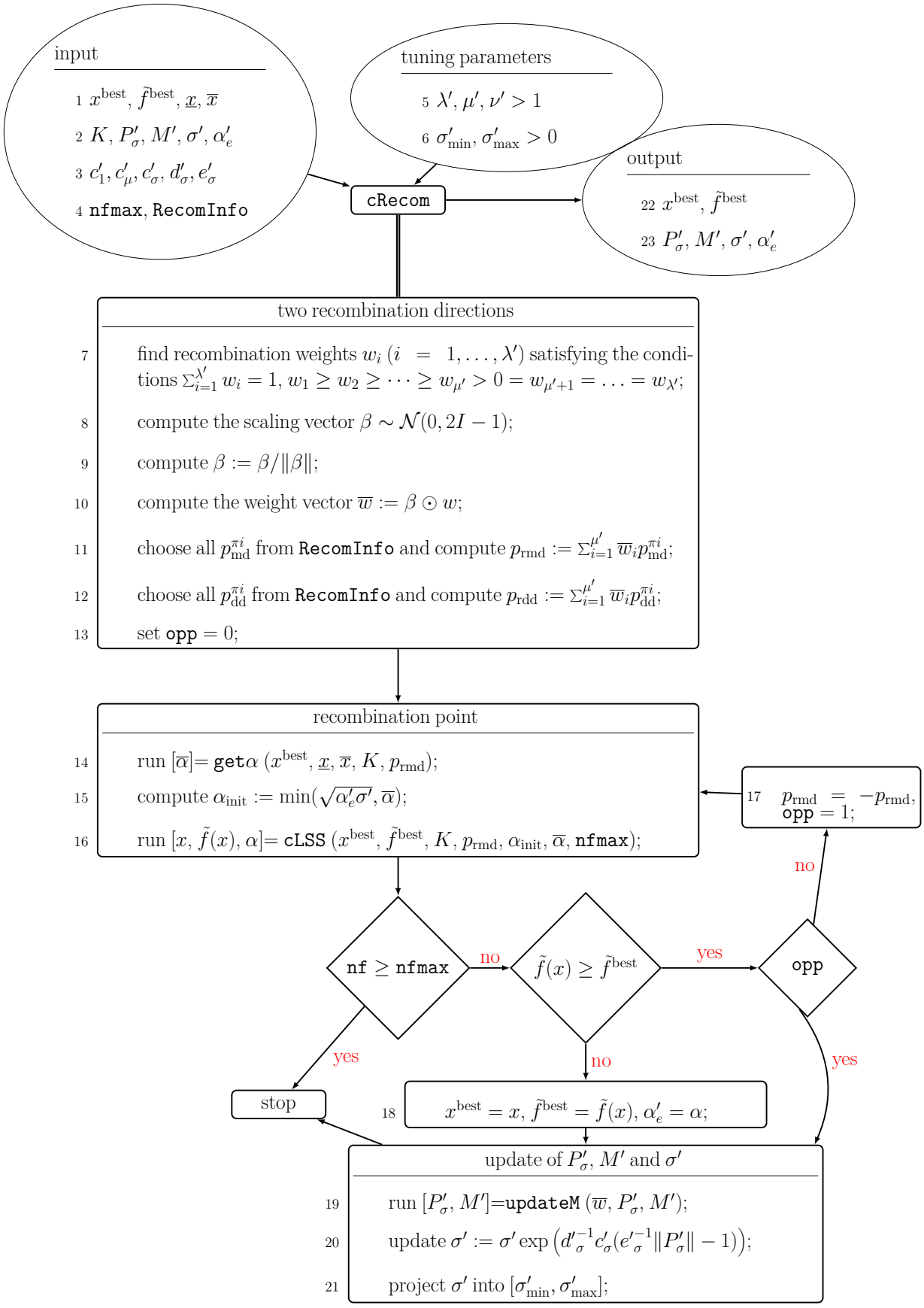
The evolution path P_σ has two goals. Its first goal is to remedy losing the sign of $p_{\mathrm{dd}}^{\pi i}$ $(i = 1, \ldots, \mu)$ in the third term of M because $p_{\mathrm{dd}}^{\pi i}(p_{\mathrm{dd}}^{\pi i})^T = -p_{\mathrm{dd}}^{\pi i}(-p_{\mathrm{dd}}^{\pi i})^T$ and $p_{\mathrm{md}}^{\pi i} = M p_{\mathrm{dd}}^{\pi i}$.

Its second goal is to update the recombination step size σ in line 20 of `cRecom`. Note that $p_{\mathrm{md}}^{\pi i}$ has been computed before in the mutation phase and here it only reuses, leading to $\mathcal{O}(n^2)$ operations due to the vector-matrix products. As a result, these three terms have different advantages and cause the affine scaling matrix behaves well in practice, compared to the rank-one update and rank-μ update.

1.3.2 `cRecom`

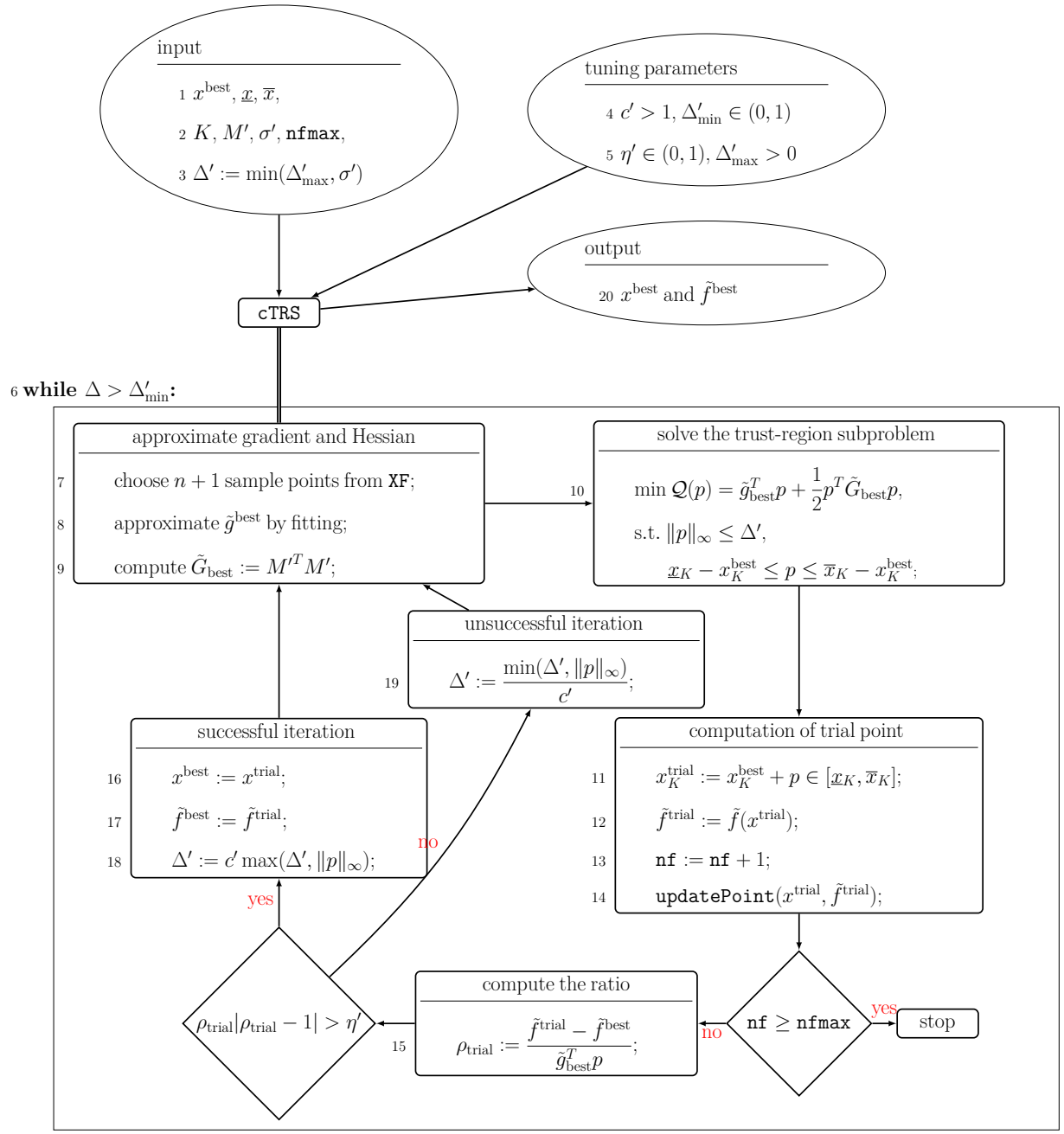
Algorithm 7 is pseudocode for `cRecom`. `cRecom` computes the weights w_i $(i = 1, \ldots, \lambda')$ in line 7 and the scaling vector β in line 8, where $\mathcal{N}(0, 2I - 1)$ is a normal distribution with zero mean and variance $2I - 1$. Then it scales the weights w_i $(i = 1, \ldots, \lambda')$ by β in line 9 for reordering a possible fair sort during the selection phase when noise is high. This is a new feature of our algorithm. In line 10, $\odot$ denotes the componentwise product. In Section 3 of [12], we computed numerically w_i for $i = 1, \ldots, \lambda'$. Given `RecomInfo` $:= \{x^{\pi i}, \tilde{f}(x^{\pi i}), p_{\mathrm{dd}}^{\pi i}, p_{\mathrm{md}}^{\pi i}\}_{i=1}^{\mu}$, in lines 11-12 of `cRecom`, the continuous recombination mutation direction p_{rmd} and the continuous recombination distribution direction p_{rdd} are computed, which are used as input for `cLSS` in line 16 of `cRecom` and to update M' in line 9 of `updateM`, respectively. They are the weighted average of the μ' mutation directions and the weighted average of the μ' distribution directions, respectively. In line 13 of `cRecom`, `opp` $= 0$ is chosen, which means the opposite direction has not been tried yet. In line 16 of `cRecom`, to avoid the generation of too small step sizes and keep feasibility, the initial real step size α_{init} is updated only based on $\overline{\alpha}$, α_e', and σ'. In line 15 of `cRecom`, `cLSS` is performed along $\pm p_{\mathrm{rmd}} \in \mathbb{R}^{|K|\times 1}$ to update x^{best} by extrapolation. `cRecom` terminates if `nf` reaches `nfmax`. If x^{best} cannot be updated, `cRecom` sets $p_{\mathrm{rmd}} = -p_{\mathrm{rmd}}$ in line 17, evaluates `opp` $= 1$, and computes α_{init} and $\overline{\alpha}$ in lines 14-15 and reruns `cLSS` along p_{rmd} to update x^{best}. Then, `cRecom` terminates if `nf` reaches `nfmax`. If $f(x) < \tilde{f}^{\mathrm{best}}$ holds, x, which is one of the trial points evaluated by extrapolation with the lowest inexact function value among all evaluated trial points, is accepted as the new best point in line 18 of `cRecom`. In lines 20-21 of `cRecom`, the real recombination step σ' is computed, where e_σ' is an approximate value of the expected value $\mathbf{E}(\|u\|)$ of the norm of the vector $u \sim \mathcal{N}(0, I)$, the constant $0 < \sigma'_{\mathrm{max}} < \infty$ is a maximum value for σ', $0 < \sigma'_{\mathrm{min}} < 1$ is a minimum value for σ', $c_\sigma' \leq 1$ is a learning rate for the cumulation for the step size, $d_\sigma' \approx 1$ is a damping parameter (cf. [5, Section 4]), see Section 3 of [12] the numerical formulas for d_σ', c_σ', and e_σ'. Moreover, as long as there is no feasible trial point along $\pm p_{\mathrm{rmd}}$, at most n'_{scale} times p_{rmd} rescales by $p_{\mathrm{rmd}} = \beta \odot p_{\mathrm{rmd}}$. Here n'_{scale} is a tuning parameter. This improvement can be found in the Matlab code of `cRecom`.

Algorithm 7 Pseudocode for `cRecom`



1.4 cTRS

Algorithm 8 Pseudocode for cTRS



Algorithm 8 is pseudocode for `cTRS`. In line 3 of `cTRS`, $\Delta' > 0$ initially is chosen, where σ' is computed in line 21 of `cRecom` and $0 < \Delta'_{\max} < \infty$ is a tuning parameter. When the objective function value at each trial point of line search and trust-region strategies is computed, the function values and the corresponding points are saved in `XF`, whose $n+1$ current points and their function values are used to approximate $\tilde{g}_{\text{best}}$ in line 7 of `cTRS`. Then to form the trust-region subproblem, its approximate gradient vector $\tilde{g}_{\text{best}}$ is obtained by fitting in line 8 of `cTRS` as in Huyer & Neumaier [7] and its approximate symmetric Hessian matrix $\tilde{G}_{\text{best}} = {M'}^T M'$ is chosen in line 9 of `cTRS` as a new choice without additional cost. Afterwards, the trust-region subproblem defined in line 10 of `cTRS` is solved by `minq8` by Huyer & Neumaier [6]. After solving the trust-region subproblem, the continuous trial point x_K^{trial} and its inexact function value $\tilde{f}^{\text{trial}} := \tilde{f}(x^{\text{trial}})$ are computed in lines 11-12 of `cTRS` and the two different histories of points are updated in line 14 by `updatePoint`. If `nf` reaches `nfmax`, `cTRS` terminates. Given the tuning parameter $0 < \eta' < \frac{1}{4}$, if the sufficient descent condition $\rho_{\text{trial}}|\rho_{\text{trial}} - 1| > \eta'$ (suggested by Kimiaei [9] for bound-constrained ill-conditioned problems) is satisfied, the current iteration of `cTRS` is called **successful** and x^{trial} is accepted as the new best point. Then, using the tuning parameter $c' > 1$, we expend Δ' in line 18 of `cTRS` by the traditional formula of Conn et al. [3]. Otherwise, the current iteration of `cTRS` is called **unsuccessful**. In this case, Δ' is reduced in line 19 of `cTRS`.

1.5 cMATRS

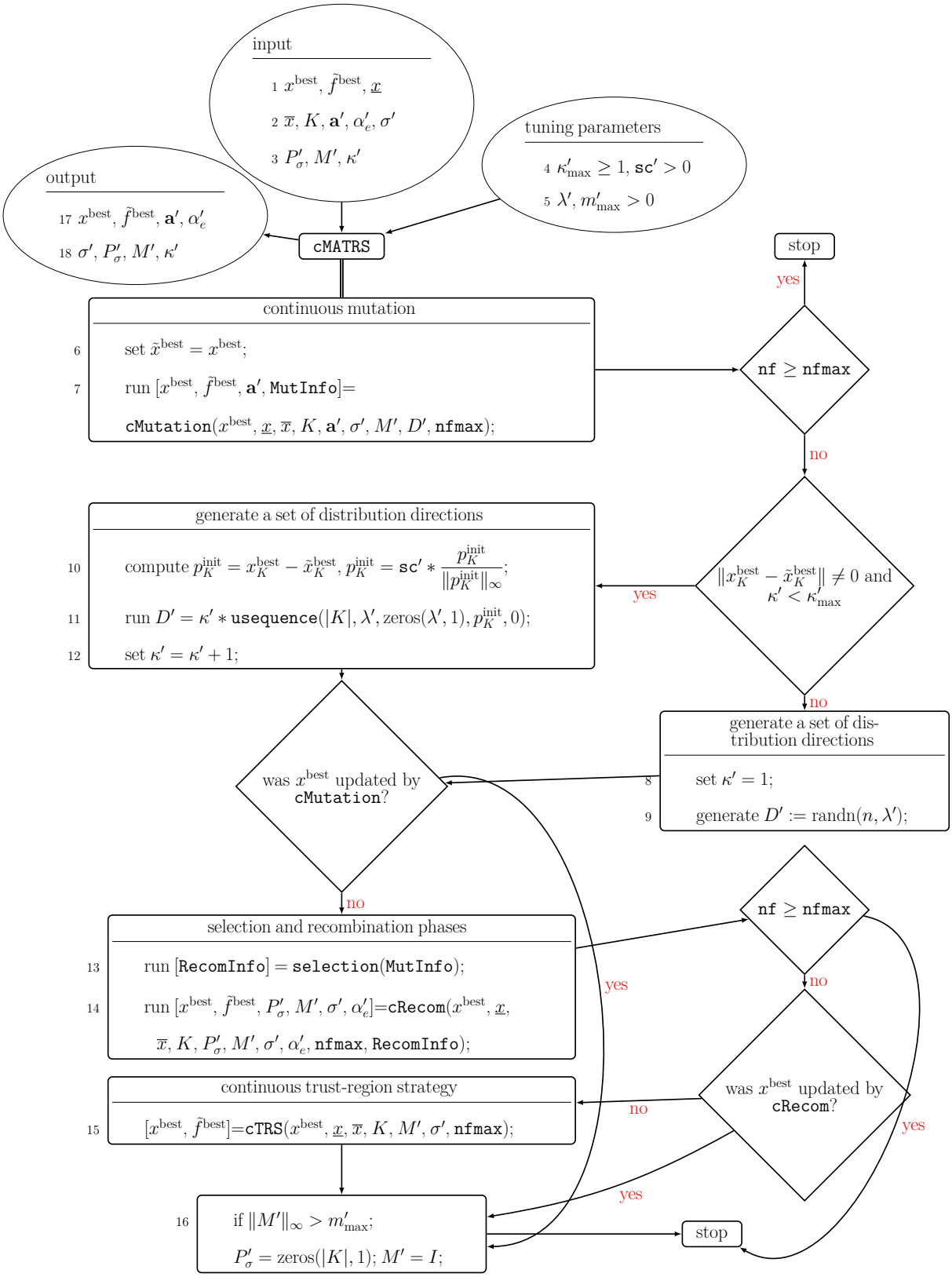
Algorithm 9 is pseudocode for `cMATRS`. The variable κ' is the counter for the number of times that the set D' of distribution directions must be generated by `usequence`. If `cMutation` cannot update x^{best} and κ' does not reach its upper bound $\kappa'_{\max} > 1$: (i) `cMATRS` computes the new combination direction p_K^{init} in line 10, where $\tilde{x}^{\text{best}}$ is the old best point and x^{best} is the current best point found and `sc`$'$ is a positive tuning parameter. For the next call to `cMutation`, there is a good chance to find decreases in the inexact function value by performing `cLSS` along at least one of $\pm p_K^{\text{init}}$, leaving points with large inexact function values and going into or moving down a valley. (ii) The set D' of random directions is regenerated in a new randomized way by `usequence` in line 11 of `cMATRS` with the goal of finding the new best point in a larger neighborhood of the old best point. Here κ' is updated in each iteration of `cMATRS` in lines 8 and 12. Otherwise, it sets $\kappa' = 1$ and distribution directions are selected from $\mathcal{N}(0, I)$ with zero mean and variance I.

In line 16 of `cMATRS`, if $\|M'\|_\infty$ is greater than a positive tuning parameter $m'_{\max}$, both the evolution path P'_σ and the affine scaling matrix M' are replaced by a zeros vector and an identity matrix, respectively, since large steps in `cMutation` and `cRecom` are one of the causes for line search failure.

1.6 An integer mutation phase

Pseudocode for `iMutation` is not introduced here because it shares the same structure as `cMutation`. Rather, here are some distinctions between `iMutation` and `cMutation`. They differ in how p^i_{dd} and p^i_{md} are computed and how $\overline{\alpha}^i$ and α^i_{init} are determined.

Algorithm 9 Pseudocode for `cMATRS`



`iMutation` replaces lines 5 and 6 of `cMutation` by

$$p^i_{\mathrm{dd}} = D''_{:i} \in \mathbb{R}^{|I|\times 1}, \quad p^i_{\mathrm{md}} = \lceil M'' p^i_{\mathrm{dd}} \rfloor \in \mathbb{R}^{|I|\times 1}.$$

Here, integer distribution directions $p^i_{\mathrm{dd}} \in \mathbb{R}^{|I|\times 1}$ $(i = 1, \ldots, \lambda'')$ are chosen from a set D'' of integer directions, unlike `cMutation`, which selects continuous distribution directions from the normal distribution (D'' is initially a tuning matrix and it is updated in `iMATRS` below). Then integer mutation directions $p^i_{\mathrm{md}} \in \mathbb{R}^{|I|\times 1}$ are computed by rounding the product of the affine scaling matrix M'' and p^i_{dd} for $i = 1, \ldots, \lambda''$. Next, `iMutation` replaces lines 8 and 9 of `cMutation` by

$$[\overline{\alpha}^i] = \texttt{get}\alpha\ (x^{\mathrm{best}},\ \underline{x},\ \overline{x},\ I,\ p^i_{\mathrm{md}}),\ \overline{\alpha}^i := \max(1, \lfloor \overline{\alpha}^i \rfloor),$$

$$\alpha^i_{\mathrm{init}} := \min\Big(\Big\lfloor \sqrt{\sigma'' \mathbf{a}''_i} \Big\rceil, \max(1, \lfloor \overline{\alpha}^i \rfloor)\Big)$$

and line 10 of `cMutation` by

$$[x^i,\ \tilde{f}(x^i),\ \alpha^i] = \texttt{iLSS}\ (x^{\mathrm{best}},\ \tilde{f}^{\mathrm{best}},\ I,\ p^i_{\mathrm{md}},\ \alpha^i_{\mathrm{init}},\ \overline{\alpha}^i,\ \texttt{nfmax}).$$

`iLSS` and `cLSS` share the same structure although they differ in a few details. `cLSS` searches in the space of x_K, while `iLSS` searches in the space of x_I. In addition, their input (p^i_{md}, α^i_{init}, $\overline{\alpha}^i$) have been computed differently.

1.7 `iRecom`

Since `iRecom` has the same structure as `cRecom`, its pseudocode is not covered here. Instead, some differences between `cRecom` and `iRecom` are listed below. There are differences in the calculations of p_{rmd}, $\overline{\alpha}$, and α_{init} between them.

`iRecom` preserves line 11 of `cRecom`, but non-integer components of p_{rmd} (if any) are rounded to integer and replaces lines 14-16 of `cRecom`, respectively, by

$$[\overline{\alpha}] = \texttt{get}\alpha\ (x^{\mathrm{best}},\ \underline{x},\ \overline{x},\ I,\ p_{\mathrm{rmd}}),\ \overline{\alpha} := \max(1, \lfloor \overline{\alpha} \rfloor),\ \alpha_{\mathrm{init}} := \min(\lfloor \sqrt{\alpha''_e \sigma''} \rceil, \overline{\alpha}),$$

$$[x,\ \tilde{f}(x),\ \alpha] = \texttt{iLSS}\ (x^{\mathrm{best}},\ \tilde{f}^{\mathrm{best}},\ I,\ p_{\mathrm{rmd}},\ \alpha_{\mathrm{init}},\ \overline{\alpha},\ \texttt{nfmax}).$$

Next, `iRecom` changes lines 19-21 of `iRecom` to

$$[P''_\sigma,\ M''] = \texttt{updateM}\ (\overline{w},\ P''_\sigma,\ M''),$$

$$\sigma'' := \left\lfloor \sigma'' \exp\Big(\frac{c''_\sigma}{d''_\sigma}\Big(\frac{\|P''_\sigma\|}{e''_\sigma} - 1\Big)\Big) \right\rfloor \in [\sigma''_{\min}, \sigma''_{\max}].$$

The affine scaling matrix M'' is computed by `updateM` without rounding its entries to integer. In the computation of integer mutation directions non-integer entries of these directions are rounded to integers. As in the continuous case, the real step size is computed, but rounded to integer. Moreover, if there is no feasible trial point along $\pm p_{\mathrm{rmd}}$, at most n''_{scale} times $p_{\mathrm{rmd}} := \lfloor \beta \odot p_{\mathrm{rmd}} \rceil$ is recomputed. Here n''_{scale} is a tuning parameter.

1.8 iTRS

Since iTRS and cTRS have the same structure, pseudocode for iTRS is not introduced here. For iTRS and cTRS, the trust-region subproblem is solved differently, and the trust-region radius is updated differently as well.

iTRS chooses the initial integer radius $\Delta'' := \sigma'' \in [\Delta''_{\min}, \Delta''_{\max}]$. The two tuning parameters $\Delta''_{\max} > \Delta''_{\min} \geq 1$ control Δ'', so that it is neither too small or too large. Then, iTRS transforms the trust-region subproblem

$$\begin{array}{ll} \min & \mathcal{Q}(p) = \tilde{g}_{\text{best}}^T p + \dfrac{1}{2} p^T \tilde{G}_{\text{best}} p \\ \text{s.t.} & \|p\| \leq \Delta'', \quad p \text{ integral}, \\ & x_I^{\text{best}} + p \in [\underline{x}_I, \overline{x}_I] \end{array}$$

into the bound-constrained integer least squares problem

$$\begin{array}{ll} \min & \frac{1}{2}\|M'' p - r\|_2^2 \\ \text{s.t.} & \|p\| \leq \Delta'', \quad p \text{ integral}, \\ & x_I^{\text{best}} + p \in [\underline{x}_I, \overline{x}_I] \end{array}$$

by choosing $r := -M''^{-T} \tilde{g}_{\text{best}}$ and setting

$$\tilde{g}_{\text{best}}^T p + \frac{1}{2} p^T \tilde{G}_{\text{best}} p = \frac{1}{2}\|M'' p - r\|_2^2 - \frac{1}{2}\|r\|^2. \tag{1}$$

This least squares problem is solved by a variant of Schnorr–Euchner search [2, 4]. If the approximate solution of such a problem is zero, it is replaced by

$$p = x^{\text{best}} - x_1, \quad p = \Delta'' \big\lfloor p/\|p\| \big\rceil,$$

where x_1 is the first sample point used for computing $\tilde{g}_{\text{best}}$. In (1), the approximate gradient vector $\tilde{g}_{\text{best}}$ is obtained by fitting as in Huyer & Neumaier [7] and the approximate symmetric Hessian matrix $\tilde{G}_{\text{best}} = M''^T M''$ is chosen without additional cost. The objective function value at each trial point of line search and trust region is computed and saved in a list whose $n+1$ current points and their function values are used to approximate $\tilde{g}_{\text{best}}$.

Given the integer tuning parameters $\overline{\Delta} > 1$ and $1 < c'' < \infty$, iTRS updates the trust-region radius for unsuccessful iterations by

$$\Delta'' = \begin{cases} \lfloor \min(\|p\|_\infty, \Delta'')/c'' \rfloor & \text{if } \Delta'' \leq \overline{\Delta}, \\ \Delta'' - 1 & \text{otherwise.} \end{cases}$$

Here the traditional formula $\Delta'' = \min(\|p\|_\infty, \Delta'')/c''$ of Conn et al. [3] is used to quickly reduce the trust-region radius to take advantage of small steps and increase the accuracy of the model function. The new choice $\Delta'' = \Delta'' - 1$ has the same goal as the traditional formula, but it reduces slowly the trust-region radius, which is not now large, and therefore iTRS may try many trial feasible points to update the best point before the radius becomes

one. With the same goal as unsuccessful iterations, `iTRS` updates the trust-region radius for successful iterations by

$$\Delta'' = \begin{cases} \lfloor c'' \min(\|p\|_\infty, \Delta'') \rfloor & \text{if } \Delta'' \geq \overline{\Delta}, \\ \Delta'' + 1 & \text{otherwise.} \end{cases}$$

If an integer trust-region method cannot find new different feasible trial points, it terminates without updating the best point. Hence, `iTRS` carries out attempts to find feasible trial points that differs from the evaluated points in one of the following two ways: (i) Different sample points can be chosen randomly from the list of previous evaluated points to differently approximate the gradient of the trust-region subproblem. (ii) Different trust-region radii are generated in a new randomized way.

We first randomly select $n+1$ points from the list of stored evaluated points to compute g_{best}. If a new integer feasible point cannot be found by this change, to update the trust-region radius we randomly use at most `stuckmax` one of the two formulas

$$\Delta'' = \lfloor \Delta''/\zeta \rceil \;\; \text{with } \zeta = \text{randi}([1, \Delta''_{\min}], 1)$$

and

$$\Delta'' = |\Delta'' + \text{sign}(\text{rand} - 0.5)\zeta| \;\; \text{with } \zeta = \text{randi}([1, \Delta''_{\min}], 1)$$

until $\Delta'' \geq 1$. Here `stuckmax` ≥ 1 is a tuning parameter, $\Delta''_{\min} > 1$ is an integer tuning parameter, $\text{randi}([a_i, b_i], n, 1)$ generates an integer random value that is independent and uniformly distributed within $[a_i, b_i]$, and rand is a real random value that is independent, and uniformly distributed in $[0, 1]$.

1.9 `iMATRS`

Pseudocode for `iMATRS` is not discussed here because it shares the same structure as `cMATRS`. `iMATRS` and `cMATRS` differ in how they generate the set of distribution directions, solve trust-region subproblems, and update trust-region radii and line search step sizes.

`iMATRS` replaces line 7 of `cMATRS` by

$$[x^{\text{best}},\, \tilde{f}^{\text{best}},\, \mathbf{a}'',\, \texttt{MutInfo}] \;\; = \;\; \texttt{iMutation}(x^{\text{best}},\, \underline{x},\, \overline{x},\, I,\, \mathbf{a}'',\, \sigma'',\, M'',\, D'',\, \texttt{nfmax})$$

and then line 8 of `cMATRS` by $\kappa'' = 1$. It calls `igeneratorD` to compute the set D'' of integer distribution directions instead of lines 9 of `cMATRS`. For further information, refer to the Matlab code of `igeneratorD` for D'', which is one of three sets: a set of permuted coordinate directions, a set generated by `usequence`, and a combination of them. If `iMutation` cannot update x^{best} and κ'' does not reach its upper bound $\kappa''_{\max} > 1$, `iMATRS` replaces lines 10-12 of `cMATRS` by

$$p_I^{\text{init}} = x_I^{\text{best}} - \tilde{x}_I^{\text{best}}, \;\; p_I^{\text{init}} = \texttt{sc}'' * \Big\lceil \frac{p_I^{\text{init}}}{\|p_I^{\text{init}}\|_\infty} \Big\rceil,$$

$$D'' = \texttt{usequence}(|I|, \lambda'', \kappa'' \,\text{ones}(\lambda'', 1), p_I^{\text{init}}, 0), \;\; \kappa'' = \kappa'' + 1.$$

`iMATRS` computes the new combination direction p_I^{init}, where $\tilde{x}^{\text{best}}$ is the old best point and x^{best} is the current best point found and $\texttt{sc}''$ is a positive tuning parameter. For the next

call to `iMutation`, there is a good chance to find decreases in the inexact function value by performing `iLSS` along at least one of $\pm p_I^{\text{init}}$, leaving points with large inexact function values and going into or moving down a valley. Next, `iMATRS` replaces line 14 of `cMATRS` by

$$[x^{\text{best}}, \tilde{f}^{\text{best}}, P''_\sigma, M'', \sigma'', \alpha''_e] = \texttt{iRecom}(x^{\text{best}}, \tilde{f}^{\text{best}}, \underline{x}, \overline{x}, I, P''_\sigma, M'', \sigma'', \alpha''_e, \texttt{nfmax}, \texttt{RecomInfo})$$

and line 15 of `cMATRS` by

$$[x^{\text{best}}, \tilde{f}^{\text{best}}] = \texttt{iTRS}(x^{\text{best}}, \underline{x}, \overline{x}, I, M'', \sigma'', \texttt{nfmax}).$$

Large steps in `iMutation` and `iRecom` are one of the causes for line search failure. To avoid these, `iMATRS` changes line 16 of `cMATRS` by replacing the evolution path P''_σ by a zeros vector and the affine scaling matrix M'' by an identity matrix if $\|M''\|_\infty$ is greater than a positive tuning parameter $m''_{\max}$.

1.10 A mixed-integer `MATRS`

1.10.1 `miLSS`

Algorithm 10 is pseudocode for `miLSS`. In line 5 of `miLSS`, the initial real and integer step sizes are chosen and in lines 6-8 of `miLSS` the first trial point in the space of all x and its function value is computed. Afterwards, if the second trial point and its function value can be computed in lines 10-13, then extrapolation at least in one of spaces of all x, x_K, and x_I is performed. After computing each trial point, if `nf` reaches `nfmax`, `miLSS` ends. If $\alpha' < \overline{\alpha}'$ and $\alpha'' < \overline{\alpha}''$ hold, `miLSS` does not reduce to `cLSS` or `iLSS`. Otherwise, if $\alpha' < \overline{\alpha}'$ holds, `miLSS` is converted to `cLSS`, and if $\alpha'' < \overline{\alpha}''$ holds, `miLSS` is converted to `iLSS`. After computing each trial point and its function value in lines 9, 14, 21, `updatePoint` is performed to update the two histories of points.

Extrapolation evaluates at least one more trial point, one of which with the lowest inexact function value is accepted as the new best point. To simplify `miLSS`, this case does not appear in its pseudocode.

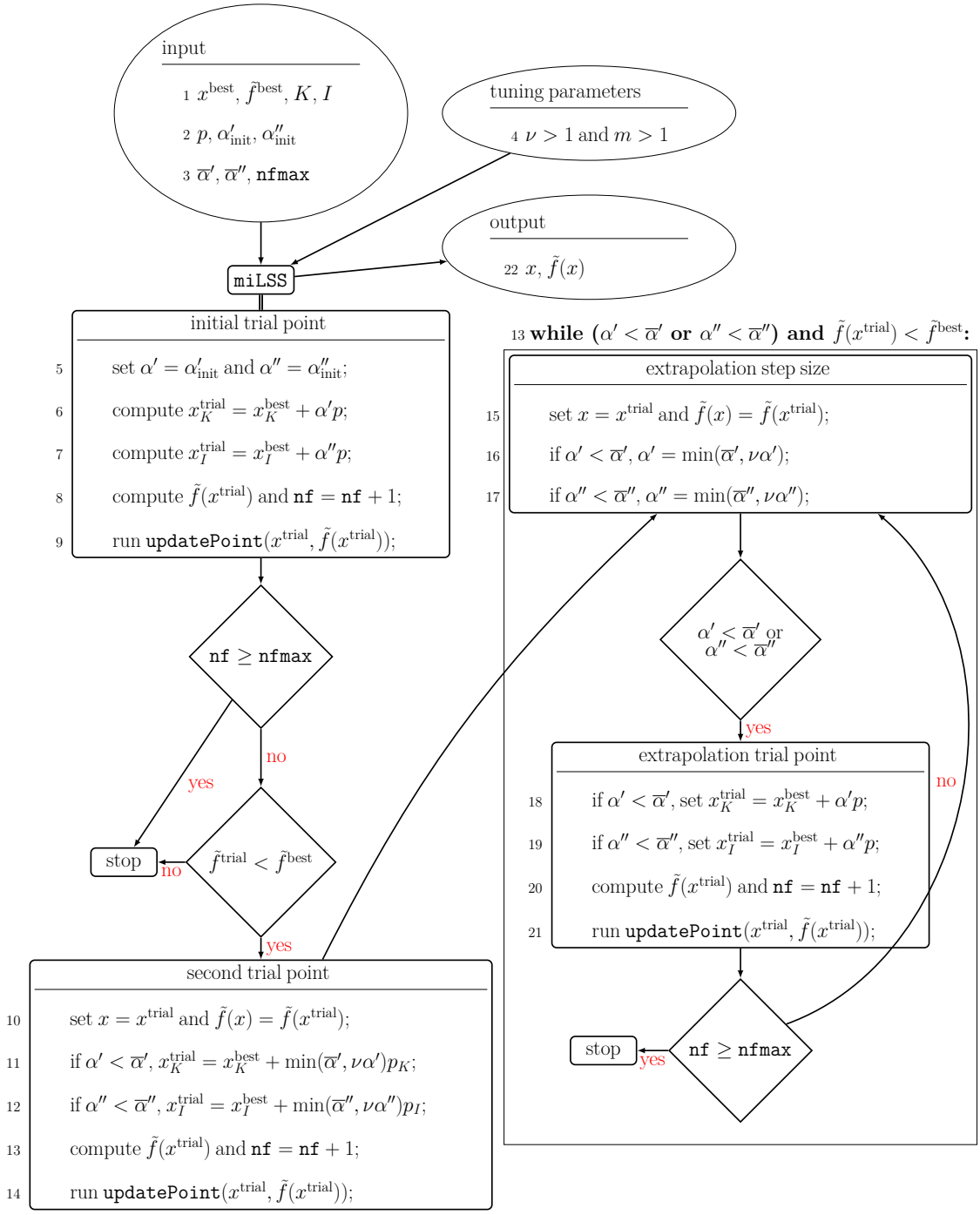
1.10.2 `miMATRS`

Algorithm 11 is pseudocode for `miMATRS`. To compute the first and (possibly) second combination directions, in line 9 of `miMATRS`, the scaling vector `sc` is computed by

$$\begin{aligned} dX_{:i} &= X_{:i} - x^{\text{best}}, \quad \text{for } i = 1, \ldots, m, \quad \texttt{sc} = |\sup(dX)|, \\ \texttt{sc}_j &= 1, \quad \text{for } j \in J = \{j \mid \texttt{sc}_j = 0\}, \\ \texttt{sc}_j &= \max(1, \lceil 1/\texttt{sc}_j \rceil), \quad \text{for } j \in I, \quad \texttt{sc}_j = \min(1, 1/\texttt{sc}_j), \quad \text{for } j \in K. \end{aligned}$$

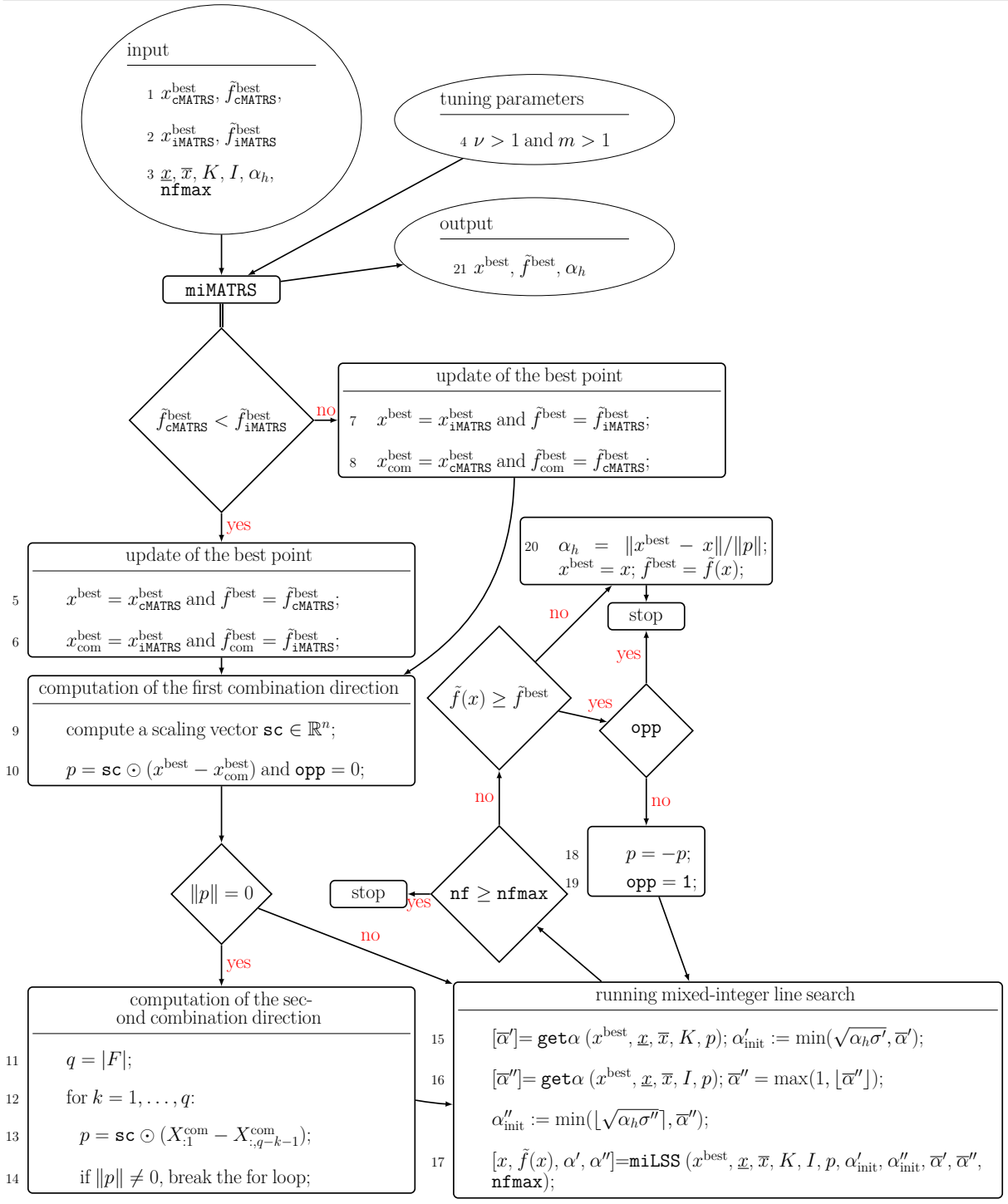
Here the tuning parameter m is the number of points used for the computation of combination directions. Then, in line 10, the product of `sc` and the difference p of the two best points found by `cMATRS` and `iMATRS` is computed and chosen as the first combination

Algorithm 10 Pseudocode for `miLSS`



direction. In this case, at least one of `cMATRS` and `iMATRS` can update x^{best}. The Boolean variable `opp` $= 0$ is evaluated, which means `miLSS` has not been performed along the first combination direction yet. If $\|p\| = 0$, it means that both `cMATRS` and `iMATRS` could not update x^{best}. In this case, in line 13 of `miMATRS` the second combination direction is computed, which is the product of `sc` and the difference p of the best point and the worst point saved in `XF`. Then, `miMATRS` performs `miLSS` along this direction or its opposite direction in line 17 after the initial real and integer step sizes α'_{init} and α''_{init} and the largest real and integer allowed step size $\overline{\alpha}'$ and $\overline{\alpha}''$ are found as in lines 15-16 of `cRecom` and `iRecom`, respectively, but with the difference that the heuristic step size α_h is used instead of α'_e and α''_e. The reason for this difference is that in practice finding α'_e and α''_e is difficult since extrapolation may be at least in one of the spaces of all x, x_I, and x_K.

Algorithm 11 Pseudocode for `miMATRS`



2 Results for `global`, `bcp`, and `prince`

stopping test: $q_f \leq 0.0001$, `sec` ≤ 360, `nf` $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
215 of 216 `global` problems solved						
dim∈[1,30]		# of anomalies			`eff`%	
solver	solved	#n	#t	#f	`nf`	`sec`
`MATRS`	201	15	0	0	61	65
`CMAES`	192	24	0	0	20	32
`NOMAD`	182	0	0	34	57	50
`BFO`	181	0	0	35	24	53
`BCDFO`	98	4	2	112	30	19
`DFOTR`	49	157	4	6	10	8

Table 1: Tabulated results for `global` for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

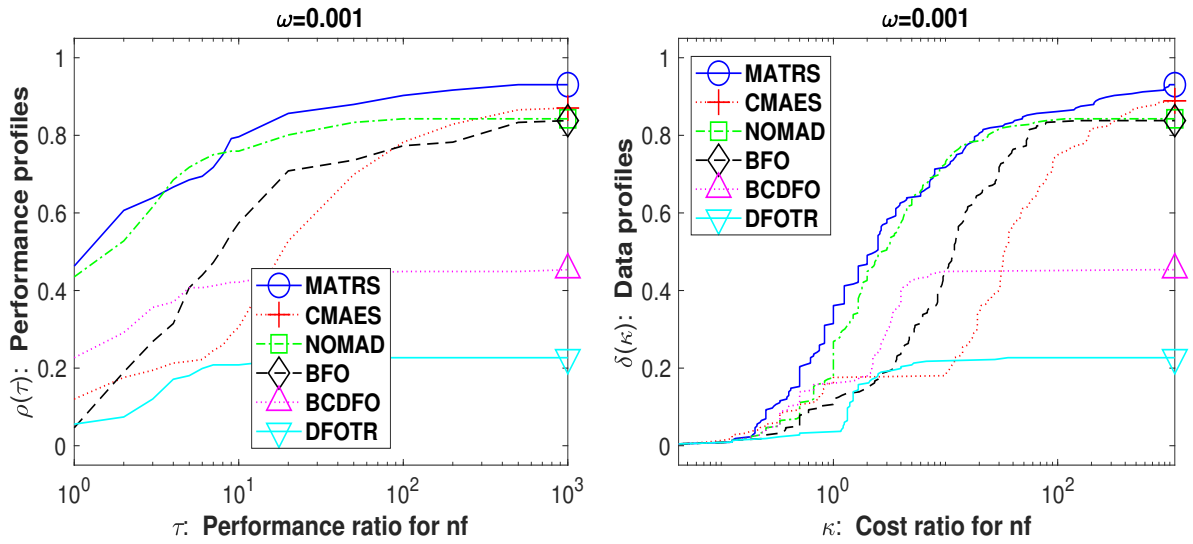


Figure 1: Plots for `global` for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Performance profiles $\rho(\tau)$ (first row) are in dependence of a bound τ on the performance ratio, while data profiles $\delta(\kappa)$ (second row) are in dependence of a bound κ on the cost ratio. Problems solved by no solver are ignored.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
212 of 216 global problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	200	16	0	0	61	66
CMAES	184	32	0	0	19	31
BFO	178	0	0	38	26	52
NOMAD	176	0	0	40	56	46
BCDFO	96	10	5	105	29	19
DFOTR	42	166	3	5	8	8

Table 2: Tabulated results for global for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

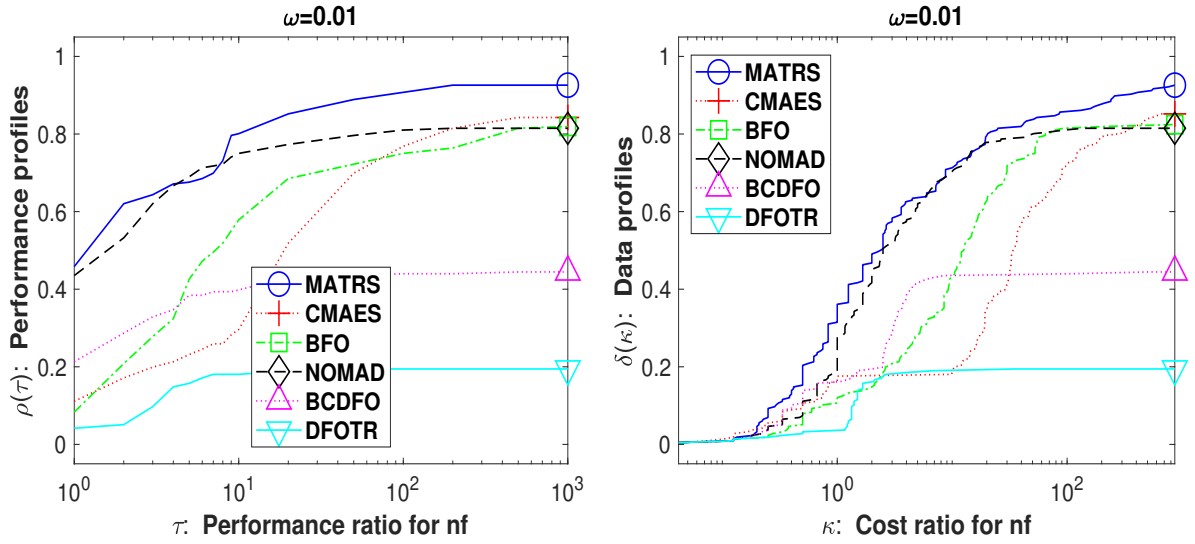


Figure 2: Plots for global for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
209 of 216 global problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	199	17	0	0	63	68
CMAES	173	43	0	0	20	30
BFO	172	0	0	44	26	51
NOMAD	170	0	0	46	55	47
BCDFO	79	11	4	122	26	14
DFOTR	39	170	0	7	7	5

Table 3: Tabulated results for global for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

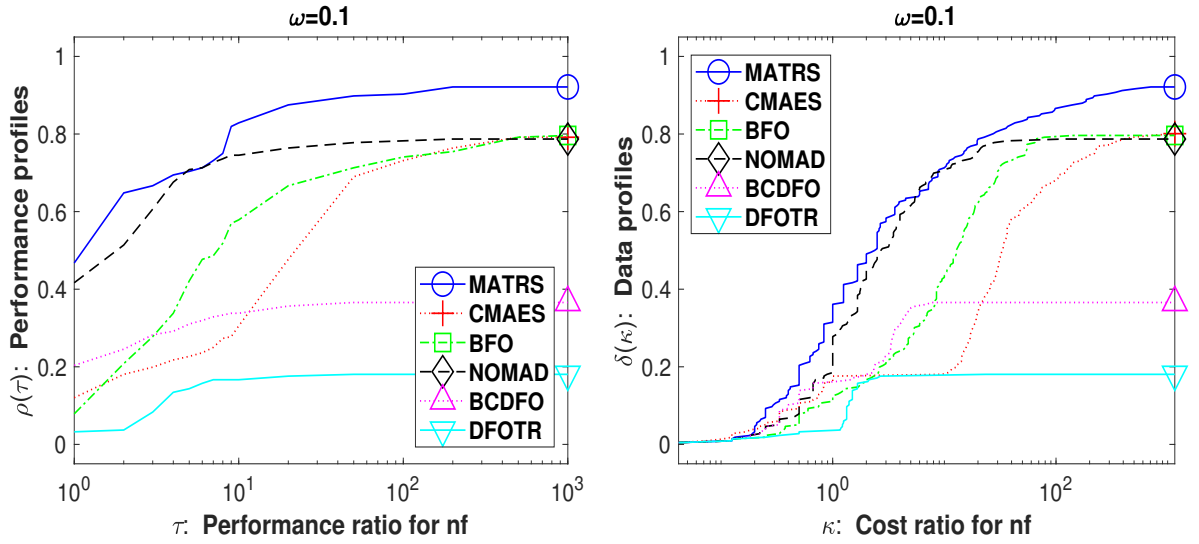


Figure 3: Plots for global for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
211 of 230 bcp problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	193	37	0	0	40	46
NOMAD	183	0	1	46	51	36
CMAES	182	48	0	0	25	42
BFO	136	5	0	89	28	48
DFOTR	105	90	1	34	34	19
BCDFO	87	65	1	77	24	11

Table 4: Tabulated results for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

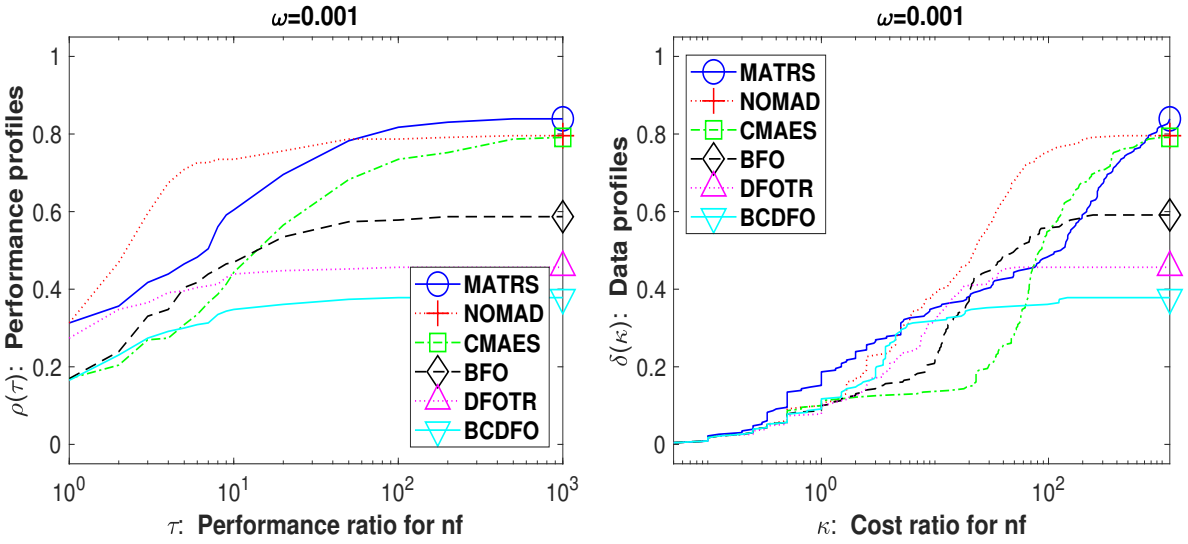


Figure 4: Plots for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
193 of 230 bcp problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	173	57	0	0	42	45
NOMAD	159	0	2	69	45	36
CMAES	155	75	0	0	22	33
BFO	119	0	0	111	28	40
DFOTR	75	102	0	53	24	16
BCDFO	72	81	2	75	21	10

Table 5: Tabulated results for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

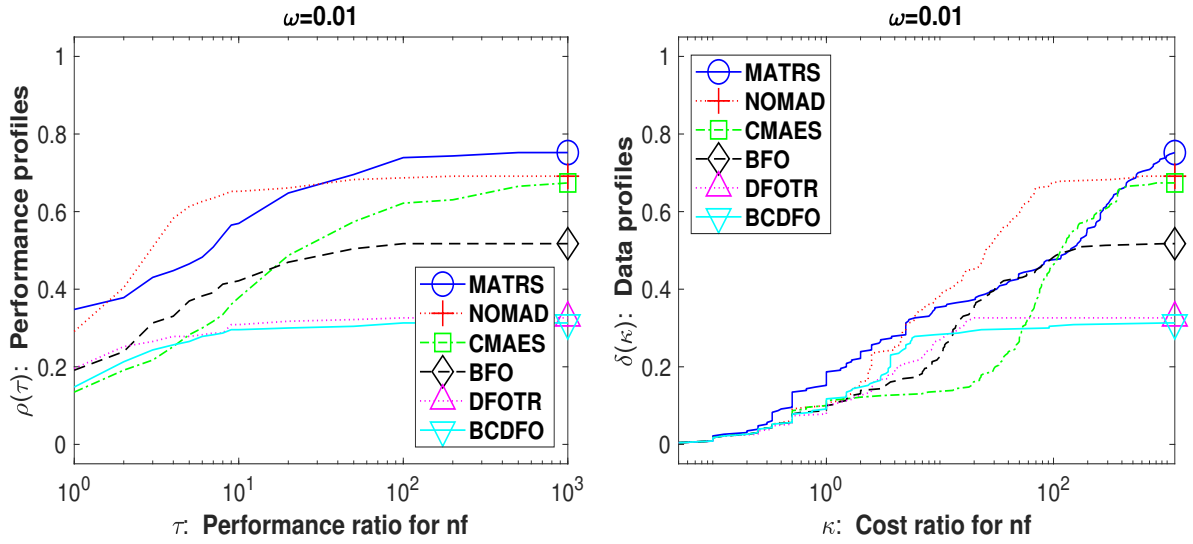


Figure 5: Plots for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
176 of 230 bcp problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	154	76	0	0	42	46
NOMAD	132	0	0	98	38	34
CMAES	124	106	0	0	20	26
BFO	87	0	0	143	21	27
BCDFO	65	89	2	74	20	9
DFOTR	55	110	0	65	19	11

Table 6: Tabulated results for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

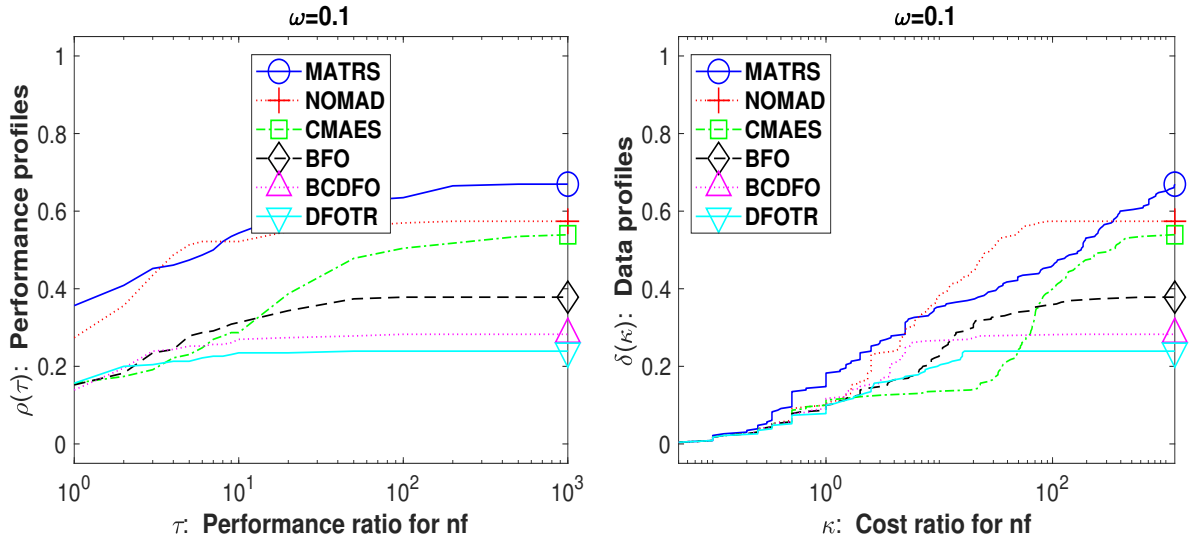


Figure 6: Plots for bcp for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
542 of 571 prince problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	498	72	0	1	62	63
NOMAD	496	0	3	72	55	44
CMAES	492	75	0	4	26	31
BFO	417	4	0	150	33	48
BCDFO	247	117	5	202	31	11
DFOTR	194	311	12	54	27	14

Table 7: Tabulated results for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

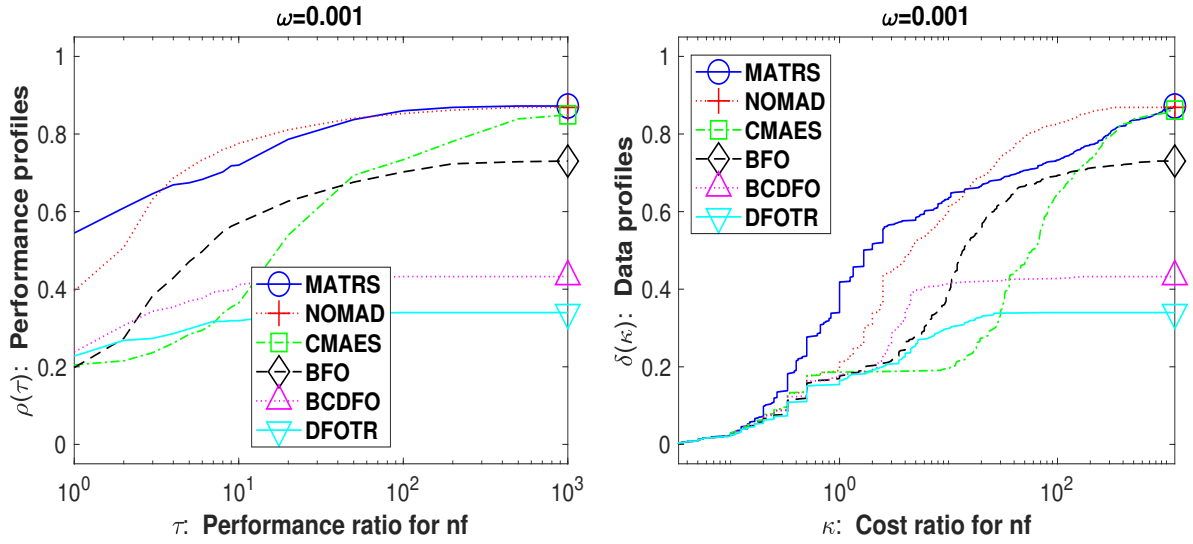


Figure 7: Plots for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
529 of 571 prince problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	483	87	0	1	62	61
NOMAD	471	0	2	98	54	43
CMAES	447	120	0	4	26	29
BFO	379	2	0	190	31	42
BCDFO	237	138	4	192	29	11
DFOTR	163	324	11	73	23	13

Table 8: Tabulated results for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

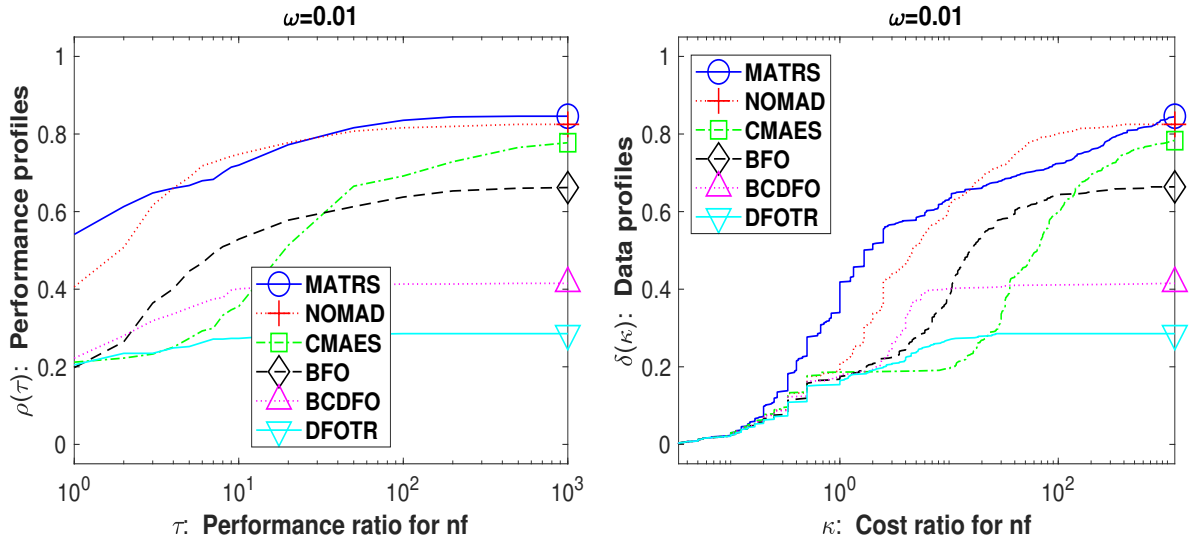


Figure 8: Plots for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
509 of 571 prince problems solved						
dim∈[1,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	459	112	0	0	62	61
NOMAD	438	0	0	133	53	42
CMAES	390	178	0	3	25	25
BFO	335	0	0	236	29	35
BCDFO	204	160	4	203	25	9
DFOTR	140	346	4	81	20	10

Table 9: Tabulated results for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

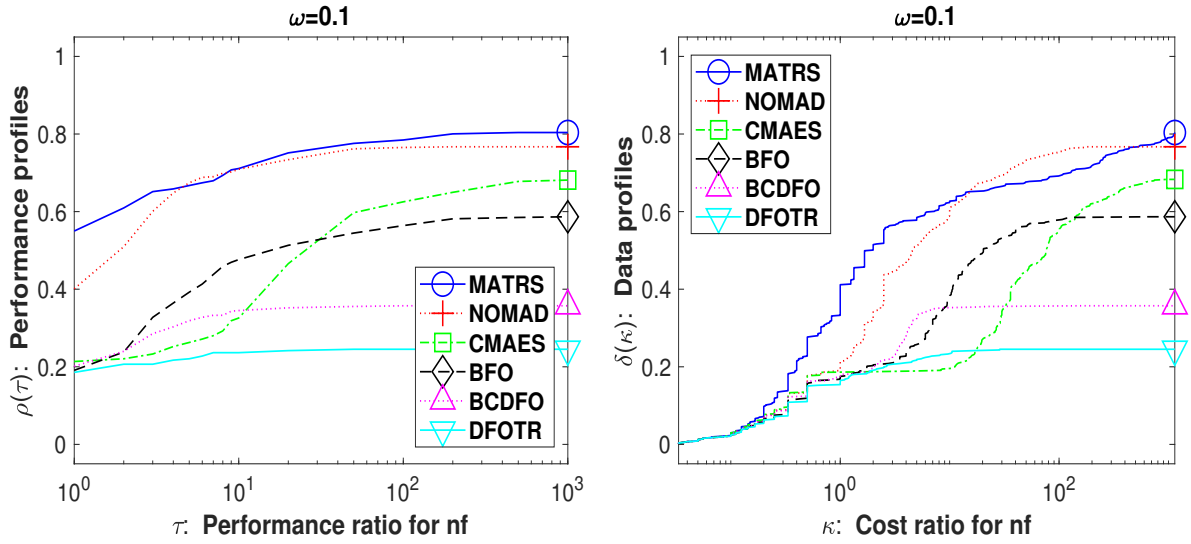


Figure 9: Plots for prince for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

Figs. 1–9 and Tables 1–9 show that for all noise levels:
• On all continuous problem collections, MATRS is the most robust solver.
• On bcp, NOMAD is the most efficient solver, while for global and prince, MATRS is the most efficient solver.

3 Results for globalInt, bcpInt, and princeInt

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
215 of 216 globalInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	203	11	2	0	50	46
BFO	188	0	0	28	10	23
NOMAD	184	0	4	28	49	47
DFLINT	184	0	21	11	63	63
CMAES	141	75	0	0	12	22

Table 10: Tabulated results for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

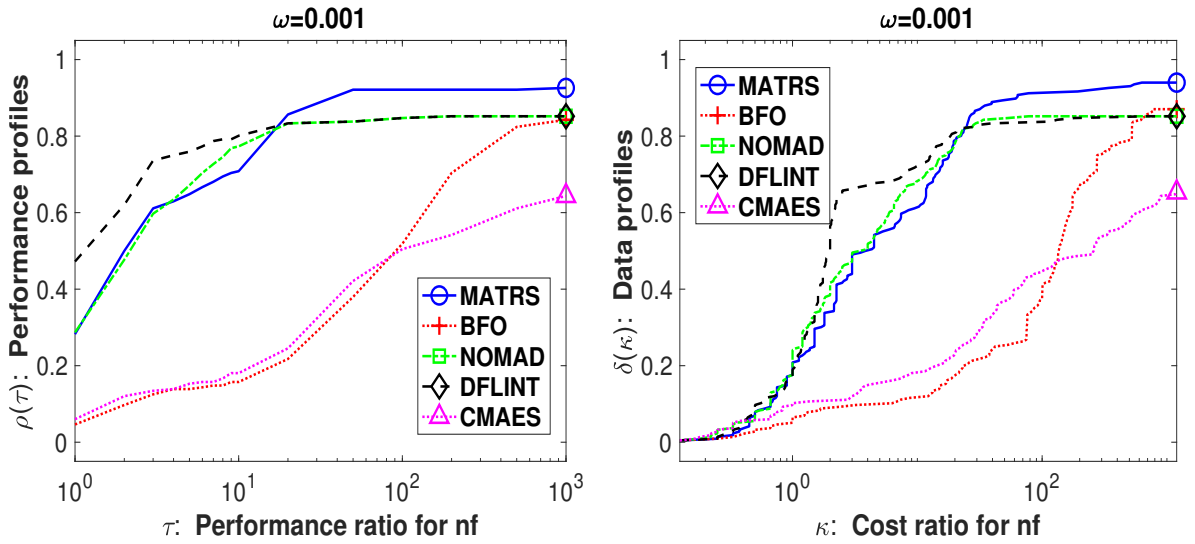


Figure 10: Plots for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
214 of 216 globalInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	202	12	2	0	49	47
NOMAD	186	0	4	26	52	47
BFO	183	0	0	33	10	22
DFLINT	181	1	22	12	62	62
CMAES	151	65	0	0	12	22

Table 11: Tabulated results for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

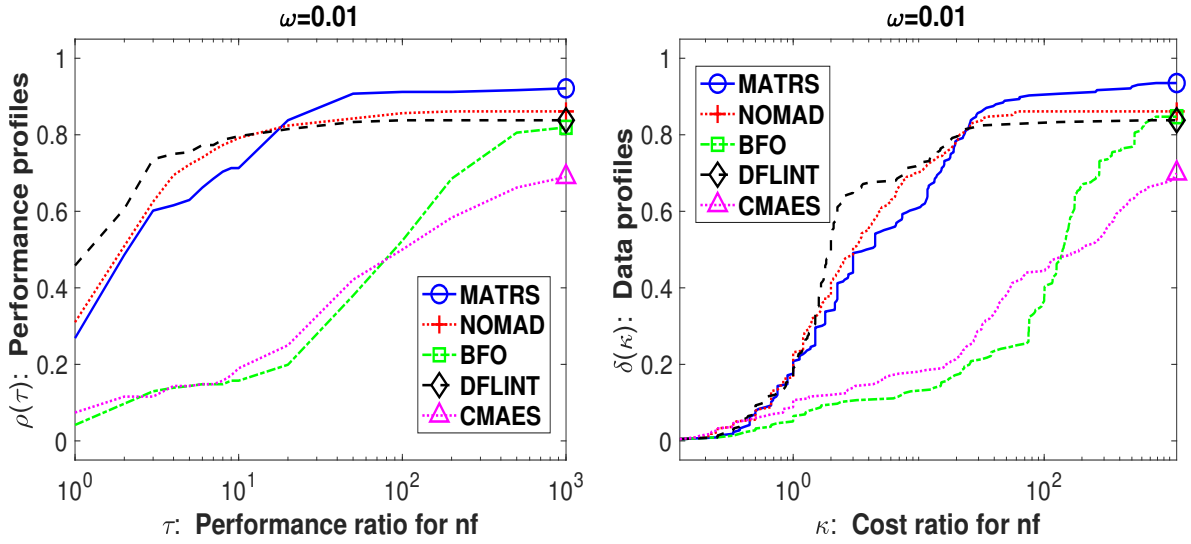


Figure 11: Plots for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
214 of 216 globalInt problems solved						
dim∈[2,30]		# of anomalies			e_s in %	
solver	solved	#n	#t	#f	nf	sec
MATRS	196	19	1	0	47	53
NOMAD	182	0	7	27	51	43
DFLINT	178	2	21	15	61	56
BFO	167	0	0	49	10	22
CMAES	152	64	0	0	13	25

Table 12: Tabulated results for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

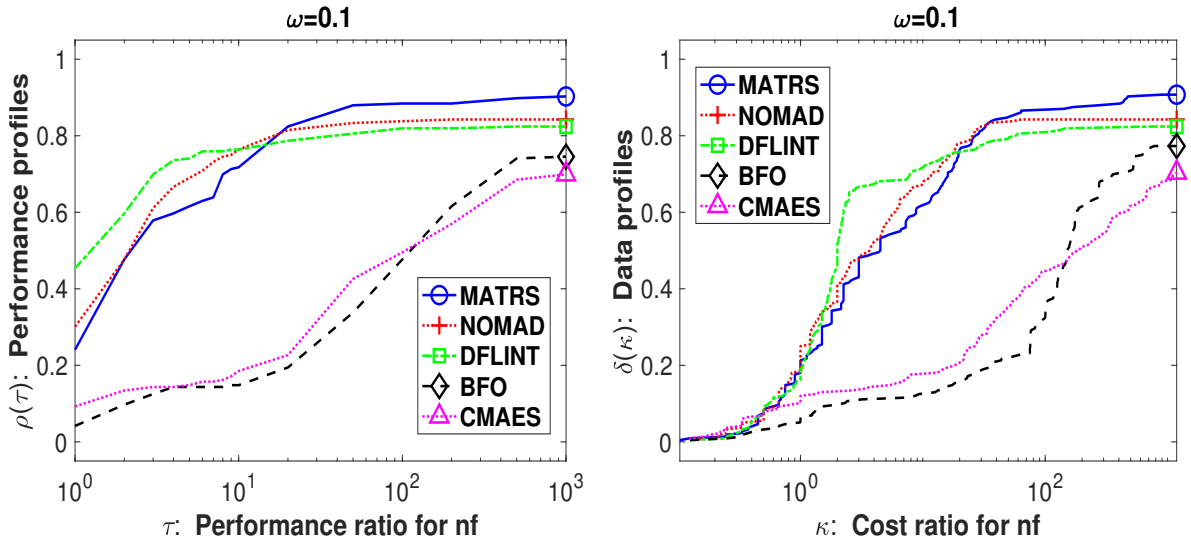


Figure 12: Plots for globalInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
226 of 230 bcpInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	207	20	3	0	57	51
DFLINT	186	30	3	11	44	55
NOMAD	181	0	1	48	47	38
BFO	105	1	0	124	19	27
CMAES	73	157	0	0	8	13

Table 13: Tabulated results for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

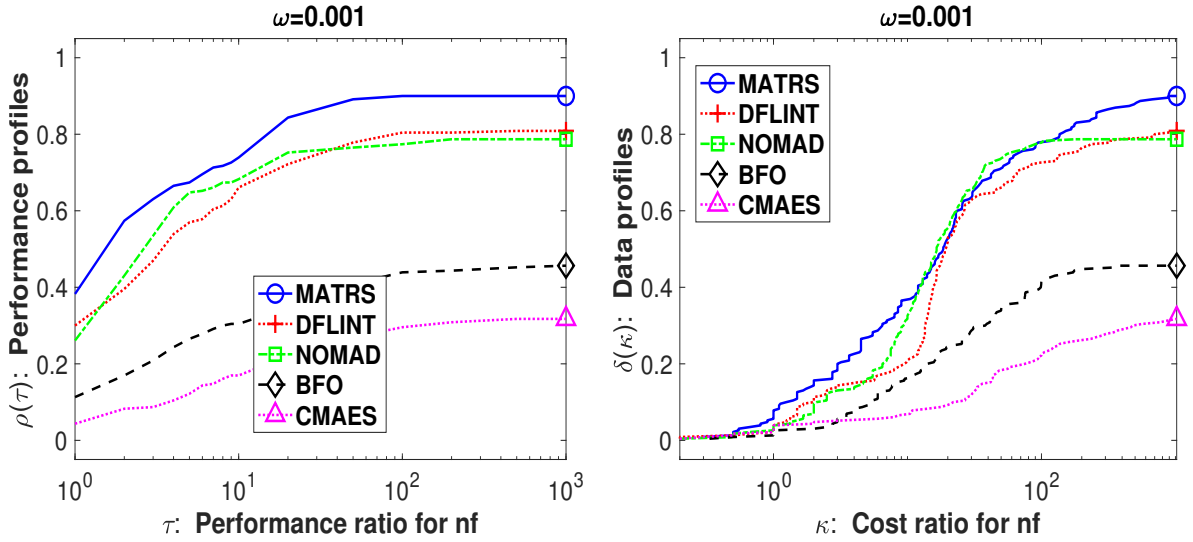


Figure 13: Plots for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
216 of 230 bcpInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	199	26	5	0	53	51
DFLINT	181	25	4	20	44	53
NOMAD	171	0	1	58	42	34
BFO	103	1	0	126	18	26
CMAES	71	159	0	0	8	12

Table 14: Tabulated results for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

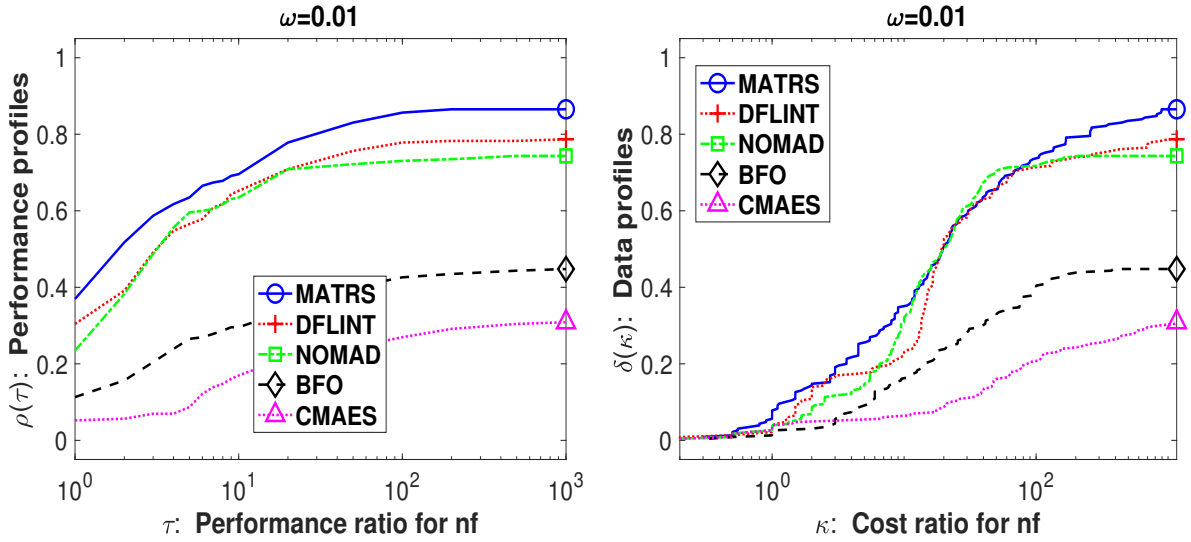


Figure 14: Plots for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
203 of 230 bcpInt problems solved						
dim$\in$[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	173	50	7	0	47	45
DFLINT	159	39	3	29	37	45
NOMAD	151	0	5	74	38	33
BFO	75	1	0	154	14	19
CMAES	73	157	0	0	10	15

Table 15: Tabulated results for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

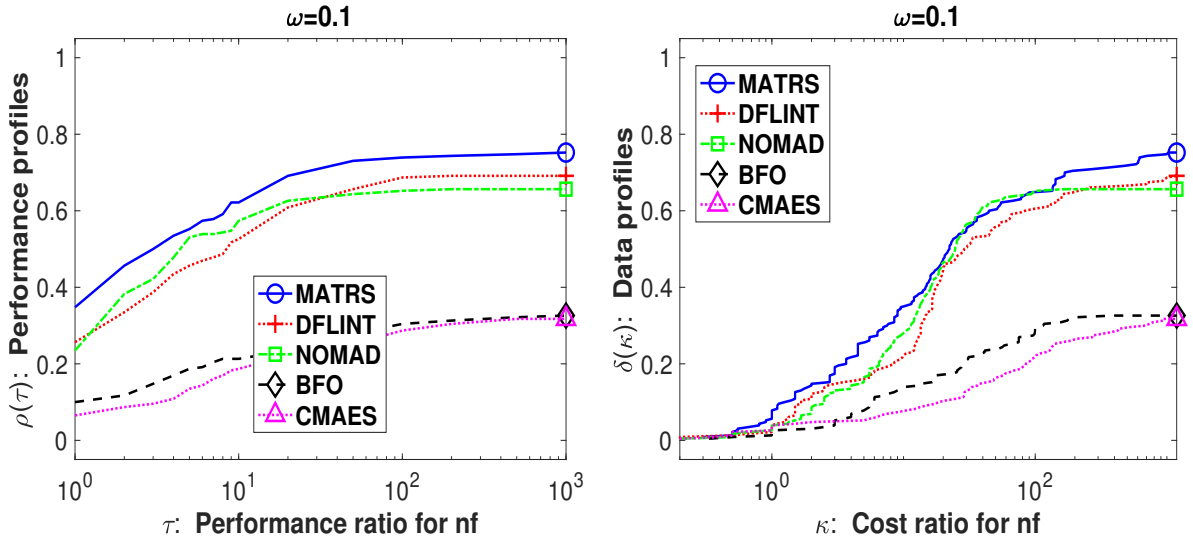


Figure 15: Plots for bcpInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-3}$						
557 of 571 princeInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	550	13	7	1	61	63
DFLINT	537	11	6	17	59	61
NOMAD	521	0	1	49	51	41
BFO	444	0	0	127	15	31
CMAES	368	202	0	1	15	26

Table 16: Tabulated results for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$.

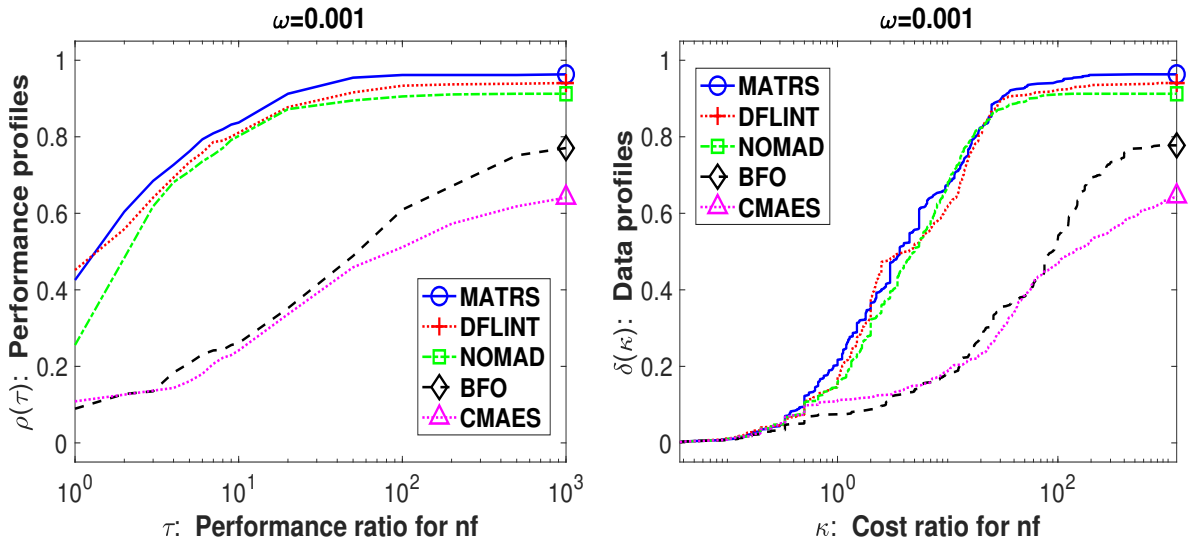


Figure 16: Plots for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.001$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-2}$						
557 of 571 princeInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	550	12	8	1	61	64
DFLINT	522	20	6	23	59	55
NOMAD	510	0	3	58	49	38
BFO	428	0	0	143	15	31
CMAES	367	204	0	0	14	26

Table 17: Tabulated results for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$.

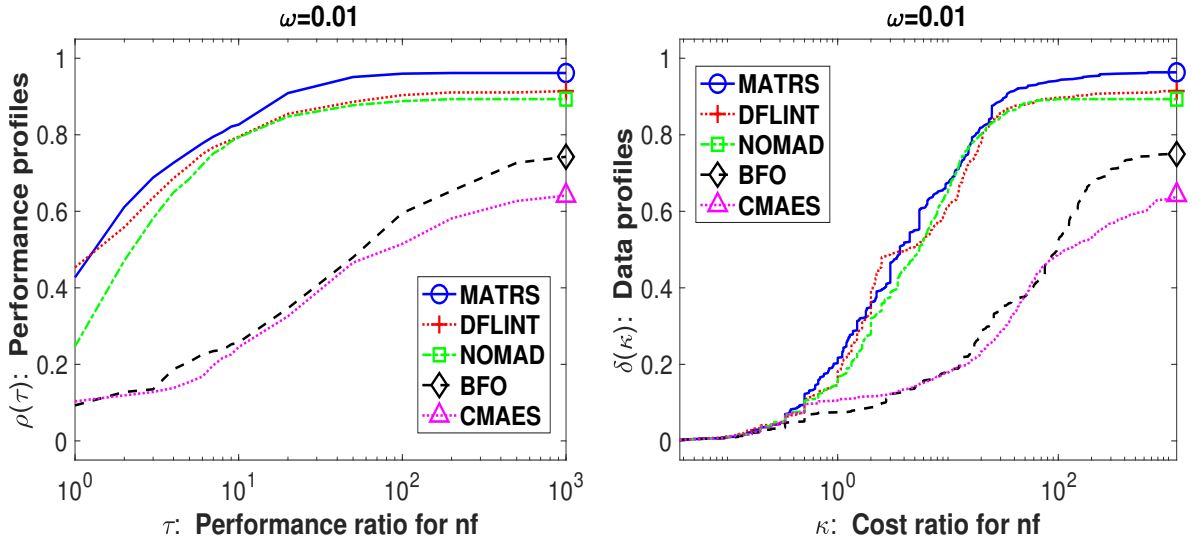


Figure 17: Plots for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.01$. Other details are as in Fig. 1.

stopping test: $q_f \leq 0.0001$, sec ≤ 360, nf $\leq 1200 * n$						
noise level: $\omega = 10^{-1}$						
551 of 571 princeInt problems solved						
dim∈[2,30]		# of anomalies			eff%	
solver	solved	#n	#t	#f	nf	sec
MATRS	531	34	5	1	62	66
DFLINT	492	36	5	38	56	50
NOMAD	479	0	2	90	47	33
CMAES	369	202	0	0	15	27
BFO	351	0	0	220	13	26

Table 18: Tabulated results for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$.

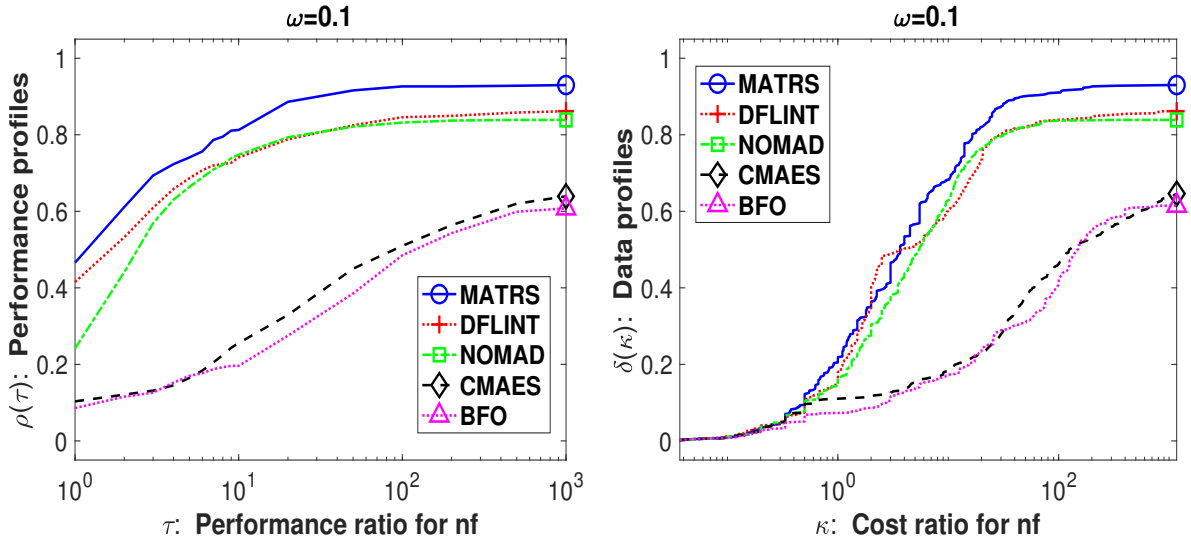


Figure 18: Plots for princeInt for dimensions $2 \leq n \leq 30$ and noise level $\omega = 0.1$. Other details are as in Fig. 1.

Figs. 10–18 and Tables 10–18 show that for all noise levels:
- On all three integer collections, MATRS is the most robust solver.
- On bcpInt and princeInt, MATRS is the most efficient solver.
- On globalInt, DFLINT is the most efficient solver.

References

[1] H. G. Beyer. Design principles for matrix adaptation evolution strategies (2020). In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO'20). ACM, New York, 682–700. https://doi.org/10.1145/3377929.3389870.

[2] X. W. Chang, X. Yang, and T. Zhou. MLAMBDA: A modified LAMBDA method for integer least-squares estimation. *J. Geod.* **79** (2005), 552–565.

[3] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust region methods*. Society for Industrial and Applied Mathematics (2000).

[4] A. Ghasemmehdi and E. Agrell. Faster recursions in sphere decoding. *IEEE Trans. Inf. Theory* **57** (2011), 3530–3536.

[5] N. Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772* (2016).

[6] W. Huyer and A. Neumaier. MINQ8: general definite and bound constrained indefinite quadratic programming. *Comput. Optim. Appl.* **69** (2018), 351–381.

[7] W. Huyer and A. Neumaier. SNOBFIT – stable noisy optimization by branch and fit. *ACM. Trans. Math. Softw.* **35** (2008), 1–25.

[8] G. Liuzzi, S. Lucidi, and F. Rinaldi. An algorithmic framework based on primitive directions and nonmonotone line searches for black-box optimization problems with integer variables. *Math. Program. Comput.* **12** (2020), 673–702.

[9] M. Kimiaei. An active set trust-region method for bound-constrained optimization. *Bull. Iran. Math. Soc.* **48** (2022), 1721–1745.

[10] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.* **14** (2022), 365–414.

[11] M. Kimiaei. (2023–2025). GS1400/MATRS: MATRSv4.0. Zenodo. https://doi.org/10.5281/zenodo.14993723.

[12] M. Kimiaei and A. Neumaier. MATRS – Heuristic methods for derivative-free bound-constrained mixed-integer optimization. *Math. Program. Comput.* (2025).