*Chin. Ann. Math. Ser. B*
44(5), 2023, 719–734
DOI: 10.1007/s11401-023-0040-y

**Chinese Annals of Mathematics, Series B**


# A Derivative-Free Optimization Algorithm Combining Line-Search and Trust-Region Techniques*

Pengcheng XIE[1] Ya-xiang YUAN[1]

**Abstract** The speeding-up and slowing-down (SUSD) direction is a novel direction, which is proved to converge to the gradient descent direction under some conditions. The authors propose the derivative-free optimization algorithm SUSD-TR, which combines the SUSD direction based on the covariance matrix of interpolation points and the solution of the trust-region subproblem of the interpolation model function at the current iteration step. They analyze the optimization dynamics and convergence of the algorithm SUSD-TR. Details of the trial step and structure step are given. Numerical results show their algorithm's efficiency, and the comparison indicates that SUSD-TR greatly improves the method's performance based on the method that only goes along the SUSD direction. Their algorithm is competitive with state-of-the-art mathematical derivative-free optimization algorithms.

**Keywords** Nonlinear optimization, Derivative-Free, Quadratic model, Line-Search, Trust-Region
**2000 MR Subject Classification** 90C56, 90C30, 65K05, 90C90

## 1 Introduction and Motivation

### 1.1 Introduction

Most mathematical optimization methods need to use the derivative of the objective function. However, in practice, the objective function is sometimes costly to compute, and its derivatives are not available. The optimization of this type is called derivative-free optimization. Derivative-free optimization methods are numerical optimization methods in which no derivatives are required. For more details, one can refer to the paper of Zhang [26] and the review of Larson, Menickelly and Wild [11], the book of Conn, Scheinberg and Vicente [5] and Audet and Hare's book [2] for the introduction and review of derivative-free optimization. Derivative-free methods can be of different types, which mainly contain direct-search methods (see [10, 13, 21–22]), model-based methods (see [4, 15–18, 24–25]), line-search methods (see [6–7, 19–20]), hybrid methods [8] and heuristic algorithms (see [3, 23]). Derivative-free methods have a deep impact on the development of many problems raised in both of the theoretical and practical fields.

Manuscript received March 20, 2023. Revised March 28, 2023.
[1]State Key Laboratory of Scientific/Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100190, China.
E-mail: xpc@lsec.cc.ac.cn yyx@lsec.cc.ac.cn
*This work was supported by the National Natural Science Foundation of China (No. 12288201).

### 1.2 Problem formulation and motivation

Consider the unconstrained optimization problem

$$\min_{\boldsymbol{x}\in\Re^n} f(\boldsymbol{x}), \tag{1.1}$$

where $f : \Re^n \to \Re$ is a scalar objective function without any derivative information. As we know, both line-search and trust-region methods are widely used for optimization problems. The line-search method looks for an (optimal) stepsize along the current search direction (the stepsizes for each of the group of points for algorithms based on SUSD direction). However, when the search direction is not very effective, the line-search method may result in slow convergence. In contrast, the trust-region method can adapt better to different search directions by constructing a local quadratic model function in a trust region to find a model's minimizer. This approach can mitigate the problem of ineffective search directions in the line-search method and perform better for some problems. However, solving the trust-region subproblem requires higher computational cost. Nocedal and Yuan [14] discussed the combination of the line-search method and trust-region method in the optimization using the gradient information of the objective functions.

Observing the flow of the moving and iteration of a group of queried points along the PCA-based speeding-up and slowing-down (SUSD for short) direction (see [1]), we prefer to call the iteration process of a group of points the large-scale step. The interior structure of the iteration points does not obtain a modification, especially in the large-scale step.

Our motivation is to modify the structure of the group of iteration points and look for a minimizer of a local interpolation model before the next step going along the SUSD direction by selecting a better point in the trust region and dropping one. This new point can modify or even reverse the iteration direction of the group of points, especially in the case where such a direction is deviated. We consider modifying points according to the interpolation model in addition to the large-scale moving, which combines the line-search and trust-region techniques.

### 1.3 Organization

This article is organized as follows. We give our algorithm of the combination of SUSD direction and trust-region interpolation in Section 2. Section 3 gives the convergence analysis of SUSD-TR by giving the optimization dynamics and the convergence of the line-search direction in the algorithm SUSD-TR. In Section 4, we show more details of the trial step and the structure step contained in the trust-region step in our algorithm. At the end, we give the conclusions.

## 2 Combination of SUSD Direction and Trust-Region Interpolation

The proposed algorithm mainly has two steps to update the iteration points: The trust-region step and the line-search step. The trust-region step is derived by solving the trust-region subproblem of the interpolation model function determined by the interpolation agents at each step. The line-search step is designed to push the agents along the SUSD direction, chosen as $\boldsymbol{v}_1$ in the following discussion. Notice that we use the notation agent, which is exactly the sample/iteration point.

Suppose that there are $m$ virtual search agents, where each agent acts as a candidate solution $\boldsymbol{x}_i \in \Re^n$. We request that $m \geq n$, where $n$ is the dimension of the optimization problem (1.1). We define the covariance matrix $\boldsymbol{C} \in \Re^{n\times n}$ as

$$\boldsymbol{C} = \sum_{i=1}^{m} (\boldsymbol{x}_i - \boldsymbol{x}_c)(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top, \tag{2.1}$$

where $\boldsymbol{x}_c = \frac{1}{m}\sum\limits_{i=1}^{m} \boldsymbol{x}_i$ is the center of the agents. Let $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ denote the eigenvectors of the covariance matrix $\boldsymbol{C}$ associated with the eigenvalues $\mu_1, \cdots, \mu_n$, ordered from the smallest eigenvalue $\mu_1$ to the largest eigenvalue $\mu_n$ (suppose that $\mu_1 \neq 0$). The vector $\boldsymbol{v}_1$ is the SUSD direction (see [1]). Algorithm 1 gives the pseudocode description of the derivative-free optimization algorithm SUSD-TR, which is based on the SUSD direction and the interpolation and trust-region techniques. In Algorithm 1, $\overline{f}^{(k)}$ is the smallest function value from the interpolation points/agents at the $k$-th step. In Algorithm 1, $k$ is applied to express the iteration, and we write some parameters as the function of $k$.

SUSD-TR is a derivative-free optimization algorithm based on the combination of line-search and trust-region techniques, since the group of points will go along the direction $\boldsymbol{v}_1$, which is line-search type, and then the algorithm will solve the trust-region subproblem to obtain the modification, which forms the loop of the algorithm.

There are some advantages of SUSD-TR. Firstly, it can be calculated distributedly or parallelly if we transport the data (including the function values and so on) among different computational nodes, since the algorithm evaluates the function values at different $m$ points simultaneously at the line-search step. This reduces the time cost in the function evaluation and is especially efficient for time-cost evaluation problems. Secondly, Algorithm 1 is not based on the traditional gradient estimation, so it can still be used without the explicit gradient estimation. Thirdly, we can give the optimization dynamics, which is not usual in derivative-free optimization but is novel and important. Last but not the least, the points can be in a more flexible region than those providing the traditional finite difference estimation.

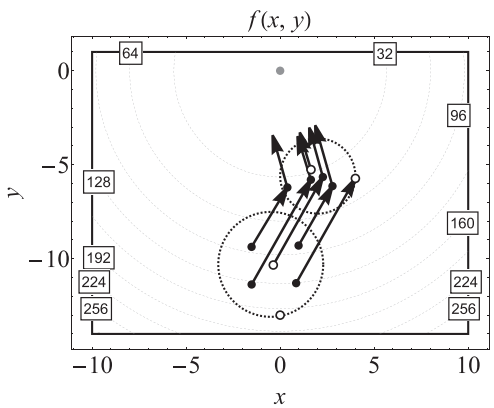



Figure 1 Illustration of the general framework of SUSD-TR for a 2-dimensional problem.

Figure 1 illustrates the process of the algorithm SUSD-TR, where the white point denotes the dropped point $\boldsymbol{x}_d$ at each step. At the $k$-th step, the under-determined quadratic interpolation

**Algorithm 1** SUSD-TR

**input**: Number of agents $m$, initials $\boldsymbol{x}_i^{(0)}, i = 1, \cdots, m$, obtain the stepsize coefficient $\beta$, termination parameters $P, \varepsilon$; $k = 0$

**while** $\left|\overline{f}^{(k)} - \frac{1}{P}\sum_{h=1}^{P}\overline{f}^{(k-h)}\right| < \varepsilon$ **do**

  compute $\boldsymbol{C}^{(k)}$ and $\boldsymbol{v}_1^{(k)}$ by the traditional PCA algorithm

  **for** $i = 1, \cdots, m$ **do**

    evaluate $\alpha_i^{(k)} = f_i^{(k)}$, note: $f_i^{(k)} := f(\boldsymbol{x}_i^{(k)})$

  **end for**

  $\Delta_k = \max_i(\|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_c^{(k)}\|_2)$.

  **if** $\Delta_k > \kappa\Delta_{k-1}$ **then**

    **structure step**: Replace the farthest point $\boldsymbol{x}_d^{(k)}$ by $\boldsymbol{x}_{\text{new}}^{(k)}$ according to Algorithm 2

  **else**

    **model improvement step**: Check and improve the interpolation poisedness of the interpolation set by calling the model impovement step, i.e., Algorithm 6.3 in the book of Conn, Scheinberg and Vicente [5]

    generate the current interpolation set $\mathcal{X}_k$ based on the $m$ newest agents, and construct the linear interpolation model function $L_k(\boldsymbol{x})$ or the quadratic model $Q_k(\boldsymbol{x})$ according to (2.2) or (2.3)

    **trial step**: Replace $\boldsymbol{x}_d^{(k)}$ by $\boldsymbol{x}_{\text{new}}^{(k)}$ after obtaining $\boldsymbol{x}_{\text{new}}^{(k)}$ by solving the trust-region subproblem

$$\min_{\boldsymbol{x}} L_k(\boldsymbol{x}) \text{ or } Q_k(\boldsymbol{x}),$$
$$\text{subject to } \|\boldsymbol{x} - \boldsymbol{x}_c^{(k)}\|_2 \le \Delta_k$$

    by the truncated conjugate gradient method and update the radius of the trust region $\Delta_k$. Notice that $\boldsymbol{x}_d^{(k)}$ here is the point that has the largest function value among the iteration points at the current step, and it is exactly replaced by $\boldsymbol{x}_{\text{new}}^{(k)}$

  **end if**

  compute $\overline{f}^{(k)} := \min_i f_i^{(k)}$

  **for** $i = 1, \cdots, m$ **do**

    compute $\alpha(\boldsymbol{x}_i^{(k)}) = \beta[1 - \exp(\overline{f} - f(\boldsymbol{x}_i^{(k)}))]$

    **line-search step**: Update $\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} + \alpha(\boldsymbol{x}_i^{(k)})\boldsymbol{v}_1^{(k)}$

  **end for**

  $k := k + 1$

**end while**

**return** $\boldsymbol{x}^* = \boldsymbol{x}_i$, where $f(\boldsymbol{x}_i)$ is the smallest function value of the iteration points at the last step

model $Q_k$ is obtained by solving the subproblem

$$\begin{aligned} &\min_{Q\in\mathcal{Q}} \|\nabla^2 Q\|_F^2 \\ &\text{subject to } Q(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad \forall\ \boldsymbol{x}_i \in \mathcal{X}_k, \end{aligned} \tag{2.2}$$

where the $k$-th set of interpolation points is $\mathcal{X}_k = \{\boldsymbol{x}_1^{(k)}, \cdots, \boldsymbol{x}_m^{(k)}\}$, if the number of the agents/sample points at each iteration is less than $\frac{(n+1)(n+2)}{2}$; and the $k$-th model function $L_k$ or $Q_k$ is obtained by solving the equations

$$\begin{aligned} f(\boldsymbol{x}_i) = L_k(\boldsymbol{x}_i), \quad \forall\ \boldsymbol{x}_i \in \mathcal{X}_k, \\ \text{or } f(\boldsymbol{x}_i) = Q_k(\boldsymbol{x}_i), \quad \forall\ \boldsymbol{x}_i \in \mathcal{X}_k, \end{aligned} \tag{2.3}$$

if we obtain and use the determined linear or quadratic interpolation model separately. In the numerical experiment, we present the result of the SUSD-TR corresponding to the underdetermined quadratic model, since it performs better than using the linear model or determined quadratic model numerically for the test problems. Notice that we omit using the notation $(k)$ here and in the following to express the $k$-th iteration for simplicity.

## 3 Convergence Analysis of SUSD-TR

### 3.1 Optimization dynamics

The dynamic, or we say the gradient flow, of the optimization process in Algorithm 1 can be transformed to

$$\begin{cases} \dot{\boldsymbol{x}}_i = \alpha(\boldsymbol{x}_i)\boldsymbol{v}_1, \quad i = 1, \cdots, d-1, d+1, \cdots, m, \\ \dot{\boldsymbol{x}}_d = (\alpha(\boldsymbol{x}_d) + \varepsilon_1)(\boldsymbol{v}_1 + \boldsymbol{\varepsilon}_2), \end{cases} \tag{3.1}$$

where $\varepsilon_1 \in \Re$ and $\boldsymbol{\varepsilon}_2 \in \Re^n$ are disturbance parameters of $\boldsymbol{x}_d$, which denotes to be updated by $\boldsymbol{x}_{\text{new}}$ using the trust-region technique. The dot in (3.1) denotes the derivative with respect to the continuous time $t$, which refers to the iteration $k$. The stepsize $\alpha : \Re \to \Re$ is an exponential mapping[1] (see [1]), i.e.,

$$\alpha(\boldsymbol{x}_i) = \beta[1 - \exp(\overline{f} - f(\boldsymbol{x}_i))], \quad i = 1, \cdots, m, \tag{3.2}$$

where $\beta \in \Re$ is a positive constant, and $\overline{f}$ is the smallest function value from the interpolation points/agents at the current step. Denote $\alpha(\boldsymbol{x}_i)$ as $\alpha_i$ for simplicity.

Figure 2 explains the dynamic (3.1) with the discrete iteration (time), where $\boldsymbol{x}_d^{(k)}$ is replaced by the $\widehat{\boldsymbol{x}}_d^{(k)}$ in the trust-region step, and then $\widehat{\boldsymbol{x}}_d^{(k)}$ goes to the $\boldsymbol{x}_d^{(k+1)}$ at the line-search step. Therefore, the new iteration point can be expressed as

$$\boldsymbol{x}_d^{(k+1)} = \boldsymbol{x}_d^{(k)} + (\alpha(\boldsymbol{x}_d^{(k)}) + \varepsilon_1)(\boldsymbol{v}_1 + \boldsymbol{\varepsilon}_2),$$

where $\varepsilon_1$ denotes the disturbance of the stepsize and $\boldsymbol{\varepsilon}_2$ denotes the disturbance of the direction of $\boldsymbol{x}_d^{(k)}$. This corresponds to the dynamic (3.1). In the following, we use the continuous dynamics to give the analysis.

[1]The stepsize can also be chosen as a linear mapping, and this paper analyzes the exponential mapping case. The corresponding result of the linear mapping stepsize is the same as the one of the exponential mapping case, so we do not present more details.

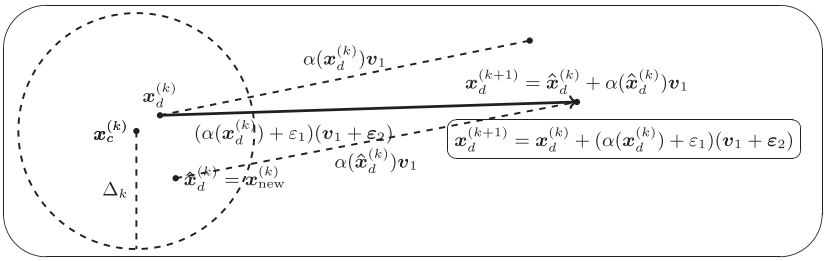



Figure 2 The disturbance parameters $\varepsilon_1$ and $\boldsymbol{\varepsilon}_2$ in (3.1).

**Remark 3.1** For the exponential mapping stepsize designed as (3.2), the point owing the smallest function value from the current step has a zero stepsize, which is different with the linear mapping stepsize. Figure 1 aims to be a demo of the general framework of SUSD-TR.

**Lemma 3.1** *Using* (3.1)*, the dynamic of the* SUSD-TR *direction is*

$$\dot{\boldsymbol{v}}_1 = \Big(\sum_{j=2}^{n} \frac{1}{\mu_1 - \mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top\Big)\Big[\sum_{i=1}^{m}(\alpha_i - \alpha_a)(\boldsymbol{x}_i - \boldsymbol{x}_c) + \varepsilon_1(\boldsymbol{x}_d - \boldsymbol{x}_c) + \Phi\boldsymbol{v}_1\Big], \tag{3.3}$$

*where* $\boldsymbol{v}_j$ *is the* $j$*-th eigenvector of the matrix* $\boldsymbol{C}$ *defined above,*

$$\alpha_a = \frac{1}{m}\sum_{i=1}^{m}\alpha_i + \frac{\varepsilon_1}{m},$$

$\alpha_i = \alpha(\boldsymbol{x}_i)$*, and*

$$\Phi = (\alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)(\boldsymbol{x}_d - \boldsymbol{x}_c)^\top + (\boldsymbol{x}_d - \boldsymbol{x}_c)(\alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)^\top.$$

**Proof** Notice that $\boldsymbol{x}_c = \frac{1}{m}\sum\limits_{i=1}^{m}\boldsymbol{x}_i$. From the dynamic (3.1), we obtain that

$$\dot{\boldsymbol{x}}_c = \alpha_a\boldsymbol{v}_1 + \frac{1}{m}(\alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2),$$

where $\alpha_a = \frac{1}{m}\sum\limits_{i=1}^{m}\alpha_i + \frac{\varepsilon_1}{m}$. Taking the time derivative of (2.1), we derive that

$$\begin{aligned}\dot{\boldsymbol{C}} &= \sum_{i=1}^{m}(\alpha_i - \alpha_a)[\boldsymbol{v}_1(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top + (\boldsymbol{x}_i - \boldsymbol{x}_c)\boldsymbol{v}_1^\top] - \frac{\alpha_d + \varepsilon_1}{m}\sum_{i=1}^{m}[\boldsymbol{\varepsilon}_2(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top + (\boldsymbol{x}_i - \boldsymbol{x}_c)\boldsymbol{\varepsilon}_2^\top]\\ &\quad + (\varepsilon_1\boldsymbol{v}_1 + \alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)(\boldsymbol{x}_d - \boldsymbol{x}_c)^\top + (\boldsymbol{x}_d - \boldsymbol{x}_c)(\varepsilon_1\boldsymbol{v}_1 + \alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)^\top\\ &= \sum_{i=1}^{m}(\alpha_i - \alpha_a)[\boldsymbol{v}_1(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top + (\boldsymbol{x}_i - \boldsymbol{x}_c)\boldsymbol{v}_1^\top] + (\varepsilon_1\boldsymbol{v}_1 + \alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)(\boldsymbol{x}_d - \boldsymbol{x}_c)^\top\\ &\quad + (\boldsymbol{x}_d - \boldsymbol{x}_c)(\varepsilon_1\boldsymbol{v}_1 + \alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2)^\top.\end{aligned} \tag{3.4}$$

Besides, it holds that

$$\dot{\boldsymbol{C}}\boldsymbol{v}_1 + \boldsymbol{C}\dot{\boldsymbol{v}}_1 = \dot{\mu}_1\boldsymbol{v}_1 + \mu_1\dot{\boldsymbol{v}}_1$$

based on $\boldsymbol{C}\boldsymbol{v}_1 = \mu_1\boldsymbol{v}_1$. We have

$$\boldsymbol{v}_j^\top\dot{\boldsymbol{C}}\boldsymbol{v}_1 + \boldsymbol{v}_j^\top\boldsymbol{C}\dot{\boldsymbol{v}}_1 = \dot{\mu}_1\boldsymbol{v}_j^\top\boldsymbol{v}_1 + \mu_1\boldsymbol{v}_j^\top\dot{\boldsymbol{v}}_1 \tag{3.5}$$

and the matrix $\dot{\boldsymbol{C}}$ is symmetric. This implies that $\boldsymbol{v}_j^\top\boldsymbol{C}\dot{\boldsymbol{v}}_1 = (\boldsymbol{C}\boldsymbol{v}_j)^\top\dot{\boldsymbol{v}}_1 = \mu_j\boldsymbol{v}_j^\top\dot{\boldsymbol{v}}_1$. In addition, $\boldsymbol{v}_j^\top\boldsymbol{v}_1 = \boldsymbol{v}_1^\top\boldsymbol{v}_j = 0$, we obtain from (3.5) that

$$\boldsymbol{v}_j^\top\dot{\boldsymbol{v}}_1 = \frac{1}{\mu_1 - \mu_j}\boldsymbol{v}_j^\top\dot{\boldsymbol{C}}\boldsymbol{v}_1. \tag{3.6}$$

Since the matrix $\boldsymbol{C}$ is symmetric, we have

$$\dot{\boldsymbol{v}}_1 = \sum_{j=2}^{n}\boldsymbol{v}_j^\top\dot{\boldsymbol{v}}_1\boldsymbol{v}_j. \tag{3.7}$$

Substituting (3.4) in (3.6), and using (3.7) and $\boldsymbol{v}_j^\top\boldsymbol{v}_1(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top\boldsymbol{v}_1 = 0$, we obtain the result.

Let $\alpha_c = \alpha(f(\boldsymbol{x}_c))$ and define the gradient $\nabla\alpha = \nabla\alpha(\boldsymbol{x}_c)$. Then we approximate $\alpha_i = \alpha(\boldsymbol{x}_i)$ using Taylor expansion with respect to the center $\boldsymbol{x}_c$, as

$$\alpha_i - \alpha_c = (\boldsymbol{x}_i - \boldsymbol{x}_c)^\top\nabla\alpha + r_i, \tag{3.8}$$

where $\alpha_c = \alpha(\boldsymbol{x}_c)$ and $r_i = \mathcal{O}(\|\boldsymbol{x}_i - \boldsymbol{x}_c\|_2^2)$. Assume that $f_c = f(\boldsymbol{x}_c)$. Let $\nabla f = \nabla f(\boldsymbol{x}_c)$ be the gradient of the function $f$ at the center $\boldsymbol{x}_c$. We obtain the following lemma.

**Lemma 3.2** *According to* (3.1) *and the Taylor expansion, we obtain that*

$$\begin{aligned}\dot{\boldsymbol{v}}_1 = &\sum_{j=2}^{n}\frac{\mu_j}{\mu_1 - \mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top\nabla\alpha + \boldsymbol{r} + \sum_{j=2}^{n}\frac{\varepsilon_1}{\mu_1 - \mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top(\boldsymbol{x}_d - \boldsymbol{x}_c)\\ &+ \sum_{j=2}^{n}\frac{1}{\mu_1 - \mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top(\Phi\boldsymbol{v}_1),\end{aligned} \tag{3.9}$$

*where*

$$\boldsymbol{r} = \Big(\sum_{j=2}^{n}\frac{1}{\mu_1 - \mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top\Big)\Big[\sum_{i=1}^{m}r_i(\boldsymbol{x}_i - \boldsymbol{x}_c)\Big].$$

**Proof** Suppose that $r_a = \frac{1}{m}\sum\limits_{i=1}^{m}r_i$. The definition of $\alpha_a$ and (3.8) imply that $\alpha_a = \alpha_c + r_a + \frac{\varepsilon_1}{m}$. Then we obtain that

$$\sum_{i=1}^{m}(\alpha_i - \alpha_a)(\boldsymbol{x}_i - \boldsymbol{x}_c) = \boldsymbol{C}\nabla\alpha + \sum_{i=1}^{m}r_i(\boldsymbol{x}_i - \boldsymbol{x}_c), \tag{3.10}$$

according to (3.8). Substituting (3.10) into (3.3), and using $\boldsymbol{v}_j^\top\boldsymbol{C} = \mu_j\boldsymbol{v}_j^\top$, we obtain (3.9).

Consequently, we give the following lemma, and we denote $\nabla f(\boldsymbol{x}_c)$ by $\nabla f$ as above.

**Lemma 3.3** *According to* (3.1) *and the exponential mapping, we obtain the search dynamics*

$$\begin{cases} \dot{f}_c = \Big\{\dfrac{\varepsilon_1}{m} + \dfrac{\beta}{m}\sum\limits_{i=1}^{m}[1-\exp(\overline{f}-f(\boldsymbol{x}_i(t)))]\Big\}(\nabla f)^\top \boldsymbol{v}_1 \\ \qquad +\Big\{\dfrac{\beta}{m}[1-\exp(\overline{f}-f(\boldsymbol{x}_d))] + \dfrac{\varepsilon_1}{m}\Big\}(\nabla f)^\top \boldsymbol{\varepsilon}_2, \\ \dot{\boldsymbol{v}}_1 = \beta\exp(\overline{f}-f_c)\sum\limits_{j=2}^{n}\dfrac{\mu_j}{\mu_1-\mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top\nabla f + \boldsymbol{r} + \sum\limits_{j=2}^{n}\dfrac{\varepsilon_1}{\mu_1-\mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top(\boldsymbol{x}_d-\boldsymbol{x}_c) \\ \qquad +\sum\limits_{j=2}^{n}\dfrac{1}{\mu_1-\mu_j}\boldsymbol{v}_j\boldsymbol{v}_j^\top(\Phi\boldsymbol{v}_1). \end{cases} \tag{3.11}$$

**Proof** According to the basic computation, it holds that

$$\begin{aligned} \dot{\boldsymbol{x}}_c &= \frac{1}{m}\sum_{i=1}^{m}\alpha_i\boldsymbol{v}_1 + \frac{1}{m}(\alpha_d\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{\varepsilon}_2 + \varepsilon_1\boldsymbol{v}_1) \\ &= \Big\{\frac{\varepsilon_1}{m} + \frac{\beta}{m}\sum_{i=1}^{m}[1-\exp(\overline{f}-f(\boldsymbol{x}_i))]\Big\}\boldsymbol{v}_1 + \Big\{\frac{\beta}{m}[1-\exp(\overline{f}-f(\boldsymbol{x}_d))] + \frac{\varepsilon_1}{m}\Big\}\boldsymbol{\varepsilon}_2. \end{aligned}$$

We substitute the above in $\dot{f}(\boldsymbol{x}_c) = (\nabla f(\boldsymbol{x}_c))^\top\dot{\boldsymbol{x}}_c$, and then we obtain the first equation of (3.11). Besides, it holds that

$$\nabla\alpha(\boldsymbol{x}_c) = \frac{\mathrm{d}\alpha}{\mathrm{d}f}\nabla f(\boldsymbol{x}_c) = \beta\exp(\overline{f}-f(\boldsymbol{x}_c))\nabla f(\boldsymbol{x}_c).$$

Hence we derive the second equation of (3.11) by substituting the above for $\nabla\alpha(\boldsymbol{x}_c)$ in (3.9).

The above analyzes the optimization dynamics of our algorithm.

### 3.2 Convergence of the SUSD-TR Direction

We will first provide the basic definitions of stable, asymptotically stable in the control and dynamic theory, and input-to-state stable, which will be used in our convergence analysis.

**Definition 3.1** (Stable) *$\eta$ is stable, if for any $\overline{\varepsilon} > 0$, there exists a $\overline{\delta}(t,\overline{\varepsilon})$ such that $|\eta(t_0)| < \overline{\delta}$, implying $|\eta(t)| < \overline{\varepsilon}$ for all $t > t_0$.*

**Definition 3.2** (Asymptotically stable) *$\eta$ is asymptotically stable, if it is stable and locally attractive, i.e., there exists a $\overline{\delta}(t_0)$ such that $|\eta(t_0)| < \overline{\delta}$, implying that $\lim\limits_{t\to\infty}\eta(t) = 0$.*

**Definition 3.3** (Class $\mathcal{K}$ function) *A scalar continuous function $g_1(r)$, defined for $r \in [0,a)$, is said to belong to class $\mathcal{K}$ if it is strictly increasing and $g_1(0) = 0$.*

**Definition 3.4** (Class $\mathcal{KL}$ function) *A scalar continuous function $g_2(r,s)$, defined for $r \in [0,a)$ and $s \in [0,\infty)$, is said to belong to class $\mathcal{KL}$ if, for each fixed $s$, the mapping $g_2(r,s)$ belongs to class $\mathcal{K}$ with respect to $r$ and, for each fixed $r$, the mapping $g_2(r,s)$ is decreasing with respect to $s$ and $g_2(r,s) \to 0$ as $s \to \infty$.*

**Definition 3.5** (Input-to-state stable) *If there exists a class $\mathcal{KL}$ function $f_1$ and a class $\mathcal{K}$ function $f_2$ such that for any initial state $\eta(t_0) \in [0,2)$ and any bounded input $\delta(t)$ satisfying*

*$|\delta(t)| \leq U$, the solution $\eta(t)$ of the system $\dot{\eta} = \psi(t, \eta, \delta)$ is defined for all $t > t_0$ and satisfies the inequality*

$$|\eta(t)| \leq f_1(|\eta(t_0)|, t - t_0) + f_2 \sup_{t_0 \leq \tau \leq t} |\delta(\tau)|,$$

*then the system is input-to-state stable with respect to the equilibrium $\eta^* = 0$, the neighborhood $\eta \in [0, 2)$, and a bound on the input $U$.*

The theorem below will be used to establish the input-to-state stability property related to our algorithm.

**Theorem 3.1** (see [9, Theorem 4.19]) *Let $\mathcal{V}(t, \eta) : [0, \infty) \times [0, 2) \to \Re$ be a continuously differentiable function. Let $\alpha_1(\eta), \alpha_2(\eta)$ be class $\mathcal{K}$ functions on $[0, 2)$, and $\rho(|\delta|)$ be a class $\mathcal{K}$ function on $[0, U]$, and $\alpha_3(\eta)$ be a continuous positive definite function on $[0, 2)$. Suppose for all $(t, \eta, \delta) \in [0, \infty) \times [0, 2) \times [-U, U]$, the function $\mathcal{V}$ satisfies*

$$\alpha_1(|\eta|) \leq \mathcal{V}(t, \eta) \leq \alpha_2(|\eta|)$$

*and whenever $|\eta| \geq \rho(|\delta|) > 0$, the total derivative of $\mathcal{V}$ satisfies*

$$\frac{\partial \mathcal{V}}{\partial t} + \frac{\partial \mathcal{V}}{\partial \eta} \psi(t, \eta, \delta) \leq -\alpha_3(\eta).$$

*Then the system $\dot{\eta} = \psi(t, \eta, \delta)$ is input-to-state stable with $\alpha_2^{-1} \circ \alpha_2 \circ \rho$.*

The notations in the definitions and theorem above are independent with the following, and some are only used for simplicity. Assume that $\boldsymbol{g} = \frac{\nabla f}{\|\nabla f\|_2}$. We now give the convergence of the SUSD-TR direction $\boldsymbol{v}_1$ to $-\boldsymbol{g}$. Define $\eta = 1 + \boldsymbol{v}_1^\top \boldsymbol{g}$, where $\eta = 0$ if and only if $\boldsymbol{v}_1 = -\boldsymbol{g}$. Then we obtain the following corollary about $\eta$.

**Corollary 3.1** *According to* (3.1) *and the exponential mapping stepsize, we obtain the search dynamic*

$$\dot{\eta} = \beta \exp(\overline{f} - f_c) \|\nabla f\|_2 \sum_{j=2}^{n} \frac{\mu_j}{\mu_1 - \mu_j} (\boldsymbol{g}^\top \boldsymbol{v}_j)^2 + \delta := \psi(t, \eta, \delta), \tag{3.12}$$

*where*

$$\delta = \boldsymbol{r}^\top \boldsymbol{g} + \boldsymbol{g}^\top \Big( \sum_{j=2}^{n} \frac{\boldsymbol{v}_j \boldsymbol{v}_j^\top}{\mu_1 - \mu_j} \Big) [\varepsilon_1 (\boldsymbol{x}_d - \boldsymbol{x}_c) + \Phi \boldsymbol{v}_1] + \boldsymbol{v}_1^\top \dot{\boldsymbol{g}}.$$

**Proof** The result (3.11) directly derives (3.12) by calculating $\dot{\boldsymbol{v}}_1^\top \boldsymbol{g}$.

The parameter $\delta$ is an external disturbance to the state $\eta$ caused by the nonlinearity of the function that cannot be controlled by the swarm, which contains the higher order components of the function. Notice that $\delta = \boldsymbol{r}^\top \boldsymbol{g} + \boldsymbol{v}_1^\top \dot{\boldsymbol{g}}$ in the case where $\varepsilon_1 = 0$ and $\varepsilon_2 = 0$, which refers to the SUSD case.

The following result shows more details about when and how $\boldsymbol{v}_1$ converges to $-\boldsymbol{g}$.

**Theorem 3.2** *Suppose that $\|\nabla f(\boldsymbol{x}_c)\|_2 > \xi$, where $\xi$ is a positive constant. Then, for* (3.12)*, the equilibrium $\eta = 0$ of the system $\dot{\eta} = \psi(t, \eta, 0)$ is asymptotically stable in which*

*whenever* $\eta(0) \in [0, 2)$*, then* $\eta(t) \to 0$ *as* $t \to \infty$*, where* $t$ *refers to the continuous iteration (time). In addition, for the disturbance satisfying* $|\delta| < \beta \exp(\overline{f} - f_c) M \frac{\mu_1}{\mu_n - \mu_1} \xi$ *for* $M \in (0, 1)$*, the equilibrium* $\eta = 0$ *of system* $\psi(t, \eta, \delta)$ *is locally input-to-state stable.*

**Proof** The proof is as same as that of [1, Theorem 1], and we omit more details here.

## 4 Trial Step and Structure Step

In this section, we will introduce more details about the trial step and structure step contained in the trust-region step in the implementation of Algorithm 1.

### 4.1 Trial step

The trial step is designed as a small-scale modification for SUSD-TR. Furthermore it denotes obtaining the new point by solving the subproblem of the interpolation model in the trust region, and replacing the one owning the largest function value among the iteration points at the current step.

$$\begin{aligned} &\min_{\boldsymbol{x}} L_k(\boldsymbol{x}) \text{ or } Q_k(\boldsymbol{x}) \\ &\text{subject to } \|\boldsymbol{x} - \boldsymbol{x}_c^{(k)}\|_2 \le \Delta_k. \end{aligned}$$

In the convergence theory, the direction $\boldsymbol{v}_1$ holds the probability of not converging. In this case, we say that $\boldsymbol{v}_1$ is failed. The following proposition shows the advantage of the trial step, in the failed case where $\boldsymbol{v}_1$ goes to the gradient ascent direction $\boldsymbol{g}$.

**Proposition 4.1** *Assume that the* $m$ *iteration points at a given step satisfy that the dynamic of the current iteration is*

$$\begin{cases} \dot{\boldsymbol{x}}_i = \boldsymbol{g}, \quad i = 1, \cdots, d-1, d+1, \cdots, m, \\ \dot{\boldsymbol{x}}_d = -\overline{\alpha}_d \boldsymbol{g}, \end{cases} \tag{4.1}$$

*and* $\overline{\alpha}_d > m - 1$*. Then the center point is moving towards the gradient descent direction.*

**Proof** We obtain the dynamics of $\boldsymbol{x}_c$ according to (4.1):

$$\dot{\boldsymbol{x}}_c = \frac{m - 1 - \overline{\alpha}_d}{m} \boldsymbol{g},$$

and then we directly obtain the conclusion.

The above shows that the trial step can pull the center of the group of points away from the gradient ascent direction, into the gradient descent direction. Figure 3 provides an illustration.

We also need to check and improve the interpolation poisedness of the interpolation set before using the model. We aim to consider the posedness of the position or distribution of the iteration/interpolation points to obtain a good interpolation model function by calling the model improvement step, i.e., Algorithm 6.3 in the book of Conn, Scheinberg and Vicente [5]. One can also see [4] for a more general discussion.

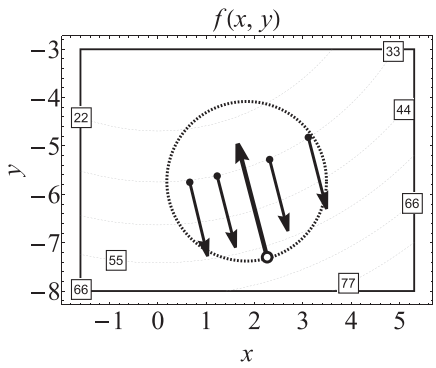



Figure 3 The "antidromic" points leading a gradient descent direction.

### 4.2 Structure step

In addition to the trial step we discussed above, we also design the structure improvement step. The motivation is to enlarge the attraction region of the locally input-to-state stable equilibrium. Recall that when we give the convergence theorem, there is an assumption that the norm of the function's gradient at the center point should be larger than $\xi$. There is a lower bound of $\xi$ in Theorem 3.2.

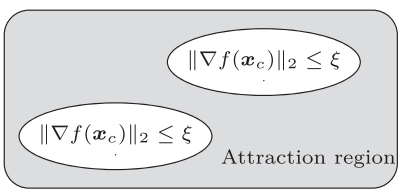



Figure 4 Attraction region ($\xi > \frac{|\delta|(\mu_n-\mu_1)}{\beta\exp(\bar{f}-f_c)M\mu_1}$).

**Remark 4.1** Figure 4 uses the shadow area to denote the attraction region for the locally input-to-state stable equilibrium of Theorem 3.2 ($\mu_1 \neq 0$).

One can see that such lower bound decreases when the condition number of the matrix $\boldsymbol{C}$ is small. In the implementation of the algorithm, we heuristically try to make the eigenvalues close to each other by reducing the largest eigenvalue. As we say, this step is a heuristical step with good numerical results and we do not aim to guarantee the covariance matrix well-conditioned strictly by this step of the algorithm. For the covariance matrix $\boldsymbol{C}$ here, it holds that the radially distributed agents can make $\mu_1$ and $\mu_n$ close to each other, which usually happens in the case where the interpolation set is well-poised, and this is considered in the model improvement step. Besides, suppose that $\boldsymbol{v} \in \Re^n$ is a nonzero eigenvector of $\boldsymbol{C}$, corresponding to the largest eigenvalue $\mu$, then we obtain that

$$\mu = \max_{\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^\top \boldsymbol{C}\boldsymbol{v} = \max_{\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^\top \Big[\sum_{i=1}^{m} (\boldsymbol{x}_i - \boldsymbol{x}_c)(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top\Big]\boldsymbol{v} = \max_{\|\boldsymbol{v}\|_2=1} \sum_{i=1}^{m} [(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top \boldsymbol{v}]^2.$$

Then we can obtain its upper bound and lower bound as

$$\sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{x}_c\|_2^2 \geq \max_{\|\boldsymbol{v}\|_2=1} \sum_{i=1}^{m} [(\boldsymbol{x}_i - \boldsymbol{x}_c)^\top \boldsymbol{v}]^2 \geq \max_i \|\boldsymbol{x}_i - \boldsymbol{x}_c\|_2^2.$$

Accordingly, Agorithm 2 can reduce both the upper bound and the lower bound of the largest eigenvalue of the covariance matrix $\boldsymbol{C}$. We omit the iteration index $k$ here.

**Algorithm 2** Structure step

1: drop the farthest point $\boldsymbol{x}_{\text{far}} := \arg\max_{\boldsymbol{x}_i} \|\boldsymbol{x}_i - \boldsymbol{x}_c t\|_2^2$.

2: add the new point $\boldsymbol{x}_{\text{new}} = \frac{1}{m-1} \sum_{i \neq \text{far}} \boldsymbol{x}_i$ (replace $\boldsymbol{x}_{\text{far}}$).

The above discusses the trial step and structure step in the algorithm.

## 5 Numerical Results

This section presents the numerical results, which include the results of solving two test problems, and the performance profile of solving test problems compared with other derivative-free optimization algorithms. The codes of SUSD-TR can be downloaded from the online repository[2].

### 5.1 Solving two test problems

We implement the Matlab codes[3] to minimize the 2-dimensional Rosenbrock problem

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

and its variant

$$f(x_1, x_2) = (x_2 - x_1^2)^2 + (1 - x_1)^2,$$

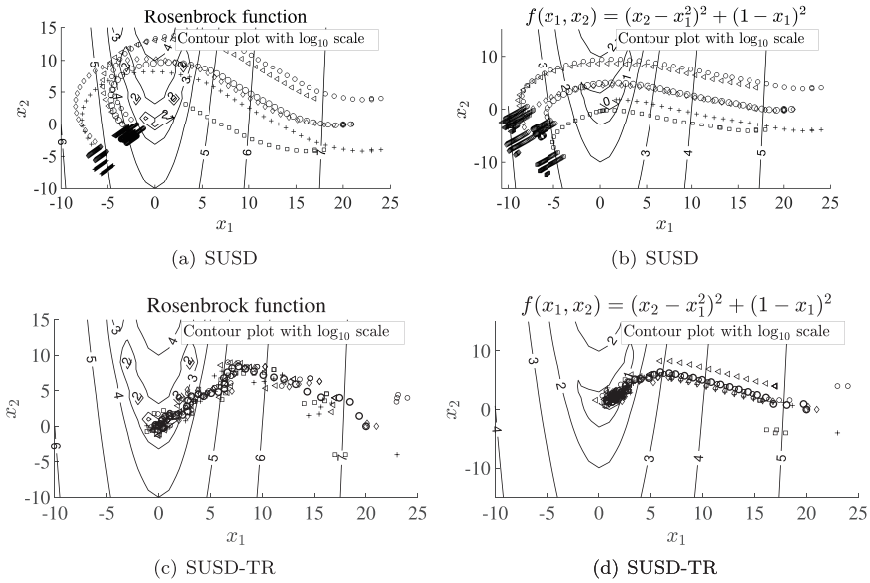
which simulate SUSD and SUSD-TR, where $(x_1, x_2)^\top$ denotes the variable $\boldsymbol{x} \in \Re^2$. Figure 5 illustrates the iteration trajectory for solving the two problems above with SUSD and SUSD-TR. The hollow circle points denote the center points during the iteration. We observe that the SUSD direction is not robust, and $\boldsymbol{v}_1$ does not converge, and finally it misses the minimizer $(1, 1)^\top$. However, if we replace the bad point by $\boldsymbol{x}_{\text{new}}$ on a small scale (applying the SUSD-TR method), we find that it converges to the minimizer. In this example, $\kappa = 1.2$ and we choose 5 points starting from

$$\boldsymbol{x}_1^{(0)} = (20, 0)^\top, \quad \boldsymbol{x}_2^{(0)} = (23, 4)^\top, \quad \boldsymbol{x}_3^{(0)} = (23, -4)^\top, \quad \boldsymbol{x}_4^{(0)} = (17, 4)^\top, \quad \boldsymbol{x}_5^{(0)} = (17, -4)^\top,$$

and the model functions in SUSD-TR are the under-determined quadratic interpolation models. For SUSD, the iteration sometimes does not converge, and the points sometimes go along the gradient ascent direction. However, our SUSD-TR efficiently converges to the minimizer with points basically following the gradient descent direction and using less than half of the function evaluations required by SUSD. A reason is that there will always exist a point going along the

[2]https://github.com/PengchengXieLSEC/SUSD-TR

[3]https://github.com/PengchengXieLSEC/Test-codes-for-SUSD-TR




(a) SUSD (b) SUSD

(c) SUSD-TR (d) SUSD-TR

Figure 5 Solving the 2-dimensional test problems by SUSD and SUSD-TR methods.

gradient descent direction of the quadratic interpolation model at the current iteration, which is also a gradient descent direction of the objective function when the model is accurate (the model is as least fully linear).

The classical examples above show the advantages of our algorithm.

### 5.2 Performance profile

To obtain a further comparison, we compare our algorithm with the derivative-free optimization algorithm based on the SUSD direction (see [1]), Nelder-Mead method (see [13]) and NEWUOA (see [18]). The test problems are taken from classical unconstrained optimization test functions collections, with the performance profile shown in Figure 6.

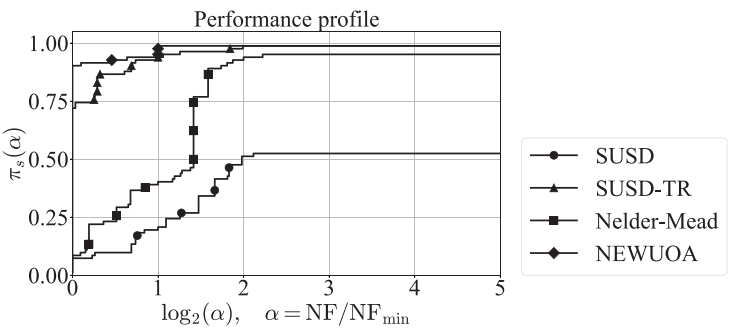



Figure 6 Performance profile

Table 1 Test problems

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ARGLINA | ARGLINA4 | ARGLINB | ARGLINC | ARGTRIG | ARWHEAD | BDQRTIC | BDQRTICP |
| BDALUE | BROWNAL | BROYDN3D | BROYDN7D | BRYBND | CHAINWOO | CHEBQUAD | CHNROSNBZ |
| CHPOWELLB | CHPOWELLS | CHROSEN | COSINECUBE | CURLY10 | CURLY20 | CURLY30 | DIXMAANE |
| DIXMAANF | DIXMAANG | DIXMAANH | DIXMAANI | DIXMAANJ | DIXMAANK | DIXMAANL | DIXMAANM |
| DIXMAANN | DIXMAANO | DIXMAANP | DQRTIC | EDENSCH | ENGVAL1 | ERRINROS | EXPSUM |
| EXTROSNB | EXTTET | FIROSE | FLETCBV2 | FLETCBV3 | FLETCHCR | FMINSRF2 | FREUROTH |
| GENBROWN | GENHUMPS | GENROSE | INDEF | INTEGREQ | LIARWHD | LILIFUN3 | LILIFUN4 |
| MOREBV | MOREBVL | NCB20 | NCB20B | NONCVXU2 | NONCVXUN | NONDIA | NONDQUAR |
| PENALTY1 | PENALTY2 | PENALTY3 | PENALTY3P | POWELLSG | POWER | ROSENBROCK | SBRYBND |
| SBRYBNDL | SCHMVETT | SCOSINE | SCOSINEL | SEROSE | SINQUAD | SPARSINE | SPARSQUR |
| SPHRPTS | SPMSRTLS | SROSENBR | STMOD | TOINTGSS | TOINTTRIG | TQUARTIC | TRIGSABS |

The performance profile (see [12]) depends on the numbers of function evaluations taken by all algorithms in an algorithm set $\mathcal{A}$ to achieve a given accuracy when solving the problems in a given problem set. We define the value

$$f_{\rm acc}^N = \frac{f(\boldsymbol{x}_N) - f(\boldsymbol{x}_{\rm int})}{f(\boldsymbol{x}_{\rm best}) - f(\boldsymbol{x}_{\rm int})} \in [0,1],$$

and the tolerance $\widehat{\tau} \in [0,1]$, where $\boldsymbol{x}_N$ denotes the best point found by the algorithm after $N$ function evaluations, $\boldsymbol{x}_{\rm int}$ denotes the initial input center point, and $\boldsymbol{x}_{\rm best}$ denotes the best known solution given by the compared derivative-free solvers. When $f_{\rm acc}^N \geq 1 - \widehat{\tau}$, we say that the solution reaches the accuracy $\widehat{\tau}$.

We denote $N_{s,p} = \min\{n \in \mathbb{N},\ f_{\rm acc}^n \geq 1 - \widehat{\tau}\}$. $T_{s,p} = 1$, if $f_{\rm acc}^N \geq 1 - \widehat{\tau}$ for some $N$; and $T_{s,p} = 0$, if the solution of solver $s$ fails to reach the accuracy $\widehat{\tau}$ on problem $p$ before the termination. Besides, we define

$$r_{s,p} = \begin{cases} \dfrac{N_{s,p}}{\min\{N_{\widetilde{s},p} : \widetilde{s} \in \mathcal{A}, T_{\widetilde{s},p} = 1\}}, & \text{if } T_{s,p} = 1, \\ +\infty, & \text{if } T_{s,p} = 0, \end{cases}$$

where $s$ is the corresponding solver or algorithm. For the given tolerance $\widehat{\tau}$ and a certain problem $p$ in the problem set $\mathcal{P}$, the parameter $r_{s,p}$ shows the ratio of the number of the function evaluations using the solver $s$ divided by that using the fastest algorithm on the problem $p$, which exactly refers to the $\alpha$. In Figure 6, NF denotes the number of function evaluations, and $\rm NF_{min}$ denotes the smallest one corresponding to the fastest solver for the test problem.

In the performance profile, $\pi_s(\alpha) = \frac{1}{|\mathcal{P}|}|\{p \in \mathcal{P} : r_{s,p} \leq \alpha\}|$, where $|\cdot|$ denotes the cardinality. Generally, $\pi_s(\alpha)$ is the fraction of problems with a performance ratio $r_{s,p}$ bounded by $\alpha$. A higher value of $\pi_s(\alpha)$ represents solving more problems successfully. All of the algorithms are implemented in Python.

Our algorithm begins with $m = 2n+1$ randomly selected initial points, and the parameters are set to $\beta = 1, P = 5, \varepsilon = 10^{-6}, \kappa = 1.2$ and $\widehat{\tau} = 0.1$ separately. The model functions in SUSD-TR are the under-determined quadratic interpolation models. We observe from Figure 6 that SUSD-TR is much more efficient than the Nelder-Mead method and as robust as NEWUOA for solving these test problems, and its performance is significantly better than that of the method based on the SUSD direction (i.e., the pure line-search type).

In addition, SUSD-TR and SUSD can be parallelized for problems with high time-cost evaluation, which is an advantage compared to NEWUOA (it can evaluate $m$ points at the same time at the line-search step). Figure 7 shows the ratio of the total time and the parallel time during one complete loop (containing the trust-region step and the line-search step) of

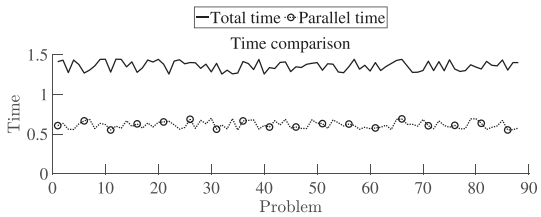



Figure 7 Parallel time ratio.

the algorithm[4] of the implementation of the algorithm SUSD-TR, and we find that the ratio is almost $\frac{1}{2}$ for solving the test problems in Table 1. To reduce the computation time of the non-parallel part, a possible approach is to solve the trust-region subproblem with a more approximate method.

Based on the numerical results above, it is evident that our algorithm's performance is improved by combining the SUSD direction with the trust-region technique, compared to only using the SUSD direction.

## 6 Conclusion and Future Work

We propose the algorithm SUSD-TR, which combines both the process of solving the trust-region subproblem with the approximation objective function and the points transporting along the SUSD direction. We present the dynamics and the convergence of the direction of the algorithm SUSD-TR. We implement a simple numerical version of the SUSD-TR. The numerical results show the advantage of our algorithm SUSD-TR. More efficient stepsizes, better combination ways and different SUSD directions will be considered. Furthermore, the small-scale modification on multiple points can also be studied on. The corresponding method for constrained problems is also a future work.

**Acknowledgement** The authors would like to thank the support of the Institute of Computational Mathematics and Scientific/Engineering Computing of the Chinese Academy of Sciences.

## Declarations

**Conflicsts of interest** The authors declare no conflicts of interest.

## References

[1] Al-Abri, S., Lin, T. X., Tao, M. and Zhang, F., A derivative-free optimization method with application to functions with exploding and vanishing gradients, *IEEE Contr. Syst. Lett.*, **5**(2), 2020, 587–592.

[2] Audet, C. and Hare, W., Derivative-free and Blackbox Optimization, Springer-Verlag, Heidelberg, 2017.

[3] Bezerra, M., Santelli, R., Oliveira, E., et al., Response surface methodology (RSM) as a tool for optimization in analytical chemistry, *Talanta*, **76**, 2008, 965–977.

[4]The non-parallel part is basically in the trust-region step.

[4] Conn, A. R., Scheinberg, K. and Vicente, L. N., Geometry of sample sets in derivative free optimization, part II: Polynomial regression and underdetermined interpolation, *IMA J. Numer. Anal.*, **28**, 2008, 721–748.

[5] Conn, A. R., Scheinberg, K. and Vicente, L. N., Introduction to Derivative-free Optimization, SIAM, Philadelphia, 2009.

[6] Gill, P. E. and Murray, W., Quasi-Newton methods for unconstrained optimization, *IMA J. of Appl. Math.*, **9**(1), 1972, 91–108.

[7] Gill, P. E., Murray, W., Saunders, M. A. and Wright, M. H., Computing forward-difference intervals for numerical optimization, *SIAM J. Sci. Statist. Comput.*, **4**(2), 1983, 310–321.

[8] Kelley, C. T., Implicit Filtering, SIAM, Philadelphia, 2011.

[9] Khalil, H. K., Nonlinear Systems, Prentice Hall, Upper Saddle River, 2002.

[10] Kolda, T., Lewis, R. and Torczon, V., Optimization by direct search: New perspectives on some classical and modern methods, *SIAM Rev.*, **45**, 2003, 385–482.

[11] Larson, J., Menickelly, M. and Wild, S. M., Derivative-free optimization methods, *Acta Numer.*, **28**, 2019, 287–404.

[12] Moré J. J. and Wild, S. M., Benchmarking derivative-free optimization algorithms, *SIAM J. Optim.*, **20**(1), 2009, 172–191.

[13] Nelder, J. A. and Mead, R., A simplex method for function minimization, *Comput. J.*, **7**(4), 1965, 308–313.

[14] Nocedal, J. and Yuan, Y., Combining trust region and line search techniques, Advances in Nonlinear Programming, Springer-Verlag, Boston, 1998, 153–175.

[15] Powell, M. J. D., UOBYQA: Unconstrained optimization by quadratic approximation, *Math. Program.*, **92**, 2002, 555–582.

[16] Powell, M. J. D., On trust region methods for unconstrained minimization without derivatives, *Math. Program.*, **97**, 2003, 605–623.

[17] Powell, M. J. D., Least Frobenius norm updating of quadratic models that satisfy interpolation conditions, *Math. Program.*, **100**, 2004, 183–215.

[18] Powell, M. J. D., The NEWUOA software for unconstrained optimization without derivatives, Large-scale nonlinear optimization, Nonconvex Optim. Appl., **83**, Springer-Verlag, New York, 2006, 255–297.

[19] Rosenbrock, H., An automatic method for finding the greatest or least value of a function, *Comput. J.*, **3**(3), 1960, 175–184.

[20] Stewart III, G., A modification of Davidon's minimization method to accept difference approximations of derivatives, *J. ACM*, **14**(1), 1967, 72–83.

[21] Torczon, V., On the convergence of the multidirectional search algorithm, *SIAM J. Optim.*, **1**(1), 1991, 123–145.

[22] Tseng, P., Fortified-descent simplicial search method: A general approach, *SIAM J. Optim.*, **10**, 1999, 269–288.

[23] Van Laarhoven, P. J. M. and Aarts, E. H. L., Simulated Annealing: Theory and Applications, Springer-Verlag, Netherlands, Dordrecht, 1987.

[24] Winfield, D., Function and Functional Optimization by Interpolation in Data Tables, Ph.D. thesis, Harvard University, 1969.

[25] Xie, P. and Yuan, Y., Least $H^2$ norm updating quadratic interpolation model function for derivative-free trust-region algorithms, 2023., arXiv: 2302.12017

[26] Zhang, Z., Derivative-free Optimization, China Discipline Development Strategy: Mathematical Optimization (in Chinese), Science Press, Beijing, 2021, 84–92.