suppMat: Supplemental material for VRDFON

A supplemental material for the VRDFON solver, called suppMat, is provided. VRDFON is an improved version of the VRDFON-basic algorithm discussed in [17]. The VRDFON package is publicly available at [16].

In Section 1, we first describe several heuristic enhancements added to the VRDFON-basic algorithm leading to VRDFON. In Section 2, we provide details on the solvers compared. In Section 3, we make a testing and tuning for five important tuning parameters of VRDFON to increase its efficiency and robustness, and discuss under what conditions the complexity results for VRDFON can be guaranteed.

1 Heuristic enhancements

In this section we propose many practical improvements that make VRDFON a very competitive solver. Two of the most important of these are *surrogate quadratic models in adaptively determined subspaces* and finding and updating step sizes in an improved version of MLS-basic, called MLS. Improved versions of DS-basic, called DS, use MLS with different search directions. DS heuristically reconstructs step sizes in each unsuccessful iteration, resulting in small extrapolation step sizes. VRDFON calls DS repeatedly until an $\varepsilon = \sqrt{\omega}$-approximate stationary point is found.

1.1 Random approximate coordinate directions

The use of random directions is preferable to the use of deterministic directions (see [3]). On the other hand, it is known that coordinate directions are useful to estimate the gradient, since at least one of these directions has a good angle with the gradient (see [8]). Inspired by these points, we construct an effective version of scaled random directions, which we call *random approximate coordinate directions*, that combine the advantages of both scaled random and coordinate directions. The random coordinate direction p multiplies a standard random direction by a scaling vector having one component equals to $1/\|p\|$ and the others equal to $\gamma_{\text{rd}}/\|p\|$ with $0 < \gamma_{\text{rd}} < 1$, where is a small tuning parameter.

1.2 Subspace information

The points Z_i with the best inexact function values are stored as columns of a matrix Z, their inexact function values $\tilde{f}(Z_i)$ in the vector F, and their step sizes α_i in the vector S. We adjust the matrix Z if its entries are contaminated by `NaN` or $\pm\infty$ by inserting a large positive tuning parameter $\gamma_Z > 0$.

We denote the points stored in Z as *sample points*, whose maximum number is defined by $m_{\max} := \min\left\{\overline{m}, \frac{1}{2}n(n+3)\right\}$, where $\overline{m}$ is a tuning parameter. The *number of sample points* is denoted by $m \in [2, m_{\max}]$ and the *subspace size* m^{o} is defined as the largest integer satisfying $\frac{1}{2}m^{\mathrm{o}}(m^{\mathrm{o}}+3) \leq m$.

If m is greater than $m_{\max}$, we update Z, F, S by replacing the worst point, its inexact function value, and its step size with the current best point, its inexact function value, and its step size, respectively. Otherwise, we append the current best point to Z, its inexact function value to F, and its step size to S.

1.3 Random subspace directions

In [18], it was shown by extensive numerical results that after using a derivative-free line search algorithm with coordinate directions, the use of random subspace directions by such an algorithm is very useful. Inspired by this, after using random coordinate directions by `VRDFON`, random subspace directions are used by such an algorithm in the hope of reducing the influence of noise.

We write $A_{II} := (A_{ij})_{i\in I, j\in I}$ for the submatrix of A with row and column indices of I by A_{II}, $A_{:k}$ for the kth column of a matrix A, and b for the index of the best point.

Random subspace directions are constructed based on the information of sample points with good function values. As in [18], we generate an $(m-1)\times 1$ standard random vector α^{rs} and then scale α^{rs} as $\alpha^{\mathrm{rs}} := \alpha^{\mathrm{rs}}/\|\alpha^{\mathrm{rs}}\|$. Then we compute the *random subspace direction* by $p := \sum_{i=1, i\neq b}^{m} \alpha_i^{\mathrm{rs}}(Z_{:i} - Z_{:b})$.

1.4 Reduced quadratic models

It is well-known [14] that model-based algorithms need at least

$$N := n + \frac{1}{2}n(n+1) = \frac{1}{2}n(n+3)$$

sample points and $\mathcal{O}(N^3)$ operations for estimating the gradient vector and Hessian matrix of an objective quadratic model. For medium or large scale problems, this is prohibitively expensive. To overcome this problem, we construct quadratic models in adaptively determined subspaces called *reduced quadratic models*, one of which is a fully quadratic model when $m = N$.

For all $i = 1, \cdots, m$, let z_i and $\tilde{f}_i := \tilde{f}(z_i)$ be the sample points and their inexact function values stored in Z and F, respectively, and define $\mathbf{s}_i := z_i - Z_{:b}$. Before defining the model errors, we need to know whether or not the number of sample points m is large enough to build a full or reduced quadratic model. Therefore, we compute all the subspace sizes that are admissible to construct quadratic models. The subspace size defined in Subsection 1.2 is calculated as follows

$$m^{\mathrm{o}} := \left\lfloor \frac{1}{2}(-3 + \sqrt{9 + 8m}) \right\rfloor. \tag{1}$$

It is clear that for $m < N$ no fully quadratic model can be constructed; reduced quadratic models are constructed instead. When the dimension is larger than $m_{\max}$, the $n \times m_{\max}$ matrix Z and the $n \times 1$ vector z^m are reduced to the $m_{\max} \times m^{\mathrm{o}}$ matrix Z^{o} and the $m^{\mathrm{o}} \times 1$ vector z^{o}, respectively. In other words, the restriction of the entries of Z and z^m is done by choosing a random subset of size m^{o}.

Some components of each best point stored in Z may be ignored. To overcome this shortcoming, reduced quadratic models are constructed several times before m exceeds $m_{\max}$, each time choosing Z^{o} (z^{o}) from a random subset of the entries of Z (z_{best}). The different subsets are not drawn separately, so entries between two consecutive samples may overlap.

We write $\tilde{g}^{\mathrm{o}}$ and $\tilde{B}^{\mathrm{o}}$ for the estimated gradient vector and the symmetric Hessian matrix in a subspace, respectively. Let $M := \frac{1}{2}m^{\mathrm{o}}(m^{\mathrm{o}} + 3)$ and $\mathbf{K} := \min(2M, m - 1)$. To evaluate the inexact $\tilde{g}^{\mathrm{o}}$ and $\tilde{B}^{\mathrm{o}}$, we define the

model errors by

$$\varepsilon_i := \frac{\tilde{f}_i - F_b - (\tilde{g}^{\mathrm{o}})^T \mathbf{s}_i - \frac{1}{2}\mathbf{s}_i^T \tilde{B}^{\mathrm{o}} \mathbf{s}_i}{\mathtt{sc}_i} \quad \text{for all } i = 1, \cdots, \mathbf{K}, \tag{2}$$

where `sc` is an appropriate scaling vector. It is shown in DEUFLHARD & HEINDL [9] that numerical methods can generally perform much better when they preserve affine invariance. The following choice ensures the affine invariance of the fitting method.

The numerator of (2) is $\mathcal{O}(\|\mathbf{s}_i\|^3)$ if $m = N$; otherwise it is $\mathcal{O}(\|\mathbf{s}_i\|^2)$. To obtain ε_i of a uniform magnitude, we choose `sc` as follows:
(1) We form the matrix $\mathbf{S} := (\, \mathbf{s}_1 \quad \cdots \quad \mathbf{s}_{\mathbf{K}} \,)^T$, where $\mathbf{s}_i := z_i^{\mathrm{o}} - Z_{:b}^{\mathrm{o}}$ for all $i = 1, \cdots, \mathbf{K}$.
(2) We compute the matrix

$$H^{\mathrm{o}} := \Big(\sum_l \mathbf{s}_l \mathbf{s}_l^T \Big)^{-1} \tag{3}$$

by constructing a reduced QR factorization $\mathbf{S} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{\mathbf{K} \times m^{\mathrm{o}}}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{m^{\mathrm{o}} \times m^{\mathrm{o}}}$ is a square upper triangular matrix.
(3) We compute the scaling vector `sc`

$$\mathtt{sc}_i := (\mathbf{s}_i^T H^{\mathrm{o}} \mathbf{s}_i)^{e/2} \quad \text{for } i = 1, \cdots, \mathbf{K}, \tag{4}$$

with $\mathbf{s}_i^T H^{\mathrm{o}} \mathbf{s}_i = \|\mathbf{R}^{-T} \mathbf{s}_i\|^2$ for $i = 1, \cdots, \mathbf{K}$. In (4), $e = 3$ if $m = N$ holds (full quadratic model) and $e = 2$ otherwise (reduced quadratic model).

We calculate H^{o} and `sc` in the same way as `SNOBFIT` [14] by performing a reduced QR factorization, except that they are performed in adaptively determined subspaces.

We have $\varepsilon = Ay - a$, where, for $i = 1, \cdots, \mathbf{K}$,

$$a_i := \frac{F_b - \tilde{f}_i}{\mathtt{sc}_i} \quad \text{and} \quad A_{ij} := \begin{cases} \dfrac{\mathbf{s}_i^j}{\mathtt{sc}_i} & \text{if } j \in \{1, \cdots, m^{\mathrm{o}}\}, \\ \dfrac{(\mathbf{s}_i^{j-m^{\mathrm{o}}})^2}{2\mathtt{sc}_i} & \text{if } j \in \{m^{\mathrm{o}}+1, \cdots, 2m^{\mathrm{o}}\}, \\ \dfrac{\mathbf{s}_i^{j'} \mathbf{s}_i^{j''}}{\mathtt{sc}_i} & \text{if } j \in \{2m^{\mathrm{o}}+1, \cdots, M\}. \end{cases} \tag{5}$$

Here j' and j'' are the remainders of the division of $j - 2m^{\text{o}}$ and $j - 2m^{\text{o}} + 1$ by m^{o}, respectively, and $\mathbf{s}_i^j$ is the jth component of the vector $\mathbf{s}_i$. To find the entries of the inexact $\tilde{g}^{\text{o}}$ and $\tilde{B}^{\text{o}}$ we solve the linear least squares problem

$$\min_{y \in \mathbb{R}^M} \|Ay - a\|_2^2. \tag{6}$$

In finite precision arithmetic, each of the vectors a, $\mathtt{sc}$, y, and hence $\tilde{g}^{\text{o}}$, $\tilde{B}^{\text{o}}$ can have entries with value $\mathtt{NaN}$ or $\pm\infty$. We replace the components of the vectors a, $\mathtt{sc}$ and y with value $\mathtt{NaN}$ or $\pm\infty$ by a large positive tuning parameter $\gamma_v > 0$.

We construct the vector a and the matrix A by (5), adjust a, and find all multipliers by solving (6). Then we adjust y and define $\tilde{g}^{\text{o}}$ by the first m^{o} components of y, the diagonal entries of $\tilde{B}^{\text{o}}$ by the next m^{o} components of y, and the off-diagonal entries of $\tilde{B}^{\text{o}}$ symmetrically by the remaining entries of y.

In summary, we construct reduced quadratic models. Two advantages of such models are the use of limited sample points and the ability to construct them multiple times by increasing the size of the subspace from 2 to $m_{\max}$. To obtain a robust model, we adjust the matrix Z and the vector y whenever they are contaminated by $\mathtt{NaN}$ or $\pm\infty$. Second, we compute $\mathtt{sc}$ and adjust it. Third, we calculate y and adjust it. Finally, we estimate $\tilde{g}^{\text{o}}$ and $\tilde{B}^{\text{o}}$.

1.5 Perturbed random directions

In addition to scaled random directions and random subspace directions, the use of other directions, such as *perturbed random directions* (see below) and *improved trust region directions* (explained later in Subsection 1.6) can improve the efficiency of the method in the presence of noise.

perturbed random directions are used if $\tilde{B}^{\text{o}}$ is not computable, which implies that at least one entry of $\tilde{B}^{\text{o}}$ is contaminated by NaN or $\pm\infty$. Otherwise, reduced quadratic models are constructed in Subsection 1.4 and improved trust region directions are computed.

A perturbed random direction $\tilde{p} := p^{\text{o}} - \alpha^{\text{o}}\tilde{g}^{\text{o}}$ is a perturbation of a standard random direction p^{o} by the steepest descent direction $-\tilde{g}^{\text{o}}$ with the approximate gradient. Both $\tilde{g}^{\text{o}}$ and p^{o} are restricted to a subspace which is

a random subset $\mathcal{J}$ of $\{1, \cdots, n\}$ whose size is m^{o}. For all $i \notin \mathcal{J}$, $\tilde{p}_i = 0$. To be numerically appropriate, both are scaled by the heuristic step size $\alpha^{\mathrm{o}} := (1 + \kappa(\tilde{g}^{\mathrm{o}})^T p^{\mathrm{o}})/\|\tilde{g}^{\mathrm{o}}\|^2$ and the decreasing sequence $\kappa := 1/(1 + n_{\mathrm{f}})^{\gamma_\kappa}$ with the tuning parameter $0 < \gamma_\kappa < 1$ and the number of function evaluations n_{f}. It is easy to show that $\tilde{p}^T \tilde{g} = (\kappa p^{\mathrm{o}} - \alpha^{\mathrm{o}} \tilde{g}^{\mathrm{o}})^T \tilde{g}^{\mathrm{o}} < 0$; hence the perturbed random directions are descent directions.

1.6 An improved trust region direction

The goal of trust region methods is to restrict steps within a trust region to increase the accuracy of surrogate models. Therefore, trust region directions can be very useful, even in the presence of noise.

We now solve the trust region subproblem in a subspace

$$\begin{aligned} \min \quad & \zeta^T \tilde{g}^{\mathrm{o}} + \frac{1}{2} \zeta^T \tilde{B}^{\mathrm{o}} \zeta \\ \text{s.t.} \quad & \|\zeta - z^{\mathrm{o}}\|_\infty \leq d, \end{aligned}$$

whose solution is denoted by ζ_{best}. Here d denotes the *trust region radius*. The trust region direction is $p^{\mathrm{o}}_{\mathrm{tr}} := \zeta_{\mathrm{best}} - z^{\mathrm{o}}$ in a subspace which is a random subset of $\{1, \cdots, n\}$ whose size is m^{o}. The idea is to construct the *improved trust region direction* by scaling $p^{\mathrm{o}}_{\mathrm{tr}}$ with the positive tuning parameter γ_p and perturbing it by $p_{\mathrm{mean}} := z_{\mathrm{mean}} - Z_{:b}$, where z_{mean} is the mean of Z. This perturbation transforms $p^{\mathrm{o}}_{\mathrm{tr}}$ into a full subspace direction enriched by p_{mean}.

1.7 `MLS` – an improved version of `MLS-basic`

In this section, we discuss an improved version of `MLS-basic`, `MLS`. Namely, `MLS` is enriched by heuristic techniques for finding and updating step sizes.

The efficiency of line search methods depends on how their step sizes are updated. The line search algorithms discussed in [18] and [21] find their step sizes in a way that does not seem to be effective, especially for large scale problems, because step sizes are updated independently; they depend only on the corresponding step size generated by the previous executions. To address this shortcoming, we construct an improved version of `MLS-basic` whose step sizes are generated and updated in a new way.

Let $0 \leq \underline{\alpha}_{\text{init}} < \overline{\alpha}_{\text{init}} \leq \infty$ be the lower and upper bounds of an initial interval as the tuning parameters. We start with the initial interval $[\underline{\alpha}, \overline{\alpha}] = [\underline{\alpha}_{\text{init}}, \overline{\alpha}_{\text{init}}]$. Then $\underline{\alpha}$ or $\overline{\alpha}$ is updated. We describe this below.

• *The initial interval.* When the trial point $z^r = z_{\text{best}} \pm \alpha_r p^r$ and its inexact function value $\tilde{f}(z^r)$ are computed in line 25 of `VRDFON-basic`, the gain $\tilde{f}(z_{\text{best}}) - \tilde{f}(z^r)$ is stored in the vector `dF` and its step size α_r in the vector $\mathbf{a}$. Namely, when an extrapolation with many sufficient gains along p^r ($r \in \{1, \cdots, R_m\}$) is found, much information is stored in the vectors `dF` and $\mathbf{a}$. Then, an index set of the stored points with the decreasing inexact function values is found by

$$\texttt{ind} := \{i \mid \texttt{dF}_i < 0 \ \text{ for } i = 1, \cdots, n\}.$$

If `ind` is non-empty, the lower bound of the interval $[\underline{\alpha}, \overline{\alpha}]$ can be found by $\underline{\alpha} := \max(\mathbf{a}_{\texttt{ind}})$; if $\underline{\alpha} = \underline{\alpha}_{\text{init}}$ is given as a positive tuning parameter, then $\underline{\alpha}$ is updated by $\underline{\alpha} := \min(\underline{\alpha}, \max(\mathbf{a}_{\texttt{ind}}))$. Next, an index set of positive gains or the corresponding step sizes strictly larger than $\overline{\alpha}$ is found by

$$\overline{\texttt{ind}} := \{i \mid \texttt{dF}_i \geq 0 \text{ or } \mathbf{a}_i > \overline{\alpha} \ \text{ for } i = 1, \cdots, n\}.$$

If $\overline{\texttt{ind}}$ is nonempty, an upper bound on the initial interval is found by $\overline{\alpha} := \min(\mathbf{a}_{\overline{\texttt{ind}}})$; if $\overline{\alpha} = \overline{\alpha}_{\text{init}}$ is given as a positive tuning parameter, the upper bound of the initial interval is updated by $\overline{\alpha} := \max(\overline{\alpha}, \min(\mathbf{a}_{\overline{\texttt{ind}}}))$.

• *The initial extrapolation step size.* If extrapolation step sizes are too small, extrapolations may be performed slowly with many function evaluations. To overcome this problem, we only change line 19 in `VRDFON-basic` to $\alpha_1 := \max\{\sqrt{\underline{\alpha}\overline{\alpha}}, \delta_k\}$ whenever the interval $[\underline{\alpha}, \overline{\alpha}] \subseteq (0, \infty)$ is found.

• *Reducing extrapolation step sizes.* Since the source of the inefficiency of the line search algorithm is generating step sizes that are too small when the interval has already been found, we change line 36 in `VRDFON-basic` to

$$\alpha_{r+1} := \max\left\{\overline{\alpha}_{\min}, \min\left\{\sqrt{\underline{\alpha}\overline{\alpha}}, \frac{\alpha_r}{\gamma_e}\right\}\right\}$$

with the goal of slowly reducing step sizes. Here $\overline{\alpha}_{\min} \in (0, 1)$ is a minimum threshold for the step size which is a tuning parameter.

• *Updating the interval.* After finding the new step size α_r, we update either the lower or the upper bound of the interval. This can be done in the following two cases:

(i) After extrapolation has been performed with $\gamma\alpha_r^2$-sufficient gain along one of $\pm p^r$ (`good` is true).
(ii) After no $\gamma\alpha_r^2$-sufficient gain along $\pm p^r$ has been found (`good` is false).
To do this, we insert the following statement between lines 40 and 41 of `VRDFON-basic`:

1: **if** $\alpha_{r+1} > \underline{\alpha}$ **then**, set $\overline{\alpha} = \alpha_{r+1}$; ▷ upper bound is updated
2: **else**, set $\underline{\alpha} = \alpha_{r+1}$; ▷ lower bound is updated
3: **end if**

• *Decrease of f in flat regions.* If the rth iteration of `MLS` is unsuccessful (no $\gamma\alpha_r^2$-sufficient gain along $\pm p^r$ is found), the best point, its inexact function value, and its step size are updated when $\tilde{f}(z_{\text{best}} + \alpha_r p^r) < \tilde{f}(z_{\text{best}})$. In particular, this can happen in very flat regions of the feasible domain that do not contain a nearby stationary point. Therefore, we insert the following statement between lines 39 and 40 in `VRDFON-basic`:

1: **if** $\tilde{f}(z^r) < \tilde{f}(z_{\text{best}})$ **then**, set $z_{\text{best}} := z^r$ and $\tilde{f}(z_{\text{best}}) := \tilde{f}_e^r$; $n_{\texttt{succ}}^{\texttt{MLS}} := n_{\texttt{succ}}^{\texttt{MLS}} + 1$;
2: **end if**

• *Updating the best point.* If the rth iteration of `MLS` is successful ($\gamma\alpha_r^2$-sufficient gains are found along one of $\pm p^r$). A point with lowest inexact function value is chosen as the new best point. As discussed in Subsection 1.2, the matrices X, F, and S are then updated. In fact, lines 36 and 37 of `VRDFON-basic` should be changes as:

1: find $i' := \min\limits_{i\geq 0} \left\{ \tilde{f}(z_{\text{best}} + \alpha_r \gamma_e^i p^r) \;\middle|\; \tilde{f}(z_{\text{best}} + \alpha_r \gamma_e^i p^r) < \tilde{f}(z_{\text{best}}) \right\}$;
2: compute $\alpha_{r+1} = \gamma_e^{i'} \alpha_r$, $z_{\text{best}} := z_{\text{best}} + \alpha_{r+1} p^r$, $\tilde{f}(z_{\text{best}}) := \tilde{f}(z_{\text{best}} + \alpha_{r+1} p^r)$;
3: update $n_{\texttt{succ}}^{\texttt{MLS}} := n_{\texttt{succ}}^{\texttt{MLS}} + 1$;

1.8 `DS` – an improved version of `DS-basic`

This section discusses `DS`, an improved version of `DS-basic` that extended to include different search directions and heuristic techniques for updating step sizes and reconstructing the lower and upper bounds of the interval. It also describes how many worst case function evaluations are used in each successful and unsuccessful iteration of `DS`.

Denote by d_t the trust region radius in the iteration t and let C be the number of random approximate coordinate directions. Let $\gamma_{d_1} > 1$ and $\gamma_{d_2} \in (0, 1)$

be the parameters for updating d_t and let $0 < d_{\min} < d_{\max} < \infty$ be the parameters for controlling d_t. As described in Section 4 of [17], `DS-basic` has R_m calls to `MLS-basic` with scaled random directions. An improved version of `DS-basic`, `DS`, retains this procedure and additionally has C calls to `MLS-basic` with random approximate coordinate directions with the goal of restoring and updating the m sample points and their inexact function values. Then, `DS` constructs a quadratic model or a linear model to produce an improved trust region direction or a perturbed random direction, and makes T_0 calls to `MLS` along these directions as long as `MLS` is efficient (`good` is true). If `DS` is inefficient for all T_0 calls to `MLS` ($n_{\texttt{succ}}^{\texttt{DS}} = 0$), the lower bound $\underline{\alpha}$ of the current interval $[\underline{\alpha}, \overline{\alpha}]$ becomes too small, so the length of this interval becomes large. Therefore, we need to restart the interval and heuristically reconstruct the lower and upper bounds of this interval using the information from the m sample points. Let $\gamma_a > 0$ be the parameter for adjusting heuristic step size. The structure of `DS` is shown below.

We discuss how many function evaluations are required for each successful and unsuccessful iteration of `DS` in the worst case:
• Lines 2-3 of this procedure require at most $2R_m$ function evaluations if only scaled random directions are used, while this procedure is inefficient. This is because `MLS` is inefficient with R_m scaled random directions and R_m their opposite directions. Here the definition of inefficient comes from [17]. It means that `MLS` finds no reduction in the inexact function value.
• Lines 2-3 of this procedure use at most $2C$ function evaluations if only random approximate coordinate directions are used, while this procedure is inefficient. Since `MLS` is inefficient with C random approximate coordinate directions and C their opposite directions.
• Lines 2-3 of this procedure use at most $2R_m + 2C$ function evaluations if both scaled random directions and random approximate coordinate directions are used, while this procedure is inefficient. Since `MLS` is inefficient with $R_m + C$ directions and $R_m + C$ their opposite directions.
• Lines 5-8 of this procedure use at most 2 function evaluations, while this procedure is inefficient. This is because `MLS` is inefficient with the first random subspace direction and its opposite direction.
• Either lines 16-20 or 22-26 of this procedure use at most 3 function evaluations. Since an extrapolation step is attempted with at most two function evaluations, `MLS` is efficient along the opposite direction of the last improved trust region direction (or the last random perturbed direction). Then, the best point is updated.
The result of this discussion is that `DS` requires at most $2R_m + 2C + 4$ function evaluations for each unsuccessful iteration and at most $2R_m + 2C + 5$

Algorithm 1 `DS` – an improved version of `DS-basic`

for $t = 1, \cdots, T_0$ **do**
  perform `MLS` with either R_m *scaled random directions*, C *random approximate coordinate directions*, or both;
  if $m \geq 3$ **then**
    while true **do** ▷ until `good` is true
      perform `MLS` with *random subspace directions*;
      if `good` is false, **break**; **end** ▷ `MLS` is inefficient
    end while
  end if
  if $m \geq 2$ **then** ▷ a reduced quadratic or a linear model
    if both estimations are computable **then**
      construct a reduced quadratic model;
      generate the trust region radius by $d_t := \|z_{\text{mean}} - Z_{:b}\|$;
      restrict the trust region radius by

$$d_t := \max(d_{\min}, \min(d_{\max}, \gamma_{d_1} d_t));$$

      minimize the model to get an improved trust region direction;
      while true **do** ▷ until `good` is true
        perform `MLS` with *improved trust region directions*;
        if `good` is false, **break**; **end** ▷ `MLS` is inefficient
        update $d_t := (\gamma_{d_2} + \text{rand}) d_t$;
      end while ▷ rand $\in (0, 1]$ is a random value
    else
      while true **do** ▷ until `good` is true
        perform `MLS` with *perturbed random directions*;
        if `good` is false, **break**; **end** ▷ `MLS` is inefficient
      end while
    end if
  end if
  if $n^{\texttt{DS}}_{\texttt{succ}}$ is zero **then** ▷ `MLS` is inefficient in all T_0 calls
    for $i = 1, \cdots, n$ **do**
      compute $\texttt{dz}_i := Z_{:i} - Z_{:b}$; ▷ the difference of the old points
      find $\mathcal{I}_i := \{j \mid \texttt{dz}_j \neq 0 \text{ and } (Z_{:b})_j \neq 0\}$;
      compute $\beta_i^t := \min\{|(Z_{:b})_j / \texttt{dz}_j| \mid j \in \mathcal{I}_i\}$;
    end for
    generate two random values μ_1 and μ_2 satisfying $0 < \mu_1 < \mu_2 < 1$;
    reconstruct $[\underline{\alpha}, \overline{\alpha}] := [\gamma_a \mu_1 \beta^t_{\min}, \gamma_a \mu_2 \beta^t_{\min}]$ with $\beta^t_{\min} := \min_{i=1,\cdots,n} \beta_i^t$;
  end if
end for

function evaluations for each successful iteration. In the next section, we use this discussion to determine how the complexity results for an implemented version of VRDFON-basic will change compared to the complexity results of VRDFON-basic.

1.9 VRDFON – an improved version of VRDFON-basic

As mentioned earlier, VRDFON is an implemented version of VRDFON-basic. It uses the DS and MLS procedures instead of their basic versions, and uses all practical enhancements to achieve an $\varepsilon = \sqrt{\omega}$-approximate stationary point.

If step sizes are too small, VRDFON-basic may end up in strongly non-convex regions before a minimizer is found. To overcome this drawback, we change line 9 in VRDFON-basic to $\delta_{k+1} := \max(\delta_k, \sqrt{\underline{\alpha}\overline{\alpha}})$ if the interval $[\underline{\alpha}, \overline{\alpha}] \subseteq (0, \infty)$ has already been found.

2 Codes compared

The VRDFON package is publicly available at [16]. We compare it with:

• VRBBO – an efficient random algorithm by Kimiaei & Neumaier [18]. It can be downloaded from

https://www.mat.univie.ac.at/~neum/software/VRBBO/.

• NOMAD (version 3.9.1) , obtained from

https://www.gerad.ca/nomad

is a Mesh Adaptive Direct Search algorithm (MADS) [1]. NOMAD1 uses the following option set

opts = nomadset('max_eval',nfmax,'max_iterations',2*nfmax,
'min_mesh_size','1e-008','initial_mesh_size','10')

while NOMAD2 uses the following option set

opts = nomadset('max_eval',nfmax,'max_iterations',2*nfmax,
'min_mesh_size','1e-008','initial_mesh_size','10','model_search','0').

• SNOBFIT, obtained from

http://www.mat.univie.ac.at/~neum/software/snobfit,

is a combination of a branching strategy to enhance the chance of finding a global minimum with a sequential quadratic programming method based on fitted quadratic models to have good local properties by Huyer & Neumaier [14].

• GRID – a grid algorithm for bound constrained optimization by Elster & Neumaier [10], available at

http://www.mat.univie.ac.at/~neum/software/GRID.

This solver was originally written in Fortran with auxiliary routines that are no longer available. We reimplemented GRID in Matlab. The trust region subproblem is solved with minq8 [15].

• UOBYQA and NEWUOA, obtained from

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

are model-based solvers by Powell [23, 24].
• BFO – a trainable stochastic derivative free solver for mixed integer bound-constrained optimization by Porcelli & Toint [22], available at

https://github.com/m01marpor/BFO.
• DSPFD – a direct search Matlab code for derivative-free optimization by Gratton et al. [11], available at

pages.discovery.wisc.edu/%7Ecroyer/codes/dspfd_sources.zip.
• MCS – a deterministic global optimization by a multilevel coordinate search by Huyer & Neumaier [13], downloaded from

https://www.mat.univie.ac.at/~neum/software/mcs/.
It used the following parameters:

iinit $= 1$; nfMCS $=$ nfmax; smax $= 5n + 10$; stop $= 3n$; local $= 50$; gamma $=$ eps; hess $=$ ones(n, n); prt $= 0$.

• BCDFO – a deterministic model-based trust region algorithm for derivative-

free bound constrained minimization by Gratton et al. [12], obtained from Anke Troeltzsch (personal communication).

• FMINUNC – a deterministic quasi Newton or trust region algorithm, available at the Matlab Optimization Toolbox at

https://ch.mathworks.com/help/optim/ug/fminunc.html.
The following options set was used:

```
opts = optimoptions(@fminunc),'Algorithm','quasi-newton'
'Display', 'Iter','MaxIter',Inf,'MaxFunEvals', limits.nfmax
'TolX', 0,'TolFun',0,'ObjectiveLimit',-1e-50);
```

• SDBOX – a derivative-free algorithm for bound constrained optimization problems discussed in [21], downloaded from

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

• CMAES, obtained from

http://cma.gforge.inria.fr/count-cmaes-m.php?Down=cmaes.m,

is the stochastic covariance matrix adaptation evolution strategy by Auger & Hansen [2]. We used CMAES with the tuning parameters

$$\begin{aligned}&\texttt{oCMAES.MaxFunEvals} = \texttt{nfmax},\ \texttt{oCMAES.DispFinal} = 0,\\ &\texttt{oCMAES.LogModulo} = 0,\ \texttt{oCMAES.SaveVariables} = 0,\\ &\texttt{oCMAES.DispModulo} = 0,\ \texttt{oCMAES.MaxIter} = \texttt{nfmax},\\ &\texttt{oCMAES.Restarts} = 7.\end{aligned}$$

• LMMAES by Loshchilov et al. [20], fMAES by Beyer [5], BiPopMAES by Beyer & Sendhoff [6], obtained from

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

are three various covariance matrix adaptation evolution strategies.

• subUOBYQA, subNEWUOA and subNMSMAX are UOBYQA, NEWUOA and NMSMAX, respectively, in the random subspace generated by the columns of $S := 2 * \mathrm{rand}(n, m) - 1$ (in Matlab) to handle problems in medium and high

dimensions. Here n is the dimension of the problem and m is the subspace dimension. Using these solvers, we recursively minimize the problem $\tilde{f}(x + \Delta Sz)$ with respect to $z \in \mathbb{R}^n$. Let Δ be the trust region radius, which is initially one, and let $0 < \zeta < 1$ be a parameter for the decrease of the function value. If $\tilde{f}(x) > \tilde{f}(x + \Delta Sz) - \zeta\Delta^2$ holds, the trial point $x + \Delta Sz$ can be accepted as a new point and Δ is expanded to $\Delta = \max(\Delta, \|Sz\|)$; otherwise Δ is reduced to $\Delta = \frac{1}{2}\min(\Delta, \|Sz\|)$. For each call to these solvers, the maximum number of function evaluations is $\max(\lceil n/m \rceil, 10m)$. According to our findings, the best value for the subspace dimension m is 10. For $m < 10$ the efficiency and robustness of these subspace solvers decrease when they are used to solve problems in medium and high dimensions, while for $10 < m \leq 30$ the number of sample points to construct the quadratic models increases, making each call to the original solver (`UOBYQA`, `NEWUOA` and `NMSMAX`) costly.

Unfortunately, software for `FDLM` by Berahas et al. [4] and `STRRS` by Chen [7] was not available to us.

To obtain the improved trust region directions discussed in Section 1.6, we solve the trust region subproblem by `minq8` [15], available at

`https://www.mat.univie.ac.at/~neum/software/minq/`.

`minq8` requires the tuning parameters `minqmax` $= 10000$ and `minqeps` $= 10^{-8}$.

3 Tuning of `VRDFON`

In this section, we perform a testing and tuning for five important tuning parameters are `comBound` (kind of the complexity results), `model` (model-based?), R (number of scaled random directions in `MLS`), C (number of random approximate coordinate directions), and T_0 (number of times `MLS` is called by `DS`). Other tuning parameters were chosen fixed as they did not change the efficiency and robustness of our solver. Accordingly, for a testing and tuning for small scale problems ($1 < n \leq 30$), we consider the 30 versions of `VRDFON` given in Table 1. Then we chose the four best versions of `VRDFON` and ran them for medium scale problems ($31 < n \leq 300$) and large scale problems ($300 < n \leq 5000$) to select the best version of `VRDFON` to compare with the other solvers. Finding the optimal tuning parameters for `VRDFON` and the other goal, such as the development of an effective mixed integer solver, is still a work in progress [19].

As in [17], VRDFON with $R_m T_0 \sim n$ has numerically better performance than VRDFON with $R_m T_0 \sim 1$, while VRDFON with $R_m T_0 \sim 1$ has a complexity factor n better than VRDFON with $R_m T_0 \sim n$. In fact, VRDFON with $R_m T_0 \sim n$ has the same complexity factor as in the deterministic case.

Table 1 contains the values of the tuning parameters of VRDFON, leading to the best numerical performance for VRDFON shown in Figures 2–4.

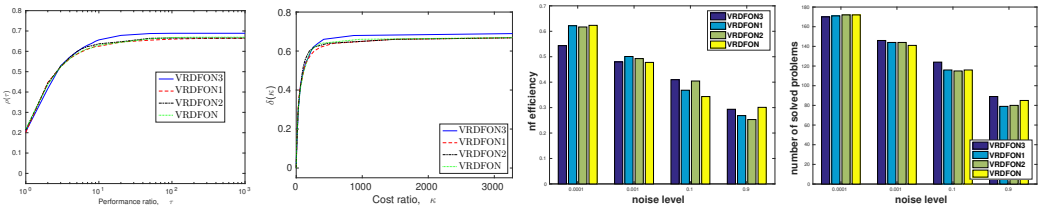
Table 1: The values of tuning parameters of VRDFON

$\overline{m} = 230$	$m_{\max} = \min(0.5n(n+3), \overline{m})$	$\delta_{\min} = 0$	$\delta_{\max} = 1$	$\gamma_Z = 100$	$\gamma_{\rm rd} = 10^{-30}$
$\gamma_{d_1} = 2$	$Q = 1.5$	$d_{\min} = 10^{-4}$	$\gamma_v = 100$	$\gamma = 10^{-6}$	$\gamma_e = 3$
$d_{\max} = 10^3$	$\underline{\alpha}_{\rm init} = 0.01$	$\overline{\alpha}_{\rm init} = 0.99$	$\gamma_\kappa = 0.85$	$\gamma_{d_2} = 0.5$	$\gamma_p = 0.25$
$\gamma_a := 10^{-5}$	$\overline{\alpha}_{\min} = 10^{-3} * \text{rand}$	$R_m = n$	$T_0 = 5$	model $= 1$	comBound $= 2$

C	T_0	R_m	`comBound`	`model`	complexity	chosen as solver
n	n	–	0	0	no	no
n	n	–	0	1	no	no
n	10	–	0	0	no	no
n	10	–	0	1	no	no
n	5	–	0	0	no	no
n	5	–	0	1	no	no
n	2	–	0	0	no	no
n	2	–	0	1	no	`VRDFON1`
10	n	–	0	0	no	no
10	n	–	0	1	no	no
5	n	–	0	0	no	`VRDFON2`
5	n	–	0	1	no	no
2	n	–	0	0	no	no
2	n	–	0	1	no	no
$\lceil n/2 \rceil$	n	$\lceil n/2 \rceil$	1	0	yes	no
$\lceil n/2 \rceil$	n	$\lceil n/2 \rceil$	1	1	yes	`VRDFON3`
–	n	n	2	0	yes	no
–	n	n	2	1	yes	no
–	10	n	2	0	yes	no
–	10	n	2	1	yes	no
–	5	n	2	0	yes	no
–	5	n	2	1	yes	`VRDFON`
–	2	n	2	0	yes	no
–	2	n	2	1	yes	no
–	n	10	2	0	yes	no
–	n	10	2	1	yes	no
–	n	5	2	0	yes	no
–	n	5	2	1	yes	no
–	n	2	2	0	yes	no
–	n	2	2	1	yes	no

Figure 1: 30 variants of tuning of `VRDFON`

For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-1}, 0.9\}$ and small scale problems ($1 < n \leq 30$), Figure 2 shows in its subfigures the cumulative (over all noise levels used) performance and data profiles in terms of the number of function evaluations and the other two plots show (the `nf` efficiency versus the noise level ω and the number of solved problems versus the noise level ω). The result of this comparison is that `VRDFON3` is slightly more robust than the others, since all the proposed directions are used. For this version, the complexity results are valid since random scaled directions were tried. `VRDFON1` and `VRDFON2` are slightly more efficient than `VRDFON3` and `VRDFON`, except for high noise. In fact, using scaled random directions guarantees the existence of complexity results and increases the robustness of our solver under low to high noise and the efficiency of our solver only under high noise. Moreover, since `VRDFON1`, `VRDFON3`, and `VRDFON` are model-based, it is confirmed that `VRDFON` is effective when it is model-based.

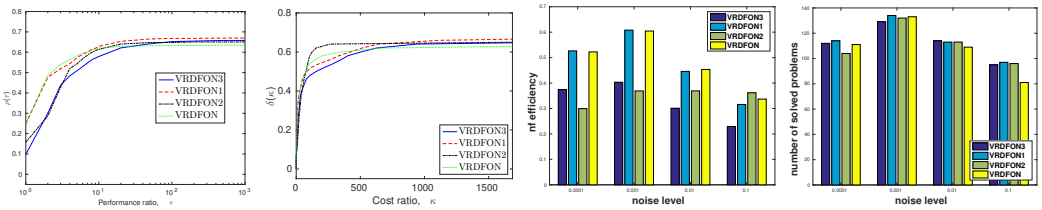


Performance profile **Data profile** **nf efficiency** **# solved problems**

Figure 2: For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-1}, 0.9\}$ and small dimensions $1 < n \leq 30$. Data profile $\delta(\kappa)$ in dependence of a bound κ on the cost ratio while performance profile $\rho(\tau)$ in dependence of a bound τ on the performance ratio. Problems solved by no solver are ignored. Here '# solved problems' counts the number of solved problems.

For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and medium scale problems ($30 < n \leq 300$), Figure 3 shows in its subfigures the cumulative (over all noise levels used) performance and data profiles in terms of the number of function evaluations and the other two plots show (the `nf` efficiency versus the noise level ω and the number of solved problems versus the noise level ω). We conclude from these subfigures that `VRDFON1` and `VRDFON` are slightly more efficient and robust than `VRDFON2` and `VRDFON3`.

For the noise levels $\omega \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ and large scale problems ($300 < n \leq 5000$), Figure 4 shows in its subfigures, the cumulative (over all noise levels used) performance and data profiles in terms of the number of function evaluations and the other two plots show (the `nf` efficiency versus the noise level ω and the number of solved problems versus the noise level ω). We

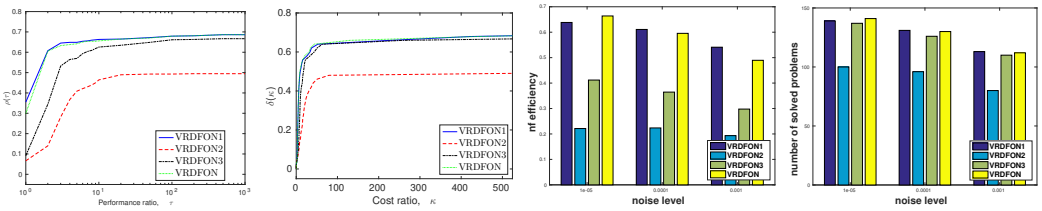


Performance profile **Data profile** **nf efficiency** **# solved problems**

Figure 3: For the noise levels $\omega \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and medium dimensions $30 < n \leq 300$. Other details as in Figure 2.

conclude from these subfigures that `VRDFON1` and `VRDFON` are more efficient and robust than `VRDFON2` and `VRDFON3`.

As a result, we choose `VRDFON` as default version to compare with the other solvers for problems in low to high dimensions. Note that our complexity results hold for this version.



Performance profile **Data profile** **nf efficiency** **# solved problems**

Figure 4: For the noise levels $\omega \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ and large dimensions $300 < n \leq 5000$. Other details as in Figure 2.

References

[1] M. A. Abramson, C. Audet, G. Couture, J. E. Dennis, Jr., S. Le Digabel and C. Tribes. The NOMAD project. Software available at `https://www.gerad.ca/nomad/`.

[2] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *2005 IEEE Congress on Evolutionary Computation*. IEEE (2005).

[3] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Convergence of trust-region methods based on probabilistic models. *SIAM J. Optim* **24** (January 2014), 1238–1264.

[4] A. S. Berahas, R. H. Byrd, and J. Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.* **29** (January 2019), 965–993.

[5] H. G. Beyer. Design principles for matrix adaptation evolution strategies (2020).

[6] H. G. Beyer and B. Sendhoff. Simplify your covariance matrix adaptation evolution strategy. *IEEE Trans. Evol. Comput.* **21** (October 2017), 746–759.

[7] R. Chen. *Stochastic Derivative-Free Optimization of Noisy Functions*. PhD thesis, Lehigh University (2015). Theses and Dissertations. 2548.

[8] C. Davis. Theory of positive linear dependence. *Amer. J. Math.* **76** (October 1954), 733.

[9] P. Deuflhard and G. Heindl. Affine invariant convergence theorems for newton's method and extensions to related methods. *SIAM J. Numer. Anal.* **16** (February 1979), 1–10.

[10] C. Elster and A. Neumaier. A grid algorithm for bound constrained optimization of noisy functions. *IMA J. Numer. Anal.* **15** (1995), 585–608.

[11] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim* **25** (January 2015), 1515–1541.

[12] 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** (October 2011), 873–894.

[13] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *J. Glob. Optim.* **14** (1999), 331–355.

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

[15] W. Huyer and A. Neumaier. MINQ8: general definite and bound constrained indefinite quadratic programming. *Comput. Optim. Appl.* **69** (October 2017), 351–381.

[16] M. Kimiaei. (2022). The VRDFON solver. Software available at `https://github.com/GS1400/VRDFON`.

[17] M. Kimiaei. VRDFON – line search in noisy unconstrained derivative-free optimization. (May 2022). https://github.com/GS1400/VRDFON

[18] M. Kimiaei and A. Neumaier. Efficient unconstrained black box optimization. *Math. Program. Comput.* http:/doi.org/10.1007/s12532-021-00215-9 (February 2022).

[19] M. Kimiaei and A. Neumaier. Testing and tuning optimization algorithm. Preprint, Vienna University, Fakultät für Mathematik, Universität Wien, Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria (2022).

[20] I. Loshchilov, T. Glasmachers, and H. G. Beyer. Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Trans. Evol. Comput.* **23** (April 2019), 353–358.

[21] S. Lucidi and M. Sciandrone. A derivative-free algorithm for bound constrained optimization. *Comput. Optim. Appl.* **21** (2002), 119–142.

[22] M. Porcelli and P. Toint. Global and local information in structured derivative free optimization with BFO. *arXiv: Optimization and Control* (2020).

[23] M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Math. Program.* **92** (May 2002), 555–582.

[24] M. J. D. Powell. Developments of NEWUOA for minimization without derivatives. *IMA. J. Numer. Anal.* **28** (February 2008), 649–664.