Continuation Newton methods with deflation techniques and quasi-genetic evolution for global optimization problems

Xin-long Luo* · Hang Xiao · Sen Zhang

Received: date / Accepted: date

Abstract The global minimum point of an optimization problem is of interest in engineering fields and it is difficult to be solved, especially for a nonconvex large-scale optimization problem. In this article, we consider a new memetic algorithm for this problem. That is to say, we use the continuation Newton method with the deflation technique to find multiple stationary points of the objective function and use those found stationary points as the initial seeds of the evolutionary algorithm, other than the random initial seeds of the known evolutionary algorithms. Meanwhile, in order to retain the usability of the derivative-free method and the fast convergence of the gradient-based method, we use the automatic differentiation technique to compute the gradient and replace the Hessian matrix with its finite difference approximation. According to our numerical experiments, this new algorithm works well for unconstrained optimization problems and finds their global minima efficiently, in comparison to the other representative global optimization methods such as the multi-start methods (the built-in subroutine GlobalSearch.m of MATLAB R2021b, GLODS and VRBBO), the branch-and-bound method (Couenne, a state-of-the-art open-source solver for mixed integer nonlinear programming problems), and the derivative-free algorithms (CMA-ES and MCS).

Keywords continuation Newton method · deflation technique · derivative-free method · automatic differentiation · global optimization · genetic evolution

Mathematics Subject Classification (2010) 65K05 · 65L05 · 65L20
1 Introduction

In this article, we are mainly concerned with the global minimum of the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function except for the finite points. When problem (1) is nonconvex and large-scale, it may have many local minimizers and it is a challenge to find its global minimum. For problem (1), there are many popular global optimization methods, such as the multi-start methods [11, 20, 26, 43, 73, 85], the branch-and-bound methods [10, 17, 76, 83], the genetic evolution algorithms [32, 46, 59, 63] and their memetic algorithms [35, 60, 78, 84].

For multi-start methods, they use the local optimization methods such as the trust-region methods [15, 25, 90] or the line search methods [70, 81] to solve problem (1) and find its local minimum point $x^*$, which satisfies the following first-order optimality condition

$$g(x^*) = \nabla f(x^*) = 0.$$

Since the implementations of multi-start methods are simple and successful for many real-world problems such as the design of hybrid electric vehicles [28], molecular conformal optimization problems [4, 8, 45] and acoustic problems [66], many commercial global solvers are based on multi-starting points such as GlobalSearch.m of MATLAB R2021b [61, 85]. However, for the multi-start method, a random starting point could easily lie in the basin of attraction of a previously found local minimum point [43, 73, 85]. Hence, the algorithm might find the same local minimum even if the searches are initiated from points those are far apart. Other multi-start methods based on the deterministic procedure may have too many search branches (the number of search regions is $2^{n^*}$ at the $l$-th level iteration, where $n$ is the dimension of the problem) [20, 37, 68]. Hence, the computational complexity is very high when the dimension of the problem is large for these deterministic multi-start methods.

The evolutionary algorithms [32, 65, 78, 79] are another class of popular methods due to their simplicity and attractive nature-inspired interpretations, and are used by a broad community of engineers and practitioners to solve real-world problems such as finding optimal locations for active control actuators [34], the optimization of acoustic absorber configuration [88], and musical instruments [14]. However, for the evolutionary algorithms, since there are not a guaranteed convergence and their computational time are very heavy for the large-scale problems, they are difficult to find the global minimum accurately.

In order to reduce the computational complexity of the multi-start methods and retain the advantage of the evolutionary algorithms escaping from the local minima, we use the continuation Newton methods [5, 53, 56] with the revised deflation technique [13] to find the multiple stationary points of the objective function as the initial seeds of the quasi-genetic algorithm, which are other than the random seeds of the
known evolutionary algorithms [59,63,65]. The practical multi-start methods and the evolutionary algorithms are the derivative-free methods. Namely, the gradient of the objective function does not need to be provided by the user and the derivative-free methods [16] are regarded as the black-box methods [80].

In order to retain the usability of derivative-free methods and the fast convergence of the gradient-based method, we use the automatic differentiation techniques [9, 31, 67, 86, 87] with the reverse mode to compute the gradient of the objective function and replace the Hessian matrix with its finite difference approximation. In general, the time that it takes to calculate the gradient of the objective function by automatic differentiation with the reverse mode is about a constant multiple of the function cost [9, 67, 87]. A rigorous analysis of all time costs associated with the reverse method in general shows that the cost is always less than four times the cost of the function evaluations (pp. 83-86, [31]). Therefore, the computational complexity of the gradient by automatic differentiation with the reverse mode is far less than that of the gradient by the finite difference approximation or that of the analytical gradient, which is about \( n \) times the cost of the function evaluations.

Recently, Luo et al intensively investigate the continuation methods for linearly constrained optimization problems [52, 57], the system of nonlinear equations [53, 56], unconstrained optimization problems [55], the linear programming problem [54] and the linear complementarity problem [58]. Those continuation methods are only suitable to solve a solution of nonlinear equations. In this paper, we need to confront multi-solutions of nonlinear equations. Namely, in order to obtain the global minimum of the objective function, we need to find its stationary points as many as possible. Therefore, we revise the deflation technique [13] and combine it with the continuation method [53, 56] to find multiple stationary points of the objective function.

The rest of this article is organized as follows. In the next section, we convert the nonlinear system (2) into the continuation Newton flow. Then, we use a continuation Newton method [53, 56] with the trust-region updating strategy [15, 81] to follow the continuation Newton flow and find a stationary point of the objective function. In Section 3, we revise the deflation technique [13] to construct a new nonlinear function \( G_k(\cdot) \) such that the zeros of \( G_k(x) \) are the stationary points of the objective function and exclude the known stationary points \( x^*_1, \ldots, x^*_k \) of the objective function. In Section 4, we give a quasi-genetic algorithm to find a suboptimal point \( x^{ag}_i \) by using the smallest \( L \) points \( x^*_1, \ldots, x^*_L \) of all found stationary points \( x^*_1, \ldots, x^{ag}_i \) as the evolutionary seeds. After we obtain the suboptimal point \( x^{ag}_i \) of the quasi-genetic algorithm, we refine it by using it as the initial point of the continuation Newton method [53, 56] to solve the system (2) of nonlinear equations and obtain a better approximation of the global minimum point. In Section 5, some promising numerical results of the proposed algorithm are reported, with comparison to other representative global optimization methods such as the multi-start methods (the subroutine GlobalSearch.m of MATLAB R2021b [61], GLODS: Global and Local Optimization using Direct Search [20], and VRBBO: Vienna Randomized Black Box Optimization [43]), the branch-and-bound method (a state-of-the-art open-source solver (Couenne) [10, 17]),
and the derivative-free algorithms (the covariance matrix adaptation evolution strategy (CMA-ES) [32, 33] and the multilevel coordinate search (MCS) [37, 68]). Finally, some discussions and conclusions are also given in Section 6. \(\| \cdot \|\) denotes the Euclidean vector norm or its induced matrix norm throughout the paper.

2 Finding a stationary point with the continuation Newton method

For convenience, we restate the continuation Newton method with the trust-region updating strategy [53, 56], which is used to find a stationary point of \(f(x)\), i.e., a root of nonlinear equations (2). We consider the damped Newton method [41, 42, 71, 81] for solving a root of nonlinear equations (2) as follows:

\[
x_{k+1} = x_k - \alpha_k H(x_k)^{-1} g(x_k),
\]

where \(g(x)\) is the gradient of \(f(x)\) and \(H(x)\) is its Hessian matrix.

If we regard \(x_{k+1} = x(t_k + \alpha_k),\ x_k = x(t_k)\) and let \(\alpha_k \to 0\), we obtain the following continuous Newton flow [7, 12, 21, 53, 82]:

\[
\frac{dx(t)}{dt} = -H(x)^{-1} g(x), \quad x(t_0) = x_0.
\]

Actually, if we apply an iteration with the explicit Euler method [6, 75] for the continuous Newton flow (4), we obtain the damped Newton method (3). Since the Hessian matrix \(H(x)\) may be singular, we reformulate the continuous Newton flow (4) as the following general formula:

\[
H(x) \frac{dx(t)}{dt} = -g(x), \quad x(t_0) = x_0.
\]

The continuous Newton flow (5) is an old method and can be backtracked to Davidenko’s work [21] in 1953. After that, it was investigated by Branin [12], Deuflhard et al [24], Tanabe [82] and Abbott [1] in 1970s, and applied to nonlinear boundary problems by Axelsson and Sysala [7] recently. The continuous and even growing interest in this method partially due to its global convergence as stated by the following Proposition 2.1.

**Proposition 2.1** (Branin [12] and Tanabe [82]) Assume that \(x(t)\) is a solution of the continuous Newton flow (5). Then, the energy function \(E(x(t)) = \|g(x(t))\|^2\) converges to zero when \(t \to \infty\). That is to say, for every limit point \(x^*\) of \(x(t)\), it is also a solution of nonlinear equations (2). Furthermore, every element \(g_i(x)\) \((i = 1, 2, \ldots, n)\) of \(g(x)\) has the same linear convergence rate and \(x(t)\) can not converge to the solution \(x^*\) of nonlinear equations (2) on the finite interval when the initial point \(x_0\) is not a solution of nonlinear equations (2).

**Proof.** For the proof of the first part i.e. the convergence \(\lim_{t \to \infty} g(x) = 0\), one can refer to references [12, 82]. The proof of the second part can be found in references [53, 56]. \(\square\)
Since the stiff concept of an ordinary differential equation (ODE) is a very key role to understand the numerical method integrating an ODE trajectory, we give its definition in Definition 2.1 for convenience.

**Definition 2.1** (The definition of the stiff ODE [38, 44]) An ordinary differential equation \( \frac{dx}{dt} = F(x) \) is called the stiff ODE, if \( \text{Re}(\lambda_i(x)) \leq 0 \) \((i = 1, 2, \ldots, n)\) and the ratio \( r(x) = \max_{i=1,\ldots,n} \left| \text{Re}(\lambda_i(x)) \right| / \min_{i=1,\ldots,n} \left| \text{Re}(\lambda_i(x)) \right| \) is very large, where \( \text{Re}(\lambda_i(x)) \) represents the real part of \( \lambda_i(x) \) and \( \lambda_i(x) \) \((i = 1, \ldots, n)\) are the eigenvalues of the Jacobian \( F'(x) \).

**Remark 2.1** The inverse \( H(x)^{-1} \) of the Hessian matrix \( H(x) \) can be regarded as the pre-conditioner of \( g(x) \) such that every element \( x_i(t) \) \((i = 1, 2, \ldots, n)\) of the solution \( x(t) \) of the continuous Newton flow (4) has the roughly same convergence rate, and it mitigates the stiffness of the ODE (4). This property is very useful since it makes us adopt the explicit ODE method to follow the trajectory of the Newton flow.

Actually, if we consider \( g(x) = Ax \), where \( A \) is an \( n \times n \) nonsingular and symmetric matrix, from the ODE (5), we have

\[
A \frac{dx(t)}{dt} = -Ax, \ x(0) = x_0,
\]

which gives

\[
\frac{dx(t)}{dt} = -x, \ x(0) = x_0.
\] (6)

From Definition 2.1, we know that the ODE (6) is a non-stiff ODE. By integrating the linear ODE (6), we obtain

\[
x(t) = e^{-t}x_0.
\] (7)

From equation (7), we know that every element \( x_i(t) \) \((i = 1, 2, \ldots, n)\) of the solution \( x(t) \) converges to zero linearly with the same rate when \( t \) tends to infinity.

When the Hessian matrix \( H(x) \) is singular or nearly singular, the ODE (5) is the system of differential-algebraic equations (DAEs), and its trajectory can not be efficiently followed by the general ODE methods [6, 38, 39] such as the backward differentiation formulas (the built-in subroutine ode15s.m of MATLAB R2021b [61, 75]). Therefore, we need to construct the special method to handle this problem.

We expect that the new method has the global convergence as the homotopy continuation methods, and the fast convergence rate near the stationary point \( x^* \) as the merit-function-based methods. In order to achieve these two aims, we construct the special continuation Newton method with the new step \( \alpha_k = \Delta t_k / (1 + \Delta t_k) \) [53–58] and the time step \( \Delta t_k \) is adaptively adjusted by the trust-region updating strategy [15, 89, 90] to follow the trajectory of the ODE (5).
 Firstly, we apply the implicit Euler method [75] to the continuous Newton flow (5). Then, we obtain

$$H(x_{k+1})(x_{k+1} - x_k) = -\Delta t_k g(x_{k+1}). \tag{8}$$

The scheme (8) is an implicit formula and it needs to solve a system of nonlinear equations at every iteration. To avoid solving the system of nonlinear equations, we replace the Hessian matrix $H(x_{k+1})$ with $H(x_k)$, and substitute $g(x_{k+1})$ with its linear approximation $g(x_k) + H(x_k)(x_{k+1} - x_k)$ in equation (8), respectively. Thus, we obtain the following explicit continuation Newton method:

$$H_k s^N_k = -g(x_k), \quad s_k = \frac{\Delta t_k}{1 + \Delta t_k} x^N_k, \quad x_{k+1} = x_k + s_k, \tag{9}$$

where $H_k$ equals $H(x_k)$ or its approximation.

The continuation Newton method (9) is similar to the damped Newton method (3) if we set $\alpha_k = \Delta t_k/(1 + \Delta t_k)$ in equation (9). However, from the view of the ODE method, they are different. The damped Newton method (3) is obtained by the explicit Euler method applied to the continuous Newton flow (5), and its time step $\alpha_k$ is restricted by the numerical stability [38]. That is to say, for the linear test function $g(x) = Ax$, from equation (6), we know that its time step $\alpha_k$ is restricted by the stable region $|1 - \alpha_k| \leq 1$. Therefore, the large time step $\alpha_k$ can not be adopted in the steady-state phase.

The continuation Newton method (9) is obtained by the implicit Euler method and its linear approximation applied to the continuous Newton flow (5), and its time step $\Delta t_k$ is not restricted by the numerical stability for the linear test equation (6). Therefore, the large time step can be adopted in the steady-state phase for the continuation Newton method (9), and it mimics the Newton method near the stationary point $x^*$ such that it has the fast local convergence rate.

The most of all, $\alpha_k = \Delta t_k/(\Delta t_k + 1)$ in equation (9) is favourable to adopt the trust-region updating strategy for adaptively adjusting the time step $\Delta t_k$ such that the continuation Newton method (9) accurately follows the trajectory of the continuous Newton flow in the transient-state phase and achieves the fast convergence rate near the stationary point $x^*$.

The main idea of the adaptive control step $\Delta t_k$ based on the trust-region updating strategies [22, 36, 47–51] can be described as follows. The time step $\Delta t_{k+1}$ will be enlarged when the linear model $g(x_k) + H_k s_k$ approximates $g(x_k + s_k)$ well, and $\Delta t_{k+1}$ will be reduced when $g(x_k) + H_k s_k$ approximates $g(x_k + s_k)$ badly. Thus, by using the relation (9), we enlarge or reduce the time step $\Delta t_{k+1}$ at every iteration according to the following ratio:

$$\rho_k = \frac{\|g(x_k)\| - \|g(x_k + s_k)\|}{\|g(x_k)\| - \|g(x_k) + H_k s_k\|} = 1 + \Delta t_k \frac{\|g(x_k)\| - \|g(x_k + s_k)\|}{\Delta t_k \|g(x_k)\|}. \tag{10}$$

When the gradient $g(x)$ is very nonlinear, a decrease in the gradient is rarely found and $\Delta t_k$ becomes too small and even zero before a minimizer is found. In such case, $\rho_k$
defined by equation (10) goes to -∞. To overcome this problem, \( \Delta t_k \) must be restricted from below by a tuning small positive parameter \( \Delta t < 1 \). Thus, by combining it with equation (10), we give a following particular adjustment strategy:

\[
\Delta t_{k+1} = \begin{cases} 
  c_2 \Delta t_k, & \text{if } 0 \leq |1 - \rho_k| \leq \eta_1, \\
  c_1 \Delta t_k, & \text{if } |1 - \rho_k| \geq \eta_2 \text{ and } \Delta t_k \geq \Delta t, \\
  \Delta t_k, & \text{others},
\end{cases}
\]

(11)

where the constants \( c_1, c_2, \eta_1, \eta_2 \) satisfy the conditions \( 0 < c_1 < c_2 \leq 1 \), \( 0 < \eta_1 < \eta_2 < 1 \), and \( 0 < \Delta t < 1 \) is a small threshold value. When \( \rho_k \geq \eta_a \), we accept the trial step and set

\[
x_{k+1} = x_k + s_k,
\]

(12)

where \( s_k \) is solved by equation (9) and \( \eta_a \) is a small positive number. Otherwise, we discard the trial step and set

\[
x_{k+1} = x_k.
\]

(13)

**Remark 2.2** This time-stepping selection based on the trust-region updating strategy has some advantages compared to the traditional line search strategy [48]. If we use the line search strategy and the damped Newton method (3) to track the trajectory \( x(t) \) of the continuous Newton flow (5), in order to achieve the fast convergence rate in the steady-state phase, the time step \( \alpha_k \) of the damped Newton method is tried from 1 and reduced by the half with many times at every iteration. Since the linear model \( g(x_k) + H_k s_k \) may not approximate \( g(x_k + s_k) \) well in the transient-state phase, the time step \( \alpha_k \) will be small. Consequently, the line search strategy consumes the unnecessary trial steps in the transient-state phase. However, the time-stepping selection method based on the trust-region updating strategy (10)-(11) can overcome this shortcoming.

For a system of nonlinear equations, the Hessian evaluation is heavy if we update the Hessian matrix \( H(x_k) \) at every iteration. In order to reduce the computational time of the Hessian evaluation, similarly to the adaptively updating Jacobian technique [55, 56], we set \( H_{k+1} = H_k \) when \( g(x_k) + H_k s_k \) approximates \( g(x_k + s_k) \) well. Otherwise, we update \( H_{k+1} = H(x_{k+1}) \). An effective updating strategy is give by

\[
H_{k+1} = \begin{cases} 
  H_k, & \text{if } |1 - \rho_k| \leq \eta_1, \\
  H(x_{k+1}), & \text{otherwise},
\end{cases}
\]

(14)

where \( \rho_k \) is defined by equation (10). Thus, when \( H_k \) performs well, i.e. \( |1 - \rho_k| \leq \eta_1 \), according to the updating formula (14), we set \( H_{k+1} = H_k \) in equation (9).

According to the above discussions, we give the detailed descriptions of the continuation Newton method and the trust-region updating strategy for finding a stationary point of \( f(x) \) (i.e. a root of nonlinear equations (2)) in Algorithm 1. For the
Algorithm 1 Continuation Newton methods with the trust-region updating strategy for finding a stationary point of the objective function (CNMTr)

Input:
the gradient \( g(x) = \nabla f(x) \) of the objective function \( f : \mathbb{R}^n \to \mathbb{R} \), an initial point \( x_0 \) (optional), the tolerance \( \epsilon \) (optional), the Hessian matrix \( H(x) \) of \( f(x) \) (optional).

Output:
The found stationary point \( x^* \) of the objective function, the Boolean variable \( \text{success} \).

1. Set the default \( x_0 = 1 \) (\( i = 1, 2, \ldots, n \)) and \( \epsilon = 10^{-6} \) when \( x_0 \) or \( \epsilon \) is not provided.
2. Initialize the parameters: \( \eta_a = 10^{-6}, \eta_1 = 0.25, \eta_2 = 0.75, c_1 = 0.5, c_2 = 2, \Delta t = 10^{-7}, \Delta t_{\text{init}} = 10^{-2}, \maxit = 200. \)
3. Set \( \Delta t_0 = \Delta t_{\text{init}}, \text{itc} = 0, k = 0. \)
4. Let the Boolean variable \( \text{success} \) be \( \text{false} \).
5. Let the Boolean variable \( \text{success}_{\text{ap}} \) be \( \text{true} \).
6. Evaluate \( g_0 = \nabla f(x_0) \).
7. Compute the residual \( \text{Res}_0 = \| g_0 \|_\infty \).
8. Set \( \rho_{-1} = 0, s_{-1} = 0. \)
9. while \( \text{Res}_k > \epsilon \) do
10: if \( \text{success}_{\text{ap}} \) is \( \text{true} \) then
11: Set \( \text{itc} = \text{itc} + 1. \)
12: if \( \text{itc} > \maxit \) then
13: break;
14: end if
15: if \( |1 - \rho_{-1}| > \eta_1 \) then
16: Evaluate \( H_k = H(x_k) \).
17: else
18: Set \( H_k = H_k - 1 \).
19: end if
20: Solve \( H_k s_k^N = -g_k \) to obtain the Newton step \( s_k^N \).
21: end if
22: Set \( x_k = \Delta t_k / (1 + \Delta t_k) s_k^N, x_{k+1} = x_k + s_k. \)
23: Evaluate \( g_{k+1} = \nabla f(x_{k+1}) \).
24: Compute the ratio \( \rho_k \) from equation (10).
25: Adjust the time step \( \Delta t_{k+1} \) according to the trust-region updating strategy (11).
26: if \( \rho_k \geq \eta_2 \) then
27: Accept the trial point \( x_{k+1} \). Compute the residual \( \text{Res}_{k+1} = \| g_{k+1} \|_\infty \).
28: Let the Boolean variable \( \text{success}_{\text{ap}} \) be \( \text{true} \).
29: else
30: Set \( x_{k+1} = x_k, g_{k+1} = g_k, s_{k+1}^N = s_k^N \).
31: Let the Boolean variable \( \text{success}_{\text{ap}} \) be \( \text{false} \).
32: end if
33: Set \( k \leftarrow k + 1. \)
34: end while
35: if \( \text{Res}_k \leq \epsilon \) then
36: Let the Boolean variable \( \text{success} \) be \( \text{true} \).
37: else
38: Let the Boolean variable \( \text{success} \) be \( \text{false} \).
39: end if
40: Set \( x^* = x_k. \)
41: Output the found stationary point \( x^* \) and the Boolean variable \( \text{success} \).
In line 5, the Boolean variable `success` is used to indicate that the continuation Newton method (Algorithm 1) finds a stationary point of \( f(x) \) successfully or not. When Algorithm 1 finds a stationary point of \( f(x) \) successfully, we let the Boolean variable `success` be `true`. Otherwise, we let the Boolean variable `success` be `false`.

In line 6, it evaluates the gradient at the initial point \( x_0 \). In line 7, it computes the infinite norm of the initial gradient. In lines 10-21, they give the procedures to solve the Newton step \( s_k \). In line 9, we use the infinite norm \( \text{Res}_k = \| g_k \|_\infty \) to determine if the algorithm has converged, where it can provide more accurate assessment to the status of each element \( |g_k^i| < \varepsilon \) (\( i = 1, 2, \ldots, n \)) than the 2-norm \( \| g_k \|_2 < \varepsilon \).

In line 22, it computes the trial step \( s_k \) and the trial point \( x_{k+1} \). In line 23, it evaluates the new gradient \( g(x_{k+1}) \). In lines 24, it computes the ratio \( \rho_k \) according to equation (10). In line 25, it adaptively adjusts the time step \( \Delta t_{k+1} \) according to the trust-region updating strategy (11). In lines 26-32, they give the criterion whether the trial step \( s_k \) is accepted or not according to the ratio \( \rho_k \geq \eta_k \) or not. In lines 35-39, they give a criterion how to assign the Boolean variable `success`. When the final residual \( \text{Res}_k = \| g_k \|_\infty \) is less than the given tolerance, we regard that Algorithm 1 finds a stationary point of the objective function successfully and let the Boolean variable `success` be `true`. Otherwise, we let the Boolean variable `success` be `false`.

In lines 40-41, they output the found stationary point \( x^* = x_k \) and the Boolean variable `success`.

In order to understand Algorithm 1 better, we give some explanations of its operating mechanism and some key variables. In line 2, the parameters \( c_1, c_2, \eta_1, \eta_2, \Delta t \) are used to adaptively adjust the time step \( \Delta t_k \) and they are defined by equation (11). The ratio \( \rho_k \) defined by (10) represents the ratio of the actual reduction \( \| g(x_k) \| - \| g(x_k + s_k) \| \) to the predicted reduction \( \| g(x_k) \| - \| g(x_k + H_k s_k) \| \). The parameters \( \eta_1, \eta_2 (0 < \eta_1 < 1 \leq \eta_2) \) are used to define the range of the ratio \( \rho_k \) and indicate that the linear model \( g(x_k) + H_k s_k \) approximates \( g(x_k + s_k) \) well or not. If \( |1 - \rho_k| < \eta_1 \), it indicates that \( g(x_k) + H_k s_k \) approximates \( g(x_k + s_k) \) well. Then, the time step \( \Delta t_{k+1} \) is enlarged and it equals \( c_2 \Delta t_k \). If \( |1 - \rho_k| > \eta_2 \), it indicates that \( g(x_k) + H_k s_k \) approximates \( g(x_k + s_k) \) neither well nor bad. Then, the time step \( \Delta t_{k+1} \) is reduced and it equals \( c_1 \Delta t_k \). If \( \eta_1 \leq |1 - \rho_k| \leq \eta_2 \), it indicates that \( g(x_k) + H_k s_k \) approximates \( g(x_k + s_k) \) well but not well. Then, the time step \( \Delta t_{k+1} \) is kept with \( \Delta t_k \). The parameter \( 0 > \Delta t < 1 \) is a small number and used to prevent the time step \( \Delta t_k \) from becoming too small and even zero, when the gradient \( g(x) \) is very nonlinear and \( \Delta t_k \) is adaptively adjusted by the scheme (11). \( 0 < \Delta t_{\text{init}} \leq 1 \) represents the initial time step and its default value is \( \Delta t_{\text{init}} = 10^{-2} \). The parameter `maxit` represents the maximum iterations.
3 The deflation technique

By using Algorithm 1 (CNMTr) in Section 2, we can find a stationary point \( x_1^* \) of \( f(x) \). Our strategy is that we repeatedly use Algorithm 1 to find the stationary points of \( f(x) \) as many as possible. Then, we use those found stationary points as the initial evolutionary seeds of the genetic algorithm in Section 4. If we directly use Algorithm 1 with the multi-start method, a starting point could easily lie in the basin of attraction of a previously found local minimum and algorithm 1 may find the same local minimum even if the searches are initiated from points that are far apart [20, 43, 73, 85]. In order to overcome this disadvantage, we revise the deflation technique [13] and combine it with Algorithm 1 to find stationary points of \( f(x) \) as many as possible.

Assume that \( x_k^* \) is a zero point of the function \( G_{k-1}(x) \). Then, we construct another function \( G_k(x) \) via eliminating the zero point \( x_k^* \) of \( G_{k-1}(x) \) as follows:

\[
G_k(x) = \frac{1}{\|x - x_k^*\|} G_{k-1}(x), \quad k = 0, 1, 2, \ldots, \tag{15}
\]

where \( G_0(x) = g(x) = \nabla f(x) \). Thus, from equation (15), we obtain

\[
G_k(x) = \frac{1}{\|x - x_1^*\| \cdots \|x - x_k^*\|} g(x), \tag{16}
\]

where \( x_i^* (i = 1, \ldots, k) \) are the zero points of \( g(x) \). From equation (16), we know that the zero point \( x_{k+1}^* \) of \( G_k(x) \) is also the zero point of \( g(x) \). Namely, the zero set of \( G_k(x) \) is a zero subset of \( g(x) \).

When \( g(x_i^*) = 0 \) and the Hessian matrix \( H(x_i^*) \) is nonsingular, Brown and Gearhard [13] have proved

\[
\lim_{j \to \infty} \inf \|G_k(x_j)\| > 0
\]

for any sequence \( x_j \to x_i^* \), where \( G_k(x) \) is defined by equation (16). Actually, we can obtain the following estimation of the lower bound of \( G_k(x) \) when \( x \) belongs to the neighborhood of \( x_i^* \).

**Lemma 3.1** Assume that the gradient \( g(x) \) of the objective function is continuously differential and \( x_i^* \) is a zero point of the gradient \( g(\cdot) \). Furthermore, the Hessian matrix \( H(x_i^*) \) satisfies

\[
\|H(x_i^*)y\| \geq c_l \|y\|, \quad \forall \ y \in \mathbb{R}^n, \tag{17}
\]

where \( c_l \) is a positive constant. Then, there exists a neighborhood \( B_{\delta_l}(x_i^*) = \{x : \|x - x_i^*\| \leq \delta_l\} \) of \( x_i^* \) such that

\[
\|G_k(x)\| \geq \frac{c_l}{2 \|x_1^*\| \cdots \|x_{i-1}^*\| \|x_{i+1}^* - x_i^*\| \cdots \|x_k^* - x_i^*\|} \tag{18}
\]

holds for all \( x \in B_{\delta_l}(x_i^*) (i = 1, 2, \ldots, k) \), where \( G_k(x) \) is defined by equation (16). That is to say, \( x_i^* (i = 1, 2, \ldots, k) \) is not a zero point of \( G_k(x) \).
Proof. Since $x_i^\ast$ is a zero point of $g(x)$, according to the first-order Taylor expansion, we have

$$g(x) = g(x_i^\ast) + \int_0^1 H(x_i^\ast + t(x - x_i^\ast))(x - x_i^\ast)\,dt$$

$$= \int_0^1 H(x_i^\ast + t(x - x_i^\ast))(x - x_i^\ast)\,dt.$$  \hspace{1cm} (19)

According to the nonsingular assumption (17) of $H(x_i^\ast)$ and the continuity of $H(\cdot)$, there exists a neighborhood $B_\delta(x_i^\ast)$ and the continuity of $H(\cdot)$, such that

$$\|H(x) - H(x_i^\ast)\| \leq \frac{c_f}{2}$$  \hspace{1cm} (20)

holds for all $x \in B_\delta(x_i^\ast)$. From equations (19)-(20), we have

$$\|g(x) - H(x_i^\ast)(x - x_i^\ast)\| = \left| \int_0^1 (H(x_i^\ast + t(x - x_i^\ast)) - H(x_i^\ast))(x - x_i^\ast)\,dt \right|$$

$$\leq \int_0^1 \|H(x_i^\ast + t(x - x_i^\ast)) - H(x_i^\ast)\| \|x - x_i^\ast\|\,dt \leq \frac{c_f}{2}\|x - x_i^\ast\|, \ \forall x \in B_\delta(x_i^\ast).$$ \hspace{1cm} (21)

Furthermore, from the triangle inequality $\|x - y\| \geq \|x\| - \|y\|$ and the assumption (17), we have

$$\|g(x) - H(x_i^\ast)(x - x_i^\ast)\| \geq \|H(x_i^\ast)(x - x_i^\ast)\| - \|g(x)\| \geq c_f\|x - x_i^\ast\| - \|g(x)\|.$$ \hspace{1cm} (22)

Thus, from equations (21)-(22), we obtain

$$\|g(x)\| \geq \frac{c_f}{2}\|x - x_i^\ast\|, \ \forall x \in B_\delta(x_i^\ast).$$ \hspace{1cm} (23)

By substituting inequality (23) into equation (16), we obtain

$$\|G_k(x)\| \geq \frac{c_f}{2}\|x - x_i^\ast\| \cdots \|x - x_{i-k}^\ast\| \prod_{l=1}^{i-k} \|x - x_l^\ast\| > 0, \ \forall x \in B_\delta(x_i^\ast).$$

That is to say, $x_i^\ast (i = 1, 2, \ldots, k)$ is not a zero point of $G_k(x)$. \hfill \square

In practice, we need to avoid the overflow or the underflow when we compute $G_k(x)$, and we expect that the continuation Newton method (Algorithm 1) finds stationary points of $f(x)$ as many as possible. According to our numerical experiments, Algorithm 1 can find more stationary points with $\|x - x_i^\ast\|_1$ as the deflation items than $\|x - x_i^\ast\|$ in equation (16). We also notice that $\|x - x_i^\ast\|_1 \approx \|x_i^\ast\|_1$ when $x$ is close to zero, where $n$ is the dimension of $x_i^\ast$. Thus, Algorithm 1 is apt to find the spurious root of $G_k$ when the dimension $n$ is large. Therefore, in order to improve the effect of the deflation technique, we modify $G_k(x)$ defined by equation (15) as

$$G_k(x) = \frac{\alpha_0}{\|x - x_k^\ast\|_1} G_{k-1}(x) = \left( \prod_{l=1}^{k} \frac{\alpha_l}{\|x - x_l^\ast\|_1} \right) g(x),$$  \hspace{1cm} (24)
where \( x_i^* (i = 1, 2, \ldots, k) \) are the known roots of \( g(x) \) and \( \alpha_i (i = 1, \ldots, k) \) are defined by
\[
\alpha_i = \begin{cases} n, & \text{if } \|x_i^*\|_1 \leq 10^{-6}, \\ \|x_i^*\|_1, & \text{otherwise}, \end{cases} \tag{25}
\]

Since \( G_k(x) \) has the special structure and its sub-differential exists except for finite points \( x_1^*, \ldots, x_k^* \), from equations (24)-(25), we compute its sub-differential \( \partial G_k(x) \) as follows:
\[
\partial G_k(x) = \alpha_1 \cdots \alpha_k \frac{\partial g(x)}{\partial x} + \alpha_1 \cdots \alpha_k \frac{g(x) p_k(x)^T}{\|x-x_1^*\|_1 \cdots \|x-x_k^*\|_1}, \tag{26}
\]
where \( p_k(x) \) is defined by
\[
p_k(x)^T = \frac{\text{sgn}(x-x_1^* \cdots \text{sgn}(x-x_k^*)}{\|x-x_1^*\|_1 + \cdots + \|x-x_k^*\|_1}. \tag{27}
\]
Here, the sign function \( \text{sgn}(\cdot) \) is defined by
\[
\text{sgn}(\beta) = \begin{cases} 1, & \text{if } \beta > 0, \\ 0, & \text{if } \beta = 0, \\ -1, & \text{otherwise}. \end{cases} \tag{28}
\]

Now, by using the revised deflation technique (24) of \( g(x) \) and the continuous Newton method (Algorithm 1), we can find a new stationary point of \( f(x) \) other than the known stationary points. We describe their detailed implementations in Algorithm 2.

In order to understand Algorithm 2 better, we give some explanations of its operating mechanism and its key variables. \( K \) is the number of the known stationary points of \( f(x) \). In lines 1-2, they give the default parameters used by the continuation Newton method to find a new stationary point of \( f(x) \) and they are explained in Algorithm 1. In line 3, the Boolean variable \( \text{success.dt} \) is used to indicate whether the continuation Newton method finds a new stationary point successfully or not. When the continuation Newton finds a new stationary point successfully, we let the Boolean variable \( \text{success.dt} \) be \text{true}. Otherwise, we let Boolean variable \( \text{success.dt} \) be \text{false}.

In lines 4-30, they give the pseudo codes of the continuation Newton method (CNMTr) to find a new stationary point of \( f(x) \). They are equivalent to Algorithm 1 if we replace \( g(x) \) and \( H(x) \) with \( G_k(x) \) and \( \partial G_k(x) \) in Algorithm 1, respectively. For their explanations, please refer to the related explanations in Algorithm 1. In lines 31-38, they output different values according to whether a new stationary point is found successfully or not. When \( \text{Res}_k \leq \varepsilon \), it indicates that the continuation Newton method finds a new stationary point successfully and we let the Boolean variable \( \text{success.dt} \)
Algorithm 2 Continuation Newton methods with the deflation technique for finding a new stationary point of the objective function (CNMDT)

**Input:**
- the gradient \( g(x) = \nabla f(x) \) of the objective function \( f : \mathbb{R}^n \to \mathbb{R} \), \( K \) known stationary points \( x_1^{*}, \ldots, x_K^{*} \) of \( f(x) \), an initial point \( x_0 \) (optional), the tolerance \( \epsilon \) (optional), the Hessian matrix \( H(x) \) of \( f(x) \) (optional).

**Output:**
- \( K + 1 \) found stationary points \( x_1^{*}, \ldots, x_{K+1}^{*} \) including the newly found stationary point \( x_{K+1}^{*} \) of \( f(x) \), the Boolean variable \( \text{success}_{\text{dt}} \).

1: Set the default \( x_0(i) = 1 \) (i = 1, 2, …, n) and \( \epsilon = 10^{-6} \) when \( x_0 \) or \( \epsilon \) is not provided.
2: Initialize the parameters: \( \eta_0 = 10^{-6}, \eta_1 = 0.25, \eta_2 = 0.75, \epsilon_1 = 0.5, \epsilon_2 = 2, \Delta t = 10^{-7}, \Delta t_{\text{init}} = 10^{-7}, \text{maxit} = 200 \).
3: Let the Boolean variable \( \text{success}_{\text{dt}} \) be false.
4: Set \( \Delta k = \Delta t_{\text{init}}, \text{iter} = 0, k = 0 \).
5: Evaluate \( f_0 = \nabla f(x_0) \). Then, compute \( F_0 = G_0(x_0) \) from equations (24)-(25).
6: Set \( p_{-1} = 0, s_{-1} = 0 \).
7: Let the Boolean variable \( \text{success}_{\text{ap}} \) be true.
8: Compute the residual \( \text{Res}_0 = ||F_0||_{\infty} \).
9: while \( \text{Res}_k > \epsilon \) do
10: if \( \text{success}_{\text{ap}} \) is true then
11: \( \text{iter} = \text{iter} + 1 \).
12: if \( \text{iter} > \text{maxit} \) then
13: break;
14: end if
15: Evaluate \( H_k = H(x_k) \). Then, compute the Jacobian \( J_k = \nabla G_k(x_k) \) from equations (26)-(28).
16: Solve \( J_k s_N^k = -F_k \) to obtain the Newton step \( s_N^k \).
17: end if
18: Set \( s_k = \Delta t_k / (1 + \Delta t_k) s_N^k, x_{k+1} = s_k + x_k \).
19: Evaluate \( F_{k+1} = \nabla f(x_{k+1}) \). Then, compute \( F_{k+1} = G_k(x_{k+1}) \) from equations (24)-(25).
20: Compute the ratio \( p_k \) from equation (10).
21: Adjust the time step \( \Delta t_{k+1} \) according to the trust-region updating strategy (11).
22: if \( p_k \geq \eta_1 \) then
23: Accept the trial point \( x_{k+1} \). Compute the residual \( \text{Res}_{k+1} = ||F_{k+1}||_{\infty} \).
24: Let the Boolean variable \( \text{success}_{\text{ap}} \) be true.
25: else
26: Set \( s_{k+1} = s_k, F_{k+1} = F_k, s_N^{k+1} = s_N^k \).
27: Let the Boolean variable \( \text{success}_{\text{ap}} \) be false.
28: end if
29: Set \( k = k + 1 \).
30: end while
31: if \( \text{Res}_k \leq \epsilon \) then
32: Let the Boolean variable \( \text{success}_{\text{dt}} \) be true.
33: Save the newly found stationary point as \( x_{K+1}^{*} = x_k \).
34: Output \( K + 1 \) found stationary points \( x_1^{*}, \ldots, x_{K+1}^{*} \) and the Boolean variable \( \text{success}_{\text{dt}} \).
35: else
36: Let the Boolean variable \( \text{success}_{\text{dt}} \) be false.
37: Output \( K \) known stationary points \( x_1^{*}, \ldots, x_K^{*} \) and the Boolean variable \( \text{success}_{\text{dt}} \).
38: end if

be true. Then, this newly found stationary point is saved as \( x_{K+1}^{*} \). They output \( K + 1 \) found stationary points \( x_1^{*}, \ldots, x_{K+1}^{*} \) and the Boolean variable \( \text{success}_{\text{dt}} \). Otherwise, we let the Boolean variable \( \text{success}_{\text{dt}} \) be false. Then, they output \( K \) known stationary points \( x_1^{*}, \ldots, x_K^{*} \) and the Boolean variable \( \text{success}_{\text{dt}} \).
Remark 3.1  From equation (24), we know that $\nabla f(x_i)$ may converge to a positive number when $\lim_{i \to \infty} \|x_i - x^*_i\|_1 = \infty$, where $x^*_i$ is the stationary point of $f(x)$. One of the reasons is that the Newton method fails to converge to the certain stationary point of $f(x)$ when the initial point $x_0$ belongs to a special region [1]. Thus, the continuation Newton method with the deflation technique may not find all stationary points of $f(x)$ from the same initial point $x_0$.

In order to find stationary points of $f(x)$ as many as possible, we use the continuation Newton method with the deflation technique (i.e. Algorithm 2) to find the stationary points from several far apart points sequentially. In practice, we define the $n/2 \times 1$ vector $e = (1, \ldots, 1)^T$ and select the following six points $x_j^{\text{init}}$ ($i = 1, \ldots, 6$) as the default initial points:

$$
x_1^{\text{init}} = \begin{pmatrix} e \\ -e \end{pmatrix}, x_2^{\text{init}} = -\begin{pmatrix} e \\ -e \end{pmatrix}, x_3^{\text{init}} = \begin{pmatrix} e \\ -e \end{pmatrix}, x_4^{\text{init}} = \begin{pmatrix} e \\ -e \end{pmatrix},$$

$$x_5^{\text{init}}(i) = i, i = 1, 2, \ldots, 6, x_6^{\text{init}}(i) = (n + 1) - i, i = 1, 2, \ldots, n,$$

(29)

where $x_j^{\text{init}}(i)$ represents the $i$-th element of the $n$-dimensional vector $x_j^{\text{init}}$. Thus, we repeatedly use Algorithm 2 from multi-start points to find stationary points of $f(x)$ as many as possible. Their detailed implementations are described in Algorithm 3.

In order to understand Algorithm 3 better, we give some explanations of its operating mechanism and its key variables. In line 1, $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ are $L$ default initial points defined by equation (29), where $L = 6$. In line 3, the Boolean variable $\text{success}$ represents whether a stationary point of $f(x)$ is found by CNMTr (Algorithm 1) successfully or not. When Algorithm 1 (CNMTr) finds a stationary point successfully, we let the Boolean variable $\text{success}$ be $\text{true}$. Otherwise, we let the Boolean variable $\text{success}$ be $\text{false}$. In line 4, $\text{np}$ represents the number of found stationary points and its default value equals 0.

In lines 5-15, they repeatedly call Algorithm 1 (CNMTr) to find a stationary point of $f(x)$ from $L$ default initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$. Once Algorithm 1 finds a stationary point, we let the Boolean variable $\text{success}$ be $\text{true}$, save this found stationary point as $x_1^*$, set $\text{np} = 1$, and exit the loop. Otherwise, we let the Boolean variable $\text{success}$ be $\text{false}$.

In lines 16-30, if Algorithm 1 finds a stationary point of $f(x)$ successfully, they will repeatedly call Algorithm 2 (CNMDT) to find the new stationary point of $f(x)$ from $L$ default initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ as many as possible. In line 17, the variable $i$ represents the order of given initial points. In line 18, it means that we execute lines 19-28 if the order variable $i$ is less than the number of given initial points (i.e. $L$). In line 19, it selects the $i$-th initial point $x_i^{\text{init}}$ and calls Algorithm 2 (CNMDT) to find a new stationary point of $f(x)$. In lines 20-28, they execute the different procedures according to whether a new stationary point is found successfully by Algorithm 2 (CNMDT) or not. If a new stationary point of $f(x)$ is found Algorithm 2 (CNMDT) successfully, we save this newly found stationary point and let $\text{np}$ (the number of found stationary points) increase by 1. Otherwise, we let the order variable $i$ increase.
Algorithm 3 Continuation Newton methods with the deflation technique and multi-start points for finding multi-stationary points (CNMDTM)

Input:
- the gradient $g(x) = \nabla f(x)$ of the objective function $f: \mathbb{R}^n \to \mathbb{R}$, the tolerance $\varepsilon$ (optional), $L$ initial points $x_{\text{init}}^1, \ldots, x_{\text{init}}^L$ (optional), the Hessian matrix $H(x)$ of $f(x)$ (optional).

Output:
- All found stationary points $x_1^*, \ldots, x_{\text{nsp}}^*$ of $f(x)$, and its global optimal approximation point $x_{\text{ap}}^*$.

1: Set the default tolerance $\varepsilon = 10^{-6}$.
2: Set the default initial points $x_{\text{init}}^i$ ($i = 1, \ldots, L$) by equation (29), where $L = 6$.
3: Let the Boolean variable $\text{success}$ be false.
4: Set $\text{nsp} = 0$ ($\text{nsp}$ represents the number of found stationary points of $f(x)$).
5: for $i = 1, \ldots, L$ do
6: Select the $i$-th initial point as $x_0 = x_{\text{init}}^i$ and call Algorithm 1 to find a stationary point of $f(x)$.
7: if a stationary point of $f(x)$ is found by Algorithm 1 successfully then
8: Let the Boolean variable $\text{success}$ be true.
9: Save this found stationary point of $f(x)$ as $x_1^*$.
10: Set $\text{nsp} = 1$.
11: break;
12: else
13: Let the Boolean variable $\text{success}$ be false.
14: end if
15: end for
16: if the Boolean variable $\text{success}$ is true then
17: Set $i = 1$ ($i$ represents the order of given initial points).
18: while $i \leq L$ do
19: Select the $i$-th initial point as $x_0 = x_{\text{init}}^i$ and call Algorithm 2 to find a new stationary point.
20: if a new stationary point of $f(x)$ is found successfully then
21: Save this newly found stationary point as $x_{\text{nsp}}^*$.
22: if this newly found stationary point $x_{\text{nsp}}^*$ almost equals the initial point $x_{\text{init}}^i$ then
23: Set $i = i + 1$.
24: end if
25: Set $\text{nsp} = \text{nsp} + 1$.
26: else
27: Set $i = i + 1$.
28: end if
29: end while
30: end if
31: if $\text{nsp} > 0$ then
32: Set $f_{\text{ap}}^* = f(x_1^*)$ and $x_{\text{ap}}^* = x_1^*$.
33: for $k = 2, \ldots, \text{nsp}$ do
34: Set $f_k^* = f(x_k^*)$.
35: if $f_k^* < f_{\text{ap}}^*$ then
36: Set $x_{\text{ap}}^* = x_k^*$ and $f_{\text{ap}}^* = f_k^*$.
37: end if
38: end for
39: Let the Boolean variable $\text{success}$ be true.
40: Output all found stationary points $x_1^*, \ldots, x_{\text{nsp}}^*$ of $f(x)$, its global optimal approximation point $x_{\text{ap}}^*$, its optimal approximation value $f_{\text{ap}}^*$, and the Boolean variable $\text{success}$.
41: else
42: Set $x_0 = x_0$ and $f_{\text{ap}} = f(x_0)$.
43: Let the Boolean variable $\text{success}$ be false.
44: Output $x_{\text{ap}}^*$, $f_{\text{ap}}^*$ and the Boolean variable $\text{success}$.
45: end if
by 1 and select the next initial point. In lines 22-23, they mean that we need to select the next initial point if the newly found stationary point almost equals the i-th initial point \( x_{i}^{\text{init}} \).

In lines 31-45, if \( nsp \) (the number of found stationary points) is greater than 0, we let the Boolean variable success be true and output all found stationary points \( x_{1}^{*}, \ldots, x_{nsp}^{*} \), the optimal approximation point \( x_{ap}^{*} = \arg\min \{ f(x_{1}^{*}), \ldots, f(x_{nsp}^{*}) \} \), and the Boolean variable success. Otherwise, we let Boolean variable success be false and output the optimal approximation point \( x_{ap}^{*} = x_{0} \), and the Boolean variable success.

4 The quasi-genetic evolution

By using the continuation Newton method with the deflation technique and multi-start points (Algorithm 3) in Section 3, we find the most of stationary points \( x_{1}^{*}, \ldots, x_{nsp}^{*} \) and obtain the suboptimal value \( f(x_{ap}^{*}) \). Algorithm 3 is essentially a local search method, and it can not ensure that \( f(x_{ap}^{*}) \) is the global minimum of \( f(x) \) except that all stationary points of \( f(x) \) have been found by Algorithm 3. Therefore, in order to find the global minimum at all possible, we use the idea of memetic algorithms [65,78] to approach the global minimum further. Namely, we use the property that evolutionary algorithms escape from the local minima. And we revise the heuristic crossover of the genetic algorithm such that it evolves from the found stationary points \( x_{1}^{*}, \ldots, x_{nsp}^{*} \) of \( f(x) \). After it evolves several generations such as twenty generations, we obtain an evolutionary suboptimal solution \( x_{ag}^{*} \) of \( f(x) \). Then, we use it as the initial point of the continuation Newton method (i.e. CNMT, Algorithm 1) to refine it and obtain a better approximation solution \( x_{min}^{*} \) of the global minimum point.

In order to ensure the effect of the evolutionary algorithm, the number of population (we denote it as \( L \)) can not be too small. In general, \( L \) is greater than 20. Since the number of found stationary points may be less than \( L \), we supplement \( L \) points as the candidate seeds of the evolutionary algorithm.

We define the \( n/2 \times 1 \) vector \( e = (1, \ldots, 1)^{T} \). Then, we select the following five basic seeds:

\[
x_{0}^{bs} = 0, \quad x_{1}^{bs} = \left( \begin{array}{c} e \\ e \end{array} \right), \quad x_{2}^{bs} = \left( \begin{array}{c} e \\ -e \end{array} \right), \quad x_{3}^{bs} = \left( \begin{array}{c} -e \\ e \end{array} \right), \quad x_{4}^{bs} = \left( \begin{array}{c} -e \\ -e \end{array} \right).
\]

We denote the set \( S_{0}^{bs} \) as

\[
S_{0}^{bs} = \{ x_{1}^{bs}, x_{2}^{bs}, x_{3}^{bs}, x_{4}^{bs} \}.
\]

From equations (30)-(31), we supplement \( L \) candidate seeds such as

\[
S_{0}^{\text{seed}} = \left\{ 0, 10^{-1} \times S_{0}^{bs}, \ldots, 10^{2} \times S_{0}^{bs}, 10^{3} \times S_{0}^{bs}, \ldots \right\}.
\]

In practice, we select the first \( L \) minimum points of the objective function from the set

\[
S_{0} = \{ x_{1}^{*}, \ldots, x_{nsp}^{*}, S_{0}^{\text{seed}} \}
\]
and denote these \( L \) points as \( x^0_i \) \((i = 1, \ldots, L)\), where the set \( S^\text{seed}_0 \) is defined by equation (32). We denote
\[
S_0 = \{x^0_1, \ldots, x^0_L\}
\]
and adopt them as the initial seeds of the quasi-genetic algorithm.

Secondly, we select any two elements \( x^0_i \) and \( x^0_j \) from the seed set \( S_0 \), and use the convex crossover to generate the next offspring. Namely, we set
\[
\bar{x}^1_k = \frac{1}{2}(x^0_i + x^0_j), \quad i = 1, \ldots, L, \quad j = i, i + 1, \ldots, L, \quad k = 1, 2, \ldots, \frac{L(L-1)}{2}.
\]
Similarly, we select the first \( L \) minimum points of \( f(x) \) from the set
\[
\tilde{S}_1 = S_0 \bigcup \{\bar{x}^1_1, \ldots, \bar{x}^1_{L(L-1)/2}\}
\]
and denote these \( L \) points as \( x^1_i \) \((i = 1, \ldots, L)\). We set
\[
S_1 = \{x^1_1, \ldots, x^1_L\}
\]
and regard them as the dominant population.

Repeatedly, after the \( l \)-th evolution, we generate the offspring \( \{x^{l+1}_1, \ldots, x^{l+1}_{L(L-1)/2}\} \) and select the first \( L \) minimum points of \( f(x) \) from the set
\[
\tilde{S}_{l+1} = S_l \bigcup \{x^{l+1}_1, \ldots, x^{l+1}_{L(L-1)/2}\}, \quad l = 0, 1, \ldots
\]
We denote these \( L \) points as \( x^l_i \) \((i = 1, \ldots, L)\), and set
\[
S_{l+1} = \{x^{l+1}_1, \ldots, x^{l+1}_L\}, \quad l = 0, 1, \ldots
\]
After it evolves \( N \) generations, we stop the evolution and select the minimum point of \( f(x) \) from the set \( S_N \). We denote this minimum point as \( x^{*\text{ag}} \).

Finally, in order to improve the convergence of the quasi-genetic evolution, we call Algorithm 1 (CNMTr) from \( x^{*\text{ag}} \) to find the stationary point of \( f(x) \). We denote this found stationary point as \( x^{*\text{cn}} \). Then, we select the minimum point \( x^{*\text{min}} \) of \( f(x) \) between \( x^{*\text{cn}} \) and \( x^{*\text{ag}} \), i.e.
\[
x^{*\text{min}} = \arg\min \{f(x^{*\text{cn}}), f(x^{*\text{ag}})\},
\]
and output \( x^{*\text{min}} \) as the global optimal approximation point of \( f(x) \). According to the above discussions, we give the detailed descriptions of the quasi-genetic algorithm in Algorithm 4.

In order to understand Algorithm 4 better, we describes how it works step by step and give some explanations of the key variables. \text{msp} represents the number of the known stationary points of \( f(x) \). In line 2, \text{L} represents the number of population and its default value equals 21. It supplements \( L \) candidate seeds \( x^{\text{init}}_1, \ldots, x^{\text{init}}_L \) from the
Algorithm 4 The quasi-genetic evolution for global optimization problems (QGE)

Input:
the gradient $g(x) = \nabla f(x)$ of $f(x)$, the known stationary points $x_1^0, \ldots, x_n^0$ of $f(x)$ and $L$ initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ (optional), the tolerance $\varepsilon$ (optional), the Hessian matrix $H(x)$ of $f(x)$ (optional).

Output:
the global optimal approximation point $x_{\text{min}}$ of $f(x)$.

1: Set the default tolerance $\varepsilon = 10^{-6}$
2: Set $L = 21$ and select the default initial points $x_i^{\text{init}}$ $(i = 1, \ldots, L)$ from $\mathcal{S}_0^{\text{init}}$ defined by equation (32).
3: Set $l = 0$ and $N = 20$ ($N$ is the maximum number of evolutionary generations).
4: Select the first $L$ minimum points of $f(x)$ from the set $\mathcal{S}_0$ defined by equation (33), and denote these $L$ points as $x_i^0$ $(i = 1, \ldots, L)$.
5: Set $\mathcal{S}_0 = \{x_1^0, \ldots, x_L^0\}$.
6: while $(l \leq N)$ do
7: \hspace{1em} Set $k = 1$.
8: \hspace{2em} for $i = 1, \ldots, L$ do
9: \hspace{3em} for $j = (i+1), \ldots, L$ do
10: \hspace{4em} Set $x_i^{k+1} = \frac{1}{2} (x_i^k + x_j^k)$.
11: \hspace{2em} end for
12: \hspace{1em} end for
13: \hspace{1em} Select the first $L$ minimum points of $f(x)$ from the set $\mathcal{S}_{l+1}$ defined by equation (38), and denote these $L$ points as $x_i^{k+1}$ $(i = 1, \ldots, L)$.
15: \hspace{1em} Set $\mathcal{S}_{l+1} = \{x_1^{k+1}, \ldots, x_L^{k+1}\}$.
16: \hspace{1em} Set $l \leftarrow l + 1$.
17: end while
18: Set $f_{ag}^* = f(x_{ag}^*)$ and $x_{ag}^* = x_i^*$. Use $x_{ag}$ as the initial point and call Algorithm 1 to find a stationary point $x_{ag}^*$ of $f(x)$.
20: Set $x_{\text{min}} = \arg\min \{f(x_{ag}^*), f(x_{ag}^*)\}$.
21: Output $x_{\text{min}}$ as the global optimal approximation point.

set $\mathcal{S}_0^{\text{seed}}$ defined by equation (32). In line 3, $N$ represents the maximum number of evolutionary generations, and its default value equals 20. In line 4, it selects the first $L$ minimum points of $f(x)$ from the set $\mathcal{S}_0$ defined by equation (33) as the initial seeds of the evolutionary algorithm, and denotes the first $L$ minimum points as $x_1^0, x_2^0, \ldots, x_L^0$.

In lines 6-17, they give the crossover operation to generate the next offspring, and select the first $L$ dominant offspring as the evolutionary population. After it evolves $N$ generations, the evolution will stop. In line 18, it selects the minimum point $x_{ag}$ from the final evolutionary offspring $\{x_1^N, \ldots, x_L^N\}$. In line 19, it uses $x_{ag}^*$ as the initial point of Algorithm 1 (CNMTr) and call Algorithm 1 to find a stationary point $x_{ag}^*$ of $f(x)$. In lines 20-21, they select the minimum from $\{f(x_{ag}^*), f(x_{ag}^*)\}$, and denote this minimum as $f(x_{\text{min}}^*)$. Finally, it outputs $x_{\text{min}}^*$ as the global optimal approximation solution of $f(x)$.

5 The implementation of CNMGE

In this section, according to the discussions of previous sections, we give the implementation description of the continuation Newton method with the deflation tech-
nique and the quasi-genetic evolution for global optimization problems. We denote this method as CNMGE. Its implementation diagram is illustrated by Figure 1. Firstly, CNMGE calls the subroutine CNMDTM (Algorithm 3) to find the stationary points of $f(x)$ from multi-start points as many as possible. Then, it calls QGE (Algorithm 4) and uses those found stationary points as the initial seeds of QGE to approach the global optimal approximation $x_{ag}^*$ of $f(x)$. Finally, it calls CNMTr (Algorithm 1) to refine $x_{ag}^*$ and obtain its refined point $x_{cn}^*$. It outputs $x^* = \arg\min \{ f(x_{ag}^*), f(x_{cn}^*) \}$ as the global optimal approximation point of $f(x)$. From Figure 1 and Algorithm 3, we know that the subroutine CNMDTM firstly calls the subroutine CNMTr (Algorithm 1) to find a stationary point. Then, it repeatedly calls the subroutine CNMDT (Algorithm 2) to find a new stationary point of $f(x)$ from multi-start points, and saves all found stationary points $x_1^*, \ldots, x_{nsp}^*$ as the input of the subroutine QGE.

![Fig. 1: The implementation diagram of CNMGE.](image)

In order to retain the usability of derivative-free methods, we use the automatic differentiation technique [9, 31, 67, 86, 87] with the reverse mode to compute the gradient $\nabla f(x)$ of $f(x)$ in Algorithm 1 (CNMTr) and Algorithm 2 (CNMDT). In general, the time that it takes to calculate the gradient of the objective function $f(x)$ by automatic differentiation with the reverse mode is about a constant (less than four) multiple of the function cost [9, 31, 67, 87]. We implement it via calling the built-in subroutine prob2struct.m of MATLAB 2021b [61]. At the end of this section, we give an example to show how it works.
In Algorithm 1 and Algorithm 2, in order to find a stationary point of \( f(x) \), it requires the Hessian matrix \( H(x) \) of \( f(x) \). In practice, we replace the Hessian matrix \( H(x) \) with its finite difference approximation as follows:

\[
H(x_k) \approx \left[ \frac{g(x_k + \varepsilon e_1) - g(x_k)}{\varepsilon}, \ldots, \frac{g(x_k + \varepsilon e_n) - g(x_k)}{\varepsilon} \right],
\]

where \( e_i \) represents the unit vector whose elements equal zeros except for the \( i \)-th element which equals 1, and \( \varepsilon = 2 \times 10^{-8} \).

It is worthwhile to discuss the selection of parameters in Algorithm 1 and Algorithm 2. In practice, we select the parameters \( \eta_1 = 1/4, \eta_2 = 3/4, c_1 = 1/2, c_2 = 2 \). According to our numerical experiments, those selected parameters work well. Since the continuation Newton flow defined by equation (5) varies dramatically in the transient phase, in order to follow its trajectory accurately, we choose the small initial time step \( \Delta t_{\text{init}} \) such as \( \Delta t_{\text{init}} = 10^{-2} \). This selection strategy of the initial time step is different to that of the line search method. The initial step length of the line search method is tried from \( \alpha_0 = 1 \) such that it mimics the Newton method and achieves the fast convergence rate near the stationary point [70, 81].

We write a software package to implement CNMGE with the MATLAB language and it can be executed in the MATLAB 2020b environment or after this version. For ease of use, we give an example to illustrate how to use it.

**Rosenbrock function** [74]

\[
f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2.
\]

It is not difficult to know that the global minimum of the Rosenbrock function (41) is located at \( x^* = (1, 1) \) and \( f(x^*) = 0 \). Its user guide is described in **Example 5.1** and we give some explanations as follows.

**Example 5.1** The user guide of CNMGE

1: \( n = 2; \)
2: \( x_0 = 2 * \text{ones}(n, 1); \)
3: \( \ % \text{Define the objective function.} \)
4: \( x = \text{optimvar}('x', n, 1); \)
5: \( \text{objfun} = 100 * (x(2)^2 - x(1))^2 + (x(1) - 1)^2; \)
6: \( lb = -\text{inf}; \)
7: \( ub = \text{inf}; \)
8: \( \text{unitdisk} = x(1)^2 < = 1; \)
9: \( \text{prob} = \text{optimproblem}('\text{Objective}', \text{objfun}, '\text{Constraints}', \text{unitdisk}); \)
10: \( \text{problem} = \text{prob2struct}('\text{problem}客观\text{Derivative}', 'auto-reverse'); \)
11: \( [x_{\text{opt}}, f_{\text{opt}}, \text{CPU_time}] = \text{CNMGE}('\text{problem.objective}', x_0, ub, lb); \)
12: \( \ % \text{Output the global minimum value computed by CNMGE.} \)
13: \( \text{fprintf}(''The global minimum value computed by CNMGE is %12.8f'' \text{n}, f_{\text{opt}}); \)
14: \( \ % \text{Output the computational time of CNMGE.} \)
15: \( \text{fprintf}(''The computational time of CNMGE is %8.4f'' \text{n}, \text{CPU_time}); \)
In line 4, the subroutine `optimvar` defines an $n \times 1$ vector. In line 5, it defines an objective function such as $f(x) = 100((x_2 - x_1^2)^2 + (x_1 - 1)^2)$. In line 6, it defines the lower bound as $-\infty$. In line 7, it defines the upper bound as $\infty$. In line 8, it defines a constraint $x_1^2 \leq 1$ (if it does not define this constraint, the automatic differentiation function `prob2struct` will generate the incorrect gradient of the objective function for the nonlinear least-square problem). In line 9, the subroutine `optimproblem` assigns the objective function and the constraints of an optimization problem. In line 10, the subroutine `prob2struct` generates a standard MATLAB function `generatedObjective` including the objective function and its gradient of an optimization problem. Its gradient is computed by automatic differentiation with the reverse mode. In line 11, it calls the `CNMGE` solver to find an optimal approximation solution of the global optimization problem.

6 Numerical experiments

In this section, some numerical experiments are conducted to test the performance of the continuation Newton method with the deflation technique and the quasi-genetic evolution (CNMGE) for global optimization problems. In Subsection 6.1, we verify the performance improvement of the algorithm combination. Namely, we compare the continuation method with the multi-start method (CNMTrM, Algorithm 1 with the multi-start method), the continuation method with the revised deflation technique (CNMDTM, Algorithm 3), and the continuation method with the revised deflation technique and the evolutionary algorithm (CNMGE, the combination of Algorithm 3 and Algorithm 4) for 68 open test problems.

In Subsection 6.2, in order to test the total performance of CNMGE, we compare it with other representative global optimization methods such as the multi-start methods (the built-in subroutine GlobalSearch.m of MATLAB R2021b [61, 85], GLODS: Global and Local Optimization using Direct Search [20], VRBBO: Vienna Randomized Black Box Optimization [43]), the branch-and-bound method (Couenne: Convex Over- and Under-ENvelopes for Nonlinear Estimation, one of the state-of-the-art open-source solver [10, 17]), and the derivative-free algorithms (CMA-ES [32, 33] and MCS [37, 68]).

6.1 Verifying the effect of the algorithm combination

In this subsection, in order to verify the performance improvement of the algorithm combination, we compare the continuation method [53, 56] with the multi-start method (CNMTrM, Algorithm 1 with the multi-start method), the continuation method with the revised deflation technique (CNMDTM, Algorithm 3), and the continuation method with the revised deflation technique and the evolutionary algorithm (CNMGE, the combination of Algorithm 3 and Algorithm 4) for 68 open test problems. The test problems can be found in [2, 3, 27, 45, 64, 77] and their scales vary from dimension 1 to 1000. In order to evaluate the effect of the algorithm gradient computed by automatic differentiation, we also compare CNMGE using the algorithm gradient com-
puted by automatic differentiation (we still denote it as CNMGE) with CNMGE using the analytical gradient (we denote it as CNMGE\(_{AG}\)). The termination condition of Algorithm 1 for finding a stationary point of the objective function \(f(x)\) is

\[
\|\nabla f(x_k)\|_\infty \leq 10^{-6}.
\]

(42)

The codes are executed by a HP notebook with the Intel quad-core CPU and 8Gb memory in the MATLAB2021b environment [61]. The numerical results are arranged in Tables 1-4 and Figures 2-3. Since the global minima of some test problems are unknown, we mark their global minima of those problems with the notation “\(\_\)” in Tables 1-4. For those problems, we regard that the method fails to find their global minima if the found minima are greater than those of other methods.

From Tables 1-5, we find that CNMGE and CNMGE\(_{AG}\) work well and obtain the global minima of all 68 test problems. CNMTrM fails to find the global minima of 27 test problems about 39.71\%. And CNMDTM fails to find the global minima of 6 test problems about 8.82\%. Therefore, the performance of the algorithm combination gradually becomes better and better from the continuation Newton method with the multi-start method (CNMTrM, Algorithm 1), CNMTr with the deflation technique (CNMDTM, Algorithm 3), and CNMDTM with the evolutionary algorithm (CNMGE, the combination of Algorithm 3 and Algorithm 4).

We also find that CNMGE will not sacrifice its performance if it replaces the analytical gradient with the algorithm gradient computed by the automatic differentiation technique with the reverse mode, and replaces the Hessian matrix \(\nabla^2 f(x)\) with its difference approximation (40). We find that CNMGE and CNMGE\(_{AG}\) almost take the same time to compute a problem from Figures 2-3. Consequently, CNMGE with the automatic differentiation technique retains the usability of the derivative-free method and has the fast convergence of the gradient-based method.

6.2 Testing the total performance of CNMGE

In this subsection, in order to evaluate the total performance of CNMGE, we compare it with other representative global optimization methods such as the multi-start algorithms (the built-in subroutine GlobalSearch.m of MATLAB R2021b [61, 85], GLODS [20], and VRBBO [43]), the branch-and-bound method (Couenne, one of the state-of-the-art open-source solver [10, 17]), and the derivative-free algorithms (CMA-ES [32, 33] and MCS [37, 68]). The codes are executed by a HP notebook with the Intel quad-core CPU and 8Gb memory except for Couenne, which is executed by the NEOS server [19, 23, 29, 69].

GlobalSearch [61, 85] is an efficient multi-start method for global optimization problems and it is widely used in engineering fields [61]. GLODS (Global and Local Optimization using Direct Search [20]) and VRBBO (Vienna Randomized Black Box
Table 1: Numerical results of large-scale problems (no.1-17) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

| Problem | CNMTrM | CNMDTM | CNMGE | CNMGE_AG | Global minimum |
|---------|--------|--------|-------|----------|----------------|
|         | \( \min f(x) \) (CPU time (s)) | \( \min f(x) \) (CPU time (s)) | \( \min f(x) \) (CPU time (s)) | \( \min f(x) \) (CPU time (s)) |                      |
| 1. Molecular Energy (n = 1000) [45] | -2.8789E-01 (failed) | -4.1118E+00 (1.21E+01) | -4.1118E+00 (1.59E+01) | -4.1118E+00 (2.49E+01) | -4.1118E+00 |
| 2. Ackley Function (n = 1000) [77] | 3.5744E+00 (1.59E+02) (failed) | 1.1513E-15 (3.622E+02) | 1.4997E-32 (1.60E+02) | 1.4998E-32 (1.50E+02) | 0 |
| 3. Levy Function (n = 1000) [77] | 8.817E+00 (1.60E+02) (failed) | 1.2728E-02 (6.83E+01) | 1.2728E-02 (6.37E+02) | 1.2728E-02 (3.03E+02) | 1.2728E-02 |
| 4. Schwefel Function (n = 1000) [77] | -4.1118E+00 (failed) | 8.817E+00 (1.60E+02) | 8.817E+00 (1.60E+02) | 8.817E+00 (1.60E+02) | 0 |
| 5. Rastrigin Function (n = 1000) [77] | 9.9495E+00 (5.78E+01) (failed) | 1.2728E-02 (6.83E+01) | 1.2728E-02 (6.37E+02) | 1.2728E-02 (3.03E+02) | 1.2728E-02 |
| 6. Styblinski-Tang Function (n = 1000) [77] | 2.1713E+00 (5.78E+01) (failed) | 1.4997E-32 (1.60E+02) | 1.4997E-32 (1.50E+02) | 1.4997E-32 (1.50E+02) | 0 |
| 7. Trid Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 8. Sum Squares Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 9. Sphere Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 10. Rotated Hyper-Ellipsoid Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 11. Zakharov Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 12. Dixon-Price Function (n = 1000) [77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 13. Rosenbrock Function (n = 1000) [64, 77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 14. Powell Function (n = 1000) [64, 77] | 1.2528E+01 (4.72E+00) (failed) | 0 (4.33E+00) | 0 (4.33E+00) | 0 (4.33E+00) | 0 |
| 15. Quartic With Noise Function (n = 1000) [3] | 5.000E+00 (2.01E+00) (failed) | 0 (2.03E+01) | 0 (2.03E+01) | 0 (2.03E+01) | 0 |
| 16. Schubert Function (n = 1000) [3] | 3.494E+03 (3.22E+00) (failed) | -1.2031E+04 (1.87E+02) | -1.2031E+04 (4.12E+02) | -1.2031E+04 (4.12E+02) | -1.2031E+04 |
| 17. Raydan 1 Function (n = 1000) [2] | 5.0050E+04 (3.95E+01) (failed) | 0 (2.03E+01) | 0 (2.03E+01) | 0 (2.03E+01) | 0 |
Table 2: Numerical results of large-scale problems (no. 18-34) computed by CNMGE, CNMTrM, CNMDTM and CNMGE-AG.

| Problem | CNMTrM | CNMDTM | CNMGE | CNMGE-AG | Global minimum |
|---------|--------|--------|-------|----------|----------------|
|         | min \( f(x) \) (CPU time (s)) | min \( f(x) \) (CPU time (s)) | min \( f(x) \) (CPU time (s)) | min \( f(x) \) (CPU time (s)) |
| 18. Raydan 2 Function \((n = 1000) \) \[2\] | 1.0000E+03 (7.61E-01) | 1.0000E+03 (2.50E+00) | 1.0000E+03 (2.15E+00) | 1.0000E+03 (2.64E+00) |
| 19. Extended Tridiagonal 1 Function \((n = 1000) \) \[2\] | 1.6699E-12 (1.51E+01) | 5.9698E-19 (6.15E+01) | 5.9698E-19 (6.29E+01) | 5.9598E-19 (1.19E+02) |
| 20. Extended quadratic penalty QP1 Function \((n = 1000) \) \[2\] | 3.9900E+03 (1.15E+01) | 3.9900E+03 (2.13E+00) | 3.9900E+03 (5.89E+00) | 3.9900E+03 (6.02E+00) |
| 21. Extended quadratic penalty QP2 Function \((n = 1000) \) \[2\] | 6.7752E-19 (2.09E+01) | 5.1228E-19 (3.14E+01) | 5.1228E-19 (4.75E+01) | 3.7952E-21 (1.57E+02) |
| 22. Quadratic QF2 Function \((n = 1000) \) \[2\] | -1.0001E+00 (1.62E+00) | -1.0001E+00 (1.39E+01) | -1.0001E+00 (1.44E+01) | -1.0001E+00 (1.53E+02) |
| 23. Extended PSC1 Function \((n = 1000) \) \[2\] | 3.8659E+02 (8.05E+00) | 3.8659E+02 (3.93E+00) | 3.8659E+02 (6.83E+00) | 3.8659E+02 (5.96E+00) |
| 24. Extended BD1 Function \((n = 1000) \) \[2\] | 2.3356E-14 (3.14E+00) | 3.2414E-23 (4.07E+01) | 0 (3.98E+01) | 0 (3.38E+01) |
| 25. Extended Cliff Function \((n = 1000) \) \[2\] | 9.8933E+01 (4.69E+00) | 9.8933E+01 (7.37E+01) | 9.8933E+01 (7.66E+01) | 9.8933E+01 (5.16E+01) |
| 26. Perturbed quadratic diagonal Function \((n = 1000) \) \[2\] | 1.9012E-15 (1.10E+01) | 2.5710E-15 (3.59E+00) | 0 (3.92E+02) | 0 (2.02E+01) |
| 27. Extended Hiebert Function \((n = 1000) \) \[2\] | 1.3081E-18 (7.78E+00) | 1.3081E-18 (8.29E+01) | 5.4874E-19 (8.85E+01) | 5.4874E-19 (5.97E+01) |
| 28. Extended TET Function \((n = 1000) \) \[2\] | 1.2796E+03 (3.92E+00) | 1.2796E+03 (3.06E+01) | 1.2796E+03 (3.58E+01) | 1.2796E+03 (1.63E+01) |
| 29. Diagonal 1 Function \((n = 1000) \) \[2\] | -2.7068E+06 (2.03E+00) | -2.7068E+06 (8.74E+00) | -2.7068E+06 (1.06E+01) | -2.7068E+06 (1.40E+01) |
| 30. Diagonal 3 Function \((n = 1000) \) \[2\] | -4.9575E+05 (2.05E+03) | -4.9575E+05 (1.67E+01) | -5.0050E+05 (2.04E+01) | -5.0050E+05 (3.05E+01) |
| 31. Diagonal 5 Function \((n = 1000) \) \[2\] | 6.9314E+02 (8.92E-01) | 6.9314E+02 (3.10E+01) | 6.9314E+02 (6.20E-01) | 6.9314E+02 (5.18E-01) |
| 32. Extended Maratos Function \((n = 1000) \) \[2\] | -5.0031E+02 (1.33E+01) | -5.0031E+02 (6.75E+00) | -5.0031E+02 (7.20E+00) | -5.0031E+02 (4.94E+01) |
| 33. E2 Function \((n = 1000) \) \[2\] | -9.9884E+02 (9.06E+00) | -9.9850E+02 (7.42E+01) | -9.9884E+02 (1.14E+03) | -9.9590E+02 (3.47E+01) |
| 34. SINQUAD Function \((n = 1000) \) \[2\] | 0 (3.57E+01) | 0 (9.11E+01) | 0 (9.12E+01) | 0 (5.34E-01) |
Table 3: Numerical results of small-scale problems (no.35-50) computed by CNMGE, CNMTrM, CNMDTM and CNMGE-AG.

| Problem | min f(x) (CPU time (s)) | min f(x) (CPU time (s)) | CNMGE AG f(x) (CPU time (s)) | Global minimum |
|---------|-------------------------|-------------------------|-------------------------------|----------------|
| 35. Griewank Function (n = 10) [64,77] | 1.0030E+00 (2.24E-01) | 2.7755E-15 (1.64E+00) | 0 (1.98E+00) | 0 (1.95E-02) |
| 36. Levy Function N. 13 (n = 2) [77] | 9.3717E-01 (1.62E-02) | 5.9152E-22 (3.11E-01) | 1.3497E-31 (3.87E-01) | 1.3497E-31 (2.18E-01) |
| 37. Hosaki Function (n = 2) [3] | -1.1277E+00 (2.55E-02) | -2.7434E+15 (1.24E+00) | -2.7434E+15 (1.40E+00) | -1.2200E+07 (1.98E-02) |
| 38. Beale Function (n = 2) [64,77] | 2.4143E-16 (1.81E-02) | 3.8283E-13 (1.04E-01) | 4.5343E-14 (2.38E-01) | 4.3287E-14 (7.00E-02) |
| 39. Easom Function (n = 2) [77] | -1.3953E-07 (1.17E-02) | -9.70E-01 (1.93E+00) | -1.0000E+00 (1.97E+00) | -1.0000E+00 (8.44E-01) |
| 40. Price Function (n = 2) [3] | 9.7529E-13 (1.80E-02) | 9.7529E-13 (3.79E-02) | 0 (3.76E-02) | 0 (2.04E-02) |
| 41. Branin Function (n = 2) [77] | 3.9788E-01 (1.40E-02) | 3.9788E-01 (3.45E-02) | 3.9788E-01 (4.42E-02) | 3.9788E-01 (1.17E-02) |
| 42. Trecanni Function (n = 2) [3] | 6.8029E-17 (1.06E-02) | 3.5042E-15 (2.20E-02) | 0 (2.66E-02) | 0 (1.08E-02) |
| 43. Booth Function (n = 2) [77] | 3.4010E-16 (9.12E-03) | 1.3604E-15 (1.84E-02) | 1.3604E-15 (3.58E-02) | 1.3604E-15 (9.17E-03) |
| 44. Matyas Function (n = 2) [77] | 1.8783E-14 (1.01E-02) | 1.8783E-14 (1.22E-02) | 0 (3.61E-02) | 0 (4.41E-03) |
| 45. McCormack Function (n = 2) [77] | -1.9132E+00 (1.41E-02) | -1.9132E+00 (5.59E-01) | -1.9132E+00 (5.67E-01) | -1.9132E+00 (3.30E-01) |
| 46. Power Sum Function (n = 4) [77] | 3.7740E-09 (1.93E-02) | 4.2909E-04 (1.29E-01) | 4.2909E-04 (1.86E-01) | 2.1824E-11 (2.15E-01) |
| 47. Colville Function (n = 4) [77] | 7.7098E-18 (3.99E-02) | 1.0131E-17 (8.98E-02) | 0 (1.06E-01) | 0 (1.73E-02) |
| 48. Schaffer Function N. 2 (n = 2) [77] | 9.9754E-01 (3.50E-02) | 4.6240E-14 (2.66E+00) | 0 (2.48E+00) | 0 (6.04E-01) |
| 49. Bohachevsky Function (n = 2) [77] | 3.5183E+00 (1.85E-02) | 4.12E-01 (3.80E-01) | 0 (3.93E-01) | 0 (1.16E-01) |
| 50. Three-Hump Camel Function (n = 2) [77] | 2.9863E+00 (2.40E-02) | 1.1706E-14 (8.68E-02) | 0 (9.51E-02) | 0 (1.51E-02) |
Table 4: Numerical results of small-scale problems (no.51-68) computed by CNMGE, CNMTrM, CNMDTM and CNMGE-AG.

| Problem | CNMTrM | CNMDTM | CNMGE | CNMGE-AG |
|---------|--------|--------|-------|----------|
|         | min(f(s)) | min(f(s)) | min(f(s)) | min(f(s)) |
|         | (CPU time (s)) | (CPU time (s)) | (CPU time (s)) | (CPU time (s)) |
| 51. Six-Hump Camel Function (n = 2) [77] | 5.4371E-01 | -1.0316E+00 | -1.0316E+00 | -1.0316E+00 |
|         | (3.06E-02) | (9.84E-02) | (5.39E-02) | -1.0316E+00 |
| 52. Drop-Wave Function (n = 2) [77] | -8.6245E-15 | -1.0000E+00 | -1.0000E+00 | -1.0000E+00 |
|         | (3.50E-02) | (4.16E+00) | (3.88E+00) | (8.18E-02) |
| 53. Perm Function 0, d, β (n = 4) [77] | 0 | 0 | 0 | 0 |
|         | (4.93E-02) | (3.68E-01) | (3.82E-01) | (1.94E-01) |
| 54. Hartmann 3-D Function (n = 3) [77] | -2.8731E-08 | -3.8627E+00 | -3.8627E+00 | -3.8627E+00 |
|         | (6.28E-02) | (2.98E+00) | (2.98E+00) | (1.92E+00) |
| 55. Trefethen 4 Function (n = 2) [3] | -1.8895E+00 | -2.4275E+00 | -3.1407E+00 | -2.6634E+00 |
|         | (2.57E-02) | (3.88E+00) | (3.88E+00) | (8.18E-02) |
| 56. Zettl Function (n = 2) [3] | 3.7912E-03 | 3.7912E-03 | 3.7912E-03 | 3.7912E-03 |
|         | (1.65E-02) | (4.36E-02) | (4.36E-02) | (1.94E-02) |
| 57. Exp2 Function (n = 2) [3] | 1.9476E+00 | 1.9476E+00 | 3.6953E-14 | 3.6953E-14 |
|         | (4.38E-02) | (1.83E-01) | (1.83E-01) | (2.17E-01) |
| 58. Hansen Function (n = 2) [77] | -1.7329E+00 | -inf | -6.5535E+04 | -inf |
|         | (6.27E-02) | (1.20E+00) | (1.21E+00) | (2.13E-01) |
| 59. Schaffer Function N. 4 (n = 2) [77] | -2.6977E+01 | -1.7654E+02 | -1.7654E+02 | -1.7654E+02 |
|         | (1.82E-02) | (2.15E+00) | (2.15E+00) | (2.84E+00) |
| 60. Holder Table Function (n = 2) [77] | -1.7329E+00 | -inf | -6.5535E+04 | -inf |
|         | (6.27E-02) | (1.20E+00) | (1.21E+00) | (2.13E-01) |
| 61. Gramacy & Lee (2012) Function (n = 1) [77] | 4.7641E-01 | -8.0130E-01 | -8.0130E-01 | -8.0130E-01 |
|         | (3.51E-02) | (6.77E-01) | (6.77E-01) | (6.77E-01) |
| 62. Eggholder Function (n = 2) [3] | -2.0869E-08 | -8.1500E-01 | -2.0869E-08 | -2.0869E-08 |
|         | (2.34E-02) | (4.19E-01) | (4.19E-01) | (4.19E-01) |
| 63. Michalewicz Function (n = 2) [77] | -2.0687E+03 | -2.0830E+03 | -2.0830E+03 | -2.0830E+03 |
|         | (2.94E-02) | (4.83E-01) | (4.83E-01) | (4.83E-01) |
| 64. Box-Betts Exponential Quadratic Function (n = 3) [3] | 8.3869E-17 | 8.3869E-17 | 0 | 0 |
|         | (1.97E-02) | (6.17E-02) | (7.09E-02) | (4.67E-01) |
| 65. Cross-in-Tray Function (n = 2) [77] | -2.0626E+00 | -2.0626E+00 | -2.0626E+00 | -2.0626E+00 |
|         | (2.30E-02) | (2.53E+00) | (2.53E+00) | (9.39E-01) |
| 66. Himmeblau Function (n = 2) [77] | 6.7719E+01 | 6.4197E+14 | 6.9206E+16 | 6.9206E+16 |
|         | (8.03E-02) | (8.68E-02) | (8.68E-02) | (3.82E-02) |
| 67. Forrester et al. (2008) Function (n = 1) [77] | 1.5998E+01 | -1.1490E+05 | -9.1005E+06 | -1.8288E+07 |
|         | (1.43E-02) | (5.32E-01) | (5.32E-01) | (4.78E-01) |
| 68. Goldstein-Price | 3 | 3 | 3 | 3 |
|         | (3.19E-02) | (8.22E-02) | (6.69E-02) | (2.66E-02) |
Table 5: The number of failure problems computed by CNMTrM, CNMDTM, CNMGE and CNMGE-AG.

|                      | CNMTrM | CNMDTM | CNMGE | CNMGE-AG |
|----------------------|--------|--------|-------|-----------|
| The number of failed problems | 27     | 6      | 0     | 0         |
| The probability of failure     | $\frac{27}{68} (39.71\%)$ | $\frac{6}{68} (8.82\%)$ | 0     | 0         |

Fig. 2: CPU time of large-scale problems computed by CNMGE, CNMTrM, CNMDTM and CNMGE-AG.

Fig. 3: CPU time of small-scale problems computed by CNMGE, CNMTrM, CNMDTM and CNMGE-AG.

Optimization, its MATLAB code can be downloaded from [43]) are two other representative multi-start methods. Couenne (Convex Over- and Under-ENvelopes for Nonlinear Estimation) is a state-of-the-art open source solver for global optimization problems. The covariance matrix adaptation evolution strategy (CMA-ES, its MATLAB code can be downloaded from [18]) and the multilevel coordinate search (MCS, its MATLAB code can be downloaded from [62]) are two representative derivative-free algorithms. Therefore, we select those six methods as the basis for comparison.

The numerical results are arranged in Tables 6-14. Since the global minima of some test problems are unknown, we mark their global minima of those problems with the notation $\"\}$ in Tables 6-13. For those problems, we regard that the method fails to find their global minima if the found minima are greater than those of other methods. The number of CMA-ES’s evolution generations is set to $10^5$. Due to the randomness of CMA-ES and VRBBO, we repeatedly calculate every problem about ten times with CMA-ES and VRBBO, and select the computed minima as their optimal values in Tables 6-13, respectively. The computational time of CNMGE, GlobalSearch, GLODS, VRBBO, Couenne, CMA-ES and MCS is illustrated in Figures 4-5.

When the unconstrained optimization problem (1) has many local minima, according to our numerical experiments, Algorithm 3 can find most of stationary points of $f(x)$. Thus, it makes Algorithm CNMGE possibly find the global minimum of $f(x)$ for the difficult problem. We illustrate this result via Problem 1. Problem 1 [45]
is a molecular potential energy problem and its objective function has the following form:

\[
    f(\omega) = \sum_{i=1}^{n} \left( 1 + \cos(3\omega_i) + \frac{(-1)^i}{\sqrt{10.60099896 - 4.141720682\cos(\omega_i)}} \right),
\]

(43)

where \(\omega_i\) is the torsion angle and \(n + 3\) is the number of atoms or beads in the given system. The number of its local minima increases exponentially with the size of the problem, which characterizes the principal difficult in minimizing molecular potential energy functions for the traditional global optimization methods. CNMGE can find 17 stationary points of problem (43) with \(n = 1000\) (if CNMGE uses the Hessian matrix \(H(x)\), it can find 90 stationary points) and its global minimum \(f(\omega^*) = -41.118303\) located at \(\omega^* = (1.039195, 3.141593, \ldots, 1.039195, 3.141593)\). However, for problem 1 with 50 atoms, the traditional global optimization methods such as the interval analysis method [40, 45] or the multi-start methods (GlobalSearch [61, 85], GLODS, VRBBO) are difficult to find its global minimum. Couenne, CMA-ES and MCS also fail to solve this problem.

For problems 37, 60, 62, 67, the original test problems have the default box constraints. GlobalSearch, Couenne, CMA-ES, MCS, GLODS can handle the problem with the box constraints and we use them to compute those four problems (problems 37, 60, 62, 67) with constraints. VRBBO and CNMGE can not handle the problem with the constraint and we use them compute those four problems (problems 37, 60, 62, 67) without the default box constraints. As long as they find the minima which are less than the known minima, we regard that they successfully solve those problems.

From Tables 6-14 and Figures 4-5, we find that CNMGE works well for unconstrained optimization problems and finds their global minima efficiently. From Tables 8-9 and Tables 12-13, we find that GlobalSearch, Couenne, CMA-ES and MCS all work well for small-scale problems. However, from Tables 6-7 and Tables 10-11, we find that GlobalSearch, CMA-ES and MCS fail to find the global minima of the large-scale problems about 17/34. Couenne can solve the most problems efficiently and it can not solve the large-scale problems efficiently about 12/34 (Couenne fails to solve 3 large-scale problems and 9 large-scale problems are solved by it about 8 hours). The probability of GLODS failure is 23/68 (33.82\%) and the probability of VRBBO failure is 16/68 (23.52\%). Therefore, CNMGE is more robust than other representative global optimization methods such as GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO. Consequently, CNMGE can be regarded as an alternative workhorse for the global optimization problems.
Table 6: Numerical results of large-scale problems (no.1-17) computed by CNMGE, GlobalSearch, Couenne, and CMA-ES.

| Problem                  | CNMGE | GlobalSearch | Couenne | CMA-ES | Global minimum |
|--------------------------|-------|--------------|---------|--------|---------------|
|                          | \(\min f(x)\) | \(\min f(x)\) | \(\min f(x)\) | \(\min f(x)\) |              |
|                          | (CPU time (s)) | (CPU time (s)) | (CPU time (s)) | (CPU time (s)) |              |
| 1. Molecular Energy      | -4.1118E+01 | -2.8761E-01 | -4.1118E+01 | 1.0008E+01 | -4.1118E+01 |
| Problem (n = 1000) [45] | (1.59E+01) | (1.82E+01)   | (2.68E+04)   | (2.98E+02)   |              |
| 2. Ackley Function       | 8.8817E-16 | 8.8817E-16 | 6.7078E-06  | 3.6253     | 0             |
| (n = 1000) [77]          | (1.60E+02) | (1.81E+01)   | (1.77E+03)   | (7.71E+01)   |              |
| 3. Levy Function         | 1.4997E-32 | 0.3651       | 1.1997E-15  | 1.4998E-32 | 0             |
| Problem (n = 1000) [77] | (3.67E+02) | (3.74E+00)   | (1.43E+01)   | (1.51E+02)   |              |
| 4. Scheweff Function     | 0      | 4.7903E+03   | 3.1688E+03  | 1.7096E+05 | 1.7096E+05   |
| (n = 1000) [77]          | (6.83E+01) | (4.34E+01)   | (2.42E+04)   | (2.45E+02)   | 1.2728E+02   |
| 5. Rastrign Function     | 0      | 0            | 8.5893E-26  | 1.0000E+03 | 0             |
| Problem (n = 1000) [77] | (4.33E+00) | (4.83E+00)   | (2.82E+01)   | (1.84E+02)   |              |
| 6. Styblinski-Tang       | -3.9166E+04 | 0            | 3.7885E-44  | 2.3517E-16 | 0             |
| Function (n = 1000) [77] | (2.84E+01) | (3.84E+00)   | (4.14E-02)   | (6.20E+01)   |              |
| 7. Trid Function         | 0      | 0            | 8.5893E-26  | 1.0000E+03 | 0             |
| (n = 1000) [77]          | (1.00E+01) | (3.84E+00)   | (4.14E-02)   | (6.20E+01)   |              |
| 8. Sum Squares Function  | 0      | 0            | 3.7885E-44  | 2.3517E-16 | 0             |
| (n = 1000) [77]          | (2.23E+01) | (3.22E+00)   | (2.66E+03)   | (5.91E+01)   |              |
| 9. Sphere Function       | 0      | 0            | 3.7885E-44  | 2.3517E-16 | 0             |
| (n = 1000) [77]          | (1.34E+01) | (1.54E+01)   | (3.26E-02)   | (6.64E+02)   |              |
| 10. Rotated Hyper-Ellipsoid Function (n = 1000) [77] | 0 | 0 | 3.7885E-44 | 2.3517E-16 | 0 |
| 11. Zakharov Function    | 0 | 0 | 3.7885E-44 | 2.3517E-16 | 0 |
| (n = 1000) [77]          | (6.00E+00) | (5.17E+00)   | (3.20E+00)   | (5.75E+01)   |              |
| 12. Dixon-Price Function | 6.2737E-14 | 7.2931E-01 | 1.0905E-15 | 6.6667E-01 | 0 |
| (n = 1000) [77]          | (4.96E+01) | (4.03E+00)   | (2.13E-01)   | (9.77E+01)   |              |
| 13. Rosenbrock Function  | 0 | 3.0630E+01 | 9.8899E+02 | 9.7702E+02 | 0 |
| (n = 1000) [64, 77]      | (5.17E+01) | (3.82E+00)   | (1.07E+02)   | (1.77E+02)   |              |
| 14. Powell Function      | 0 | 5.6167E-10 | 5.6167E-10 | 4.9855E-03 | 0 |
| (n = 1000) [64, 77]      | (2.03E+02) | (4.72E+00)   | (9.05E-02)   | (2.63E+02)   |              |
| 15. Quartic With Noise   | 5.0000E-01 | 5.0000E-01 | 5.0000E-01 | 5.0000E-01 | 0 |
| Function (n = 1000) [3]  | (5.53E+00) | (3.89E+00)   | (9.09E-02)   | (5.75E+01)   |              |
| 16. Schubert Function    | -1.2031E+04 | -9.9497E+03 | -1.2031E+04 | -9.9497E+03 | 0 |
| (n = 1000) [3]           | (1.87E+02) | (6.30E+01)   | (2.62E+04)   | (2.62E+04)   |              |
| 17. Raydan 1 Function    | 5.0050E+04 | 5.0050E+04 | 5.0050E+04 | 5.0050E+04 | 0 |
| (n = 1000) [2]           | (7.35E-01) | (1.03E+01)   | (1.15E-01)   | (2.83E+02)   |              |
Table 7: Numerical results of large-scale problems (no.18-34) computed by CNMGE, GlobalSearch, Couenne and CMA-ES.

| Problem | CNMGE | GlobalSearch | Couenne | CMA-ES | Global minimum |
|---------|-------|--------------|---------|--------|----------------|
|         | min f(x) (CPU time (s)) | min f(x) (CPU time (s)) | min f(x) (CPU time (s)) | min f(x) (CPU time (s)) |               |
| 18. Raydan 2 Function (n = 1000) [2] | 1.0000E+03 (2.15E+00) | 1.0000E+03 (7.60E+00) | 1.0000E+03 (9.68E-02) | 1.3038E+03 (2.93E+02) | (failed) |
| 19. Extended Tridiagonal 1 Function (n = 1000) [2] | 5.9698E-19 (6.29E+01) | 9.7972E+02 (4.51E+00) | 9.9972E+02 (5.61E+02) | 1.0000E+03 (2.66E+02) | (failed) |
| 20. Extended quadratic penalty | 3.9900E+03 (1.15E+01) | 3.9900E+03 (3.30E+00) | 3.9900E+03 (2.58E+04) | 3.9900E+03 (1.05E+02) | (failed) |
| 21. Extended quadratic penalty QP2 Function (n = 1000) [2] | 0 | 0 | 5.1228E-18 (4.75E+01) | 1.0000E+04 (3.66E+00) | (failed) |
| 22. Quadratic QF2 Function (n = 1000) [2] | 0 | 0 | 3.9962E+03 (3.66E+00) | 3.9962E+03 (2.58E+04) | (too long) |
| 23. Extended PSC1 Function (n = 1000) [2] | 3.8660E+02 (6.83E+00) | 3.9930E+02 (3.32E+00) | 3.8660E+02 (2.82E+04) | 3.8660E+02 (9.50E+01) | (failed) |
| 24. Extended BD1 Function (n = 1000) [2] | 3.8660E+02 (6.83E+00) | 3.9930E+02 (3.32E+00) | 3.8660E+02 (2.82E+04) | 3.8660E+02 (9.50E+01) | (failed) |
| 25. Extended Cliff Function (n = 1000) [2] | 9.9939E+01 (7.66E+01) | 2.3779E+02 (5.44E+00) | 9.9939E+01 (1.85E+01) | 1.0179E+02 (2.44E+02) | (failed) |
| 26. Perturbed quadratic diagonal Function (n = 1000) [2] | 3.8660E+02 (6.83E+00) | 3.9930E+02 (3.32E+00) | 3.8660E+02 (2.82E+04) | 3.8660E+02 (9.50E+01) | (failed) |
| 27. Extended Hiebert Function (n = 1000) [2] | 5.4874E-19 (8.55E+01) | 1.2500E+12 (2.21E+01) | 4.6675E-21 (1.91E-01) | 1.1715E+08 (1.22E+02) | (failed) |
| 28. Extended TET Function (n = 1000) [2] | 1.2796E+03 (3.58E+00) | 1.2796E+03 (3.65E+00) | 1.2796E+03 (2.67E+04) | 1.2796E+03 (1.75E+02) | (failed) |
| 29. Diagonal 1 Function (n = 1000) [2] | -2.7068E+06 (1.09E+00) | -2.2288E+06 (3.54E+00) | -2.7068E+06 (5.33E+01) | -2.7068E+06 (2.52E+02) | (failed) |
| 30. Diagonal 3 Function (n = 1000) [2] | -5.0050E+05 (2.04E+01) | -4.8990E+05 (1.13E+01) | -5.0050E+05 (7.34E-02) | -4.8990E+05 (2.17E+02) | (failed) |
| 31. Diagonal 5 Function (n = 1000) [2] | -1.0231E+02 (6.28E-01) | -9.9315E+02 (1.14E+01) | -1.0231E+02 (2.81E+04) | -1.0231E+02 (1.65E+02) | (failed) |
| 32. Extended Maratos Function (n = 1000) [2] | -5.0031E+02 (7.20E+01) | -5.0031E+02 (3.63E+00) | -5.0031E+02 (1.60E+01) | -5.0031E+02 (1.74E+02) | (failed) |
| 33. EG2 Function (n = 1000) [2] | 6.9315E+02 (6.60E+00) | 6.9331E+02 (1.14E+01) | 6.9315E+02 (2.81E+04) | 6.9315E+02 (1.65E+02) | (failed) |
| 34. SINQUAD Function (n = 1000) [2] | 0 | 0 | 0 | 0 | (failed) |
Table 8: Numerical results of small-scale problems (no.35-51) computed by CN-MGE, GlobalSearch, Couenne and CMA-ES.

| Problem               | CNMGE | GlobalSearch | Couenne | CMA-ES | Global minimum |
|-----------------------|-------|--------------|---------|--------|----------------|
|                       | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) |
| 35. Griewank Function (n = 10) [64, 77] | 1.9318E-14 (1.98E+00) | 1.7187E-12 (2.66E+00) | 1.5929E-26 (1.14E+03) | 6.6613E-16 (1.33E+00) | 0 |
| 36. Levy Function N. 13 (n = 2) [77] | 1.3497E-31 (3.87E-01) | 1.0981E-01 (2.85E+00) | 2.8774E-16 (8.48E-03) | 1.3498E-31 (6.78E-01) | 0 |
| 37. Hosaki Function (n = 2) [1] | -inf (2.62E+00) | -1.1277E+00 (1.14E+03) | -2.3458E+00 (2.80E+04) | -2.3458E+00 (5.70E-01) | -2.3458E+00 |
| 38. Beale Function (n = 2) [64, 77] | 4.5343E-14 (2.38E-01) | 0 (3.80E+00) | 5.0180E-23 (2.68E-03) | 1.3347E-17 (6.14E-01) | 0 |
| 39. Easom Function (n = 2) [77] | -1.0000E+00 (1.97E+00) | -1.0000E+00 (2.64E+00) | -3.0308E-05 (2.55E-03) | -3.0308E-05 (3.71E-02) | -1.0000E+00 |
| 40. Price Function (n = 2) [1] | 0 (3.76E-02) | 1.1149E-12 (5.21E+00) | 1.1663E-37 (1.46E-03) | 5.5711E-13 (9.81E-01) | 0 |
| 41. Branin Function (n = 2) [77] | 3.9788E-01 (4.42E-02) | 3.9789E-01 (2.21E+00) | 3.9789E-01 (8.57E-03) | 3.9789E-01 (6.35E-01) | 3.9789E-01 |
| 42. Trecanni Function (n = 2) [1] | 0 (2.66E-02) | 1.8313E-15 (2.43E+00) | 6.0525E-50 (8.27E-01) | -3.5527E-15 (5.41E-01) | 0 |
| 43. Booth Function (n = 2) [77] | 1.3604E-15 (3.85E-02) | 4.1193E-15 (2.45E+00) | 0 (1.05E-03) | 1.8582E-18 (5.49E-01) | 0 |
| 44. Matyas Function (n = 2) [77] | 0 (3.61E-02) | 6.9378E-16 (2.45E+00) | 0 (1.05E-03) | 7.9654E-18 (5.74E-01) | 0 |
| 45. McCormick Function (n = 2) [77] | -1.9132E+00 (5.67E-01) | -1.9132E+00 (2.99E+00) | -1.9132E+00 (7.47E-03) | -1.9132E+00 (5.37E-01) | -1.9132E+00 |
| 46. Power Sum Function (n = 4) [77] | 4.2909E-04 (1.86E-01) | 2.0472E-07 (4.19E+00) | 4.2680E-22 (4.76E-03) | 5.4319E-12 (4.46E+00) | 0 |
| 47. Colville Function (n = 4) [77] | 0 (1.00E-01) | 5.3436E-13 (3.80E+00) | 0 (1.05E-03) | 0 (1.09E+00) | 0 |
| 48. Schaffer Function N. 2 (n = 2) [77] | 0 (2.48E+00) | 0 (4.17E+00) | 0 (1.10E-03) | 2.9258E-01 (1.32E+00) | 0 |
| 49. Bohachevsky Function (n = 2) [77] | 0 (3.93E-01) | 0 (2.43E+00) | -5.551E-17 (2.43E-02) | 0 (6.54E-01) | 0 |
| 50. Three-Hump Camel Function (n = 2) [77] | 0 (9.51E-02) | 3.2640E-16 (2.32E+00) | 1.4758E-18 (2.10E-01) | 7.5064E-18 (5.50E-01) | 0 |
| 51. Six-Hump Camel Function (n = 2) [77] | -1.0316E+00 (9.84E-02) | -1.0316E+00 (2.29E+00) | -1.0316E+00 (9.38E-02) | -1.0316E+00 (6.07E-01) | -1.0316E+00 |
Table 9: Numerical results of small-scale problems (no.52-68) computed by CNMGE, GlobalSearch, Couenne and CMA-ES.

| Problem | CNMGE min f(x) (CPU time(s)) | GlobalSearch min f(x) (CPU time(s)) | Couenne min f(x) (CPU time(s)) | CMA-ES min f(x) (CPU time(s)) | Global minimum |
|---------|-----------------------------|-------------------------------------|-------------------------------|--------------------------------|----------------|
| 52. Drop-Wave Function (n = 2) [77] | -1.0000E+00 (3.88E+00) | -1.0000E+00 (3.12E+00) | -1.0000E+00 (2.59E-02) | -9.8389E-01 (7.34E-01) | -1.0000E+00 |
| 53. Perm Function 0, d, β (n = 4) [77] | 0 (3.82E-01) | 6.3025E+02 (5.79E+00) | 5.3868E-10 (2.10E-01) | 3.0872E-16 (1.11E+00) | 0 |
| 54. Hartmann 3-D Function (n = 3) [77] | -3.8627E+00 (2.89E+00) | -3.8628E+00 (2.77E+00) | -3.8628E+00 (4.50E-01) | -3.8628E+00 (7.34E-01) | -3.8628E+00 |
| 55. Trefethen 4 Function (n = 2) [77] | -3.1407E+00 (8.16E-02) | -2.8376E+00 (5.91E+00) | -3.0699E+00 (7.88E-01) | -3.2965E+00 (8.23E-01) | -3.3069E+00 |
| 56. Zettl Function (n = 2) [3] | -3.7912E-03 (2.82E-01) | -3.7912E-03 (2.36E+00) | -3.7912E-03 (1.80E-02) | -3.7912E-03 (6.59E-01) | -3.7912E-03 |
| 57. Exp2 Function (n = 2) [3] | -1.7654E+02 (1.21E+00) | -1.9299E+01 (3.36E+00) | -1.9299E+01 (3.81E-01) | -1.9299E+01 (6.05E-01) | -1.9299E+01 |
| 58. Hansen Function (n = 2) [77] | -8.6901E-01 (6.77E+00) | -8.6901E-01 (5.49E+00) | -8.6901E-01 (2.31E+03) | -8.6901E-01 (5.67E+01) | -8.6901E-01 |
| 59. Schaffer Function N. 4 (n = 2) [77] | -3.7274E+03 (3.74E-01) | -3.7274E+03 (3.53E+00) | -3.7274E+03 (9.28E+02) | -3.7274E+03 (8.48E+01) | -3.7274E+03 |
| 60. Holder Table Function (n = 2) [77] | -1.3061E+00 (5.12E+00) | -1.8013E+00 (4.32E+00) | -1.8013E+00 (3.62E+01) | -1.8013E+00 (5.97E+01) | -1.8013E+00 |
| 61. Gramacy & Lee (2012) Function (n = 1) [77] | -9.1005E+06 (5.32E+01) | -6.0207E+00 (2.75E+00) | -6.0207E+00 (6.78E+02) | -6.0207E+00 (4.10E+01) | -6.0207E+00 |
| 62. Eggholder Function (n = 2) [3] | -3.0000E+00 (2.13E+00) | -3.0000E+00 (3.88E+00) | -3.0000E+00 (7.46E-04) | -3.0000E+00 (5.95E-01) | -3.0000E+00 |
| 63. Michalewicz Function (n = 2) [77] | -1.0000E+00 (2.13E+00) | -1.0000E+00 (3.88E+00) | -1.0000E+00 (7.46E-04) | -1.0000E+00 (5.95E-01) | -1.0000E+00 |
Table 10: Numerical results of large-scale problems (no.1-17) computed by CNMGE, MCS, GLODS and VRBBO.

| Problem                        | CNMGE (CPU time (s)) | MCS (CPU time (s)) | GLODS (CPU time (s)) | VRBBO (CPU time (s)) | Global minimum |
|-------------------------------|---------------------|-------------------|---------------------|---------------------|-----------------|
| 1. Molecular Energy           | -4.1118E+01 (1.59E+01) | -3.6902E-01 (2.01E+02) | 6.2322E-04 (2.12E+02) | -2.8789E-01 (1.51E+01) | -4.1118E+01     |
| 2. Ackley Function            | 8.8817E-16 (1.60E+02) | 0 (1.28E+04)       | 8.8817E-16 (1.14E+02) | 4.8252E-07 (4.81E+00) | 0               |
| 3. Levy Function              | 1.4997E-32 (3.67E+02) | 2.2419E-09 (1.10E+02) | 8.9970E+01 (1.48E+02) | 7.3494E-12 (1.43E+01) | 1.2728E-02      |
| 4. Schwefel Function          | 1.2728E-02 (6.83E+01) | 1.1844E+05 (2.60E+02) | 2.9976E-01 (1.35E+02) | 2.1713E+05 (1.36E+01) | 0               |
| 5. Rastrigin Function         | 0 (4.33E+00)        | 0 (1.25E+02)       | 0 (1.48E+02)         | 2.8867E-09 (4.26E+00) | 0               |
| 6. Styblinski-Tang Function   | 3.9166E+04 (5.78E+01) | -3.9166E+04 (3.17E+02) | 9.4000E+02 (1.27E+02) | -2.5029E+04 (3.82E+01) | -3.9166E+04     |
| 7. Trid Function              | -1.6717E+08 (2.84E+01) | -9.9990E+02 (1.84E+02) | -9.6100E+02 (1.53E+02) | -2.3258E+07 (3.81E+00) | -1.6716E+08     |
| 8. Sum Squares Function       | 0 (1.00E+01)        | 0 (1.26E+04)       | 0 (1.13E+02)         | 5.9371E-09 (3.59E+00) | 0               |
| 9. Sphere Function            | 0 (2.23E-01)        | 0 (1.26E+04)       | 0 (1.13E+02)         | 1.4551E-09 (3.27E+00) | 0               |
| 10. Rotated Hyper-Ellipsoid   | 0 (1.34E+01)        | 0 (1.43E+04)       | 0 (1.30E+02)         | 8.6292E-09 (1.71E+02) | 0               |
| 11. Zakharov Function         | 0 (6.00E+00)        | 1.7016E+04 (1.44E+02) | 0 (1.13E+02)         | 8.4247E-03 (3.60E+00) | 0               |
| 12. Dixon-Price Function      | 6.2737E-14 (4.96E+01) | 6.6667E-01 (8.45E+02) | 6.6796E-01 (1.19E+02) | 9.4552E+03 (9.98E-01) | 0               |
| 13. Rosenbrock Function       | 0 (5.17E+01)        | 2.1794E+02 (2.18E+02) | 9.9815E+02 (1.30E+02) | 7.2905E-09 (4.16E+00) | 0               |
| 14. Powell Function           | 0 (2.03E+02)        | 2.1627E+01 (2.05E+02) | 0 (1.22E-02)         | 2.1787E-04 (1.95E+01) | 0               |
| 15. Quartic With Noise Function| 5.0000E-01 (5.53E+00) | 5.0000E-01 (1.27E+04) | 5.0000E-01 (1.14E+02) | 5.0000E-01 (5.77E+00) | 0               |
| 16. Schubert Function         | 1.2031E+04 (1.87E+02) | 2.6399E+03 (3.34E+02) | 1.2031E+04 (8.91E+01) | -1.0629E+04 (3.93E+01) | -1.2031E+04     |
| 17. Raydan I Function         | 5.0000E+04 (7.35E-01) | 5.0050E+04 (1.22E+04) | 5.0000E+04 (8.91E+01) | 5.0050E+04 (3.93E+01) | 0               |
Table 11: Numerical results of large-scale problems (no.18-34) computed by CNMGE, MCS, GLODS and VRBBO.

| Problem | CNMGE | MCS | GLODS | VRBBO |
|---------|-------|-----|-------|-------|
|         | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) | min $f(x)$ (CPU time (s)) | Global minimum |
| 18. Raydan 2 Function | 1.0000E+03 (2.15E+00) | 1.0000E+03 (1.31E+04) | 1.0000E+03 (5.16E+01) | 1.3038E+03 (5.08E+00) | \ |
| (n = 1000) | | | | | |
| 19. Extended Tridiagonal 1 Function | 5.9698E-19 (6.29E+01) | 4.036E-01 (1.59E+02) | 9.8000E+02 (1.47E+02) | 9.0949E-13 (4.76E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 20. Extended quadratic penalty QP1 Function | 3.9900E+03 (1.15E+01) | 3.9912E+03 (7.88+01) | 3.9991E+03 (1.30E+02) | 3.9900E+03 (3.36E+00) | \ |
| (n = 1000) | | | | | |
| 21. Extended quadratic penalty QP2 Function | 5.1228E-18 (4.75E+01) | 6.6117E+02 (9.7E+01) | 3.74E+02 (1.49E+02) | 3.64E-02 (3.69E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 22. Quadratic QF2 Function | -1.0001E+00 (3.92E+02) | -1.0001E+00 (1.27E+02) | 2.5022E+05 (3.69E+00) | -1.0001E+00 (7.51E+00) | \ |
| (n = 1000) | | | | | |
| 23. Extended PSC1 Function | 3.8660E+02 (6.33E+00) | 4.0143E+02 (9.22E+01) | 4.9849E+02 (4.50E+00) | 3.8660E+02 (5.08E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 24. Extended BD1 Function | 0 (3.99E+01) | 1.276E-23 (9.52E+01) | 2.0637E+03 (6.67E+00) | 7.2759E-15 (4.05E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 25. Extended Cliff Function | 9.8933E+01 (7.66E+01) | 2.9165E+02 (7.05E+01) | 4.9595E+02 (7.88E+00) | 9.9906E+01 (4.05E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 26. Perturbed quadratic diagonal Function | 0 (3.92E+02) | 0 (1.27E+04) | 0 (1.36E+02) | 1.3973E+03 (3.38E+00) | \ |
| (n = 1000) | (too long) | (failed) | (failed) | (failed) | |
| 27. Extended Hubert Function | 5.4874E-19 (8.85E+01) | 6.5817E+06 (1.41E+02) | 1.2475E+12 (1.81E+02) | 2.5105E+08 (3.34E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 28. Extended TET Function | 1.2796E+03 (3.58E+01) | 1.2796E+03 (2.17E+02) | \ (0) | 1.2796E+03 (7.51E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 29. Diagonal 1 Function | -2.7068E+06 (1.09E+00) | -2.6997E+06 (7.52E+01) | 7.3936E+02 (7.52E+01) | -2.7068E+06 (2.52E+02) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 30. Diagonal 3 Function | -5.0050E+05 (2.04E+01) | -4.9623E+05 (2.79E+02) | -5.0049E+05 (5.93E+01) | -4.9573E+05 (1.18E+01) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 31. Diagonal 5 Function | 6.9315E+02 (6.20E-01) | 6.9315E+02 (1.25E+04) | \ (0) | 6.9315E+02 (7.98E+00) | \ |
| (n = 1000) | (too long) | (failed) | (failed) | (failed) | |
| 32. Extended Maratos Function | -5.0031E+02 (7.20E-01) | -5.0026E+02 (8.24E+01) | 4.9596E+04 (1.45E+02) | -4.9736E+02 (3.43E+00) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 33. EG2 Function | -9.9950E+02 (1.14E+02) | -9.9917E+02 (1.01E+02) | -9.9946E+02 (1.29E+02) | -9.9949E+02 (1.50E+01) | \ |
| (n = 1000) | (failed) | (failed) | (failed) | (failed) | |
| 34. SINQUAD Function | 0 (3.57E+01) | 5.5349E-01 (1.70E+02) | 7.5168E-01 (1.30E+02) | 8.1854E-12 (3.98E+00) | \ |
Table 12: Numerical results of small-scale problems (no.35-51) computed by CNMGE, MCS, GLODS and VRBBO.

| Problem                          | CNMGE | MCS  | GLODS | VRBBO |
|----------------------------------|-------|------|-------|-------|
|                                  | $\min f(x)$ (CPU time (s)) | $\min f(x)$ (CPU time (s)) | $\min f(x)$ (CPU time (s)) | $\min f(x)$ (CPU time (s)) | Global minimum |
| 35. Griewank Function            | 1.9318E-14 (1.98E+00) | 0 (6.38E-01) | 0 (9.01E+00) | 3.2362E-09 (2.63E-01) | 0 |
| (n = 10) [64, 77]                |       |      |       |       |               |
| 36. Levy Function N. 13          | 1.3497E-31 (3.87E-01) | 1.3498E-31 (7.01E-01) | 1.3497E-31 (5.80E+01) | 5.4136E-06 (3.74E+00) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 37. Hosaki Function              | -inf (1.40E+00) | -2.3457E+00 (3.26E-01) | -2.3458E+00 (4.89E-01) | -1.0000E+50 (2.36E-02) | -2.3458E+00 |
| (n = 2) [3]                      |       |      |       |       |               |
| 38. Beale Function               | 4.5343E-14 (2.38E+01) | 1.5473E-22 (2.82E-02) | 2.0990E-15 (5.82E+02) | 4.8504E-07 (4.92E-01) | 0 |
| (n = 2) [64, 77]                 |       |      |       |       |               |
| 39. Easom Function               | -1.0000E+00 (1.97E+00) | -1.0000E+00 (6.13E-01) | -8.1102E-05 (5.23E+02) | -9.9991E-01 (3.71E+02) | -1.0000E+00 |
| (n = 2) [77]                     |       |      |       |       |               |
| 40. Price Function               | 3.9788E-01 (4.42E-02) | 3.9789E-01 (4.91E-01) | 3.9789E-01 (5.67E+01) | 3.9789E-01 (2.00E+02) | 3.9789E-01 |
| (n = 2) [3]                      |       |      |       |       |               |
| 41. Branin Function              | 3.6034E-15 (3.85E-02) | 0 (2.48E-01) | 4.4408E-16 (5.25E+01) | 9.5258E-06 (1.85E-01) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 42. Trecanni Function            | 0 (2.66E-02) | 0 (4.46E-01) | 0 (5.36E+01) | 2.9796E-07 (1.79E-02) | 0 |
| (n = 2) [3]                      |       |      |       |       |               |
| 43. Booth Function               | 1.3604E-15 (3.61E-02) | 0 (9.18E-02) | 4.4008E-16 (5.23E+01) | 4.0140E-10 (3.77E-02) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 44. Matyas Function              | 0 (3.61E-02) | 0 (9.18E-02) | 0 (5.23E+01) | 4.0140E-10 (3.77E-02) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 45. McCormick Function           | -1.9132E+00 (5.67E-01) | -1.9132E+00 (2.30E-01) | -1.9132E+00 (7.99E-01) | 1.2283E+00 (1.87E-02) | -1.9132E+00 |
| (n = 2) [77]                     |       |      |       |       |               |
| 46. Power Sum Function           | 4.2909E-04 (1.86E-01) | 1.4325E-06 (1.07E+00) | 0 (2.24E+01) | 2.4072E-09 (3.31E-02) | 0 |
| (n = 4) [77]                     |       |      |       |       |               |
| 47. Colville Function            | 0 (1.00E+01) | 3.8343E-19 (5.95E-01) | 1.4701E-05 (2.00E+01) | 1.4566E-09 (2.99E-02) | 0 |
| (n = 4) [77]                     |       |      |       |       |               |
| 48. Schaffer Function N. 2       | 0 (5.67E-01) | 0 (6.62E-01) | 0 (5.37E+01) | 5.0000E-01 (2.03E-02) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 49. Bohachevsky Function         | 0 (3.93E-01) | 0 (1.66E-01) | -5.5511E-17 (5.39E+01) | 2.8554E-06 (2.13E-02) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 50. Three-Hump Camel Function    | 0 (9.51E-02) | 0 (1.15E-01) | 0 (5.37E+01) | 2.3841E-07 (2.26E-02) | 0 |
| (n = 2) [77]                     |       |      |       |       |               |
| 51. Six-Hump Camel Function      | -1.0316E+00 (9.84E-02) | -1.0316E+00 (3.74E-01) | -1.0316E+00 (5.51E+01) | -1.0316E+00 (2.78E-02) | -1.0316E+00 |
Table 13: Numerical results of small-scale problems (no.52-68) computed by CN-MGE, MCS, GLODS and VRBBO.

| Problem | CNMGE | MCS | GLODS | VRBBO | Global Minimum |
|---------|-------|-----|-------|-------|----------------|
| Problem | min \( f(x) \) | min \( f(x) \) | min \( f(x) \) | min \( f(x) \) | CPU time (s) | CPU time (s) | CPU time (s) | CPU time (s) | CPU time (s) |
| 52. Drop-Wave Function (n = 2) [77] | -1.0000E+00 | -1.0000E+00 | -1.0000E+00 | -7.8506E-01 | (1.74E-01) | (5.32E-02) | (5.65E-01) | (3.88E+00) | -1.0000E+00 |
| 53. Perm Function 0, \( d, \beta \) (n = 4) [77] | 0 | 1.0130E+02 | 7.3526E+02 | 0 | (5.32E+01) | (2.16E+01) | (9.28E-01) | (2.68E+00) | (3.82E-01) |
| 54. Hartmann 3-D Function (n = 3) [77] | -3.8627E+00 | -3.8628E+00 | -3.8628E+00 | -3.8628E+00 | (3.23E+01) | (8.20E-01) | (3.89E+00) | (2.89E+00) | -3.8628E+00 |
| 55. Trefethen 4 Function (n = 2) [3] | -3.1407E+00 | -2.7612E+00 | 7.3215E+02 | -3.0897E+00 | (1.82E-01) | (5.38E-01) | (5.65E-01) | (2.89E+00) | -3.0626E+00 |
| 56. Zettl Function (n = 2) [3] | -3.7912E-03 | -3.7912E-03 | -3.7912E-03 | -3.7912E-03 | (1.84E-01) | (5.99E-02) | (1.84E-01) | (5.99E-02) | -3.7912E-03 |
| 57. Exp2 Function (n = 2) [3] | -3.7912E-03 | -3.7912E-03 | -3.7912E-03 | -3.7912E-03 | (5.39E+02) | (2.57E-01) | (5.39E+02) | (2.57E-01) | -3.7912E-03 |
| 58. Hansen Function (n = 2) [3] | -1.7654E+02 | 1.4548E+02 | -1.16E+02 | -2.43E+01 | (2.89E+00) | (7.22E-01) | (2.15E+00) | (7.22E-01) | -1.7654E+02 |
| 59. Schaffer Function N. 4 (n = 2) [77] | 2.9257E-01 | 4.9289E-01 | 5.0000E-01 | 5.0000E-01 | (3.14E+00) | (2.44E-01) | (2.44E-01) | (2.44E-01) | 2.9258E-01 |
| 60. Holland Table Function (n = 2) [77] | inf | -1.9209E+01 | -1.9209E+01 | -1.9209E+01 | (3.00E+00) | (5.86E-01) | (1.92E+00) | (1.92E+00) | -1.9209E+01 |
| 61. Gramacy & Lee (2012) Function (n = 1) [77] | -8.6901E-01 | -6.6328E-01 | -6.6327E-01 | -5.2660E-01 | (6.77E-01) | (4.56E-01) | (4.27E-01) | (4.27E-01) | -8.6901E-01 |
| 62. Eggholder Function (n = 2) [3] | -3.7274E+03 | -5.5737E+03 | -1.2893E+32 | -2.4663E+37 | (3.74E-01) | (4.70E-01) | (5.26E+01) | (5.26E+01) | -9.5964E+02 |
| 63. Michalewicz Function (n = 2) [77] | -1.3061E+00 | -1.8013E+00 | -1.8013E+00 | -1.8013E+00 | (5.12E-01) | (3.11E-01) | (5.29E+01) | (5.29E+01) | -1.8013E+00 |
| 64. Box-Botts Exponential Quadratic Function (n = 3) [3] | 0 | 0 | 0 | 0 | (7.09E-02) | (1.59E-01) | (3.28E+00) | (3.28E+00) | 1.9491E-07 |
| 65. Cross-in-Tray Function (n = 2) [77] | -2.0626E+00 | -1.8899E+00 | -2.0626E+00 | -2.0626E+00 | (2.13E+00) | (3.40E-01) | (1.32E+00) | (1.32E+00) | -2.0626E+00 |
| 66. Himmelblau Function (n = 2) [77] | 6.9200E-16 | 3.4597E-20 | 0 | 5.6383E-04 | (6.80E-02) | (4.09E-00) | (5.43E-01) | (5.43E-01) | 0 |
| 67. Forrestier et al. (2008) Function (n = 1) [77] | -9.1005E+06 | -6.0207E+00 | -6.0207E+00 | -6.0207E+00 | (5.32E-01) | (2.65E-01) | (3.00E-01) | (3.00E-01) | -8.4445E+03 |
| 68. Goldstein-Price Function (n = 2) [77] | 3.0000E+00 | 3.0000E+00 | 3.0000E+00 | 3.0000E+00 | (4.73E-01) | (7.32E-01) | (1.86E+00) | (1.86E+00) | 3.0000E+00 |
Table 14: The number of failed problems computed by CNMGE, GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO.

|                      | CNMGE | GlobalSearch | Couenne | CMA-ES |
|----------------------|-------|--------------|---------|--------|
| The number of failed problems | 0     | 21           | 3       | 19     |
| The number of problems computed by 8 hours | \    | \            | 9       | \      |
| The probability of failure     | 0  $\frac{21}{88}$ (30.88%) | $\frac{9}{88}$ (17.64%) | $\frac{19}{88}$ (27.94%) |

|                      | MCS   | GLODS | VRBBO |
|----------------------|-------|-------|-------|
| The number of failed problems | 19    | 23    | 16    |
| The probability of failure     | $\frac{19}{88}$ (27.94%) | $\frac{23}{88}$ (33.82%) | $\frac{16}{88}$ (23.52%) |

7 Conclusions

Based on the deflation technique and the genetic evolution, we give an efficient continuation Newton for solving the global minimum of an unconstrained optimization problem. We implement it in the MATLAB2021b environment and denote this solver as CNMGE. In order to retain the usability of derivative-free methods and the fast convergence of the gradient-based methods, we use the automatic differentiation technique with the reverse mode to compute the gradient of the objective function and replace the Hessian matrix with its finite difference approximation for CNMGE. The numerical results show that CNMGE works well for global optimization problems, especially some large-scale problems of which are difficult to be solved by the known global optimization methods. Therefore, CNMGE can be regarded as an alternative workhorse for unconstrained optimization problems.
Acknowledgments

The authors are grateful to Prof. Jonathan Eckstein for the suggestion of the comparison to Couenne and Prof. Nick Sahinidis for the suggestion of the comparison to the derivative-free optimization methods such as MCS and CMA-ES. The first author also thanks to Prof. Ya-xiang Yuan (his supervisor) encouraging him to pursue the ODE-based method for optimization problems. The authors thanks to three anonymous referees for their comments and suggestions which greatly improve presentation of this paper.

Declarations

Funding: This work was supported in part by Grant 61876199 from National Natural Science Foundation of China, and Grant YICB2011003HI from the Innovation Research Program of Huawei Technologies Co., Ltd..

Conflicts of interest/Competing interests: Not applicable.

Availability of data and material (data transparency): If it is requested, we will provide the test data.

Code availability (software application or custom code): https://teacher.bupt.edu.cn/luoxinlong/zh_CN/zzcg/41406/list/index.htm.

References

1. Abbott, J. P.: Numerical continuation methods for nonlinear equations and bifurcation problems, Ph.D. Thesis, Computer Center, Australian National University, 1977.
2. Andrei, N.: An unconstrained optimization test functions collection, Advanced Modeling and Optimization 10 (2008), 147-161.
3. Adorio, E. P., Diliman, U. P.: MVF-Multivariate test functions library in C for unconstrained global optimization, available at http://www.geocities.ws/eadorio/mvf.pdf, 2005.
4. Andricioaei, I., Straub, J.E.: Global optimization using bad derivatives: derivative-free method for molecular energy minimization, J. Comput. Chem. 19 (1998), 1445-1455.
5. Allgower, E. L., Georg, K.: Introduction to Numerical Continuation Methods, SIAM, Philadelphia, PA, 2003.
6. Ascher, U. M., Petzold, L. R.: Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, SIAM, Philadelphia, PA, 1998.
7. Axelsson, O., Sysala, S.: Continuation Newton methods, Comput. Math. Appl. 70 (2015), 2621-2637.
8. Averick, B. M., Carter, R. G., Moré, J. J., Xue, G. L.: The MINPACK-2 Test Problem Collection, Mathematics and Computer Science Division, Argonne National Laboratory, Preprint MCS-P153-0692, 1992.
9. Baydin, A. G., Pearlmutter, B. A., Radul, A. A. and Siskind, J. M.: Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2017), 5595-5637.
10. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP, Optim. Methods Softw. 24 (2009), 597-634.
11. Boender, C. G. E.: Bayesian stopping rules for multistart global optimization methods, Math. Program. 37 (1987), 59-80.
12. Branin, F. H.: Widely convergent method for finding multiple solutions of simultaneous nonlinear equations, IBM J. Res. Dev. 16 (1972), 504-521.
13. Brown, K. M., Gearhart, W. B.: Deflation techniques for the calculation of further solutions of a nonlinear system, Numer. Math. 16 (1971), 334-342.

14. Braden, A.: Optimization techniques for solving design problems in modern trombones, In Forum Acusticum 2005, 557-662.

15. Conn, A. R., Gould, N., Toint, Ph. L.: Trust-Region Methods, SIAM, Philadelphia, PA, 2000.

16. Conn, A. R., Scheinberg, K., Vicente, L. N.: Introduction Derivative-free Optimization, SIAM, Philadelphia, PA, 2009.

17. Couenne, a solver for non-convex MINLP problems, available at https://www.coin-or.org/Couenne/, February 2020.

18. CMA-ES, The covariance matrix adaptation evolution strategy, available at http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaes.m, 2012.

19. Czyzyk, J., Mesnier, M. P., Moré, J. J.: The NEOS Server, IEEE Comput. Sci. Eng. 5 (1998), 68-75.

20. Custódio, Madeira, J. F. A.: GLODS: Global and local optimization using direct search, J. Glob. Optim. 62 (2015), 1-28. https://doi.org/10.1007/s10898-014-0224-9.

21. Davidenko, D. F.: On a new method of numerical solution of systems of nonlinear equations (in Russian), Dokl. Akad. Nauk SSSR 88 (1953), 601-602.

22. Deuflhard, P.: Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms, Springer-Verlag, Berlin, 2004.

23. Dolan, E.: The NEOS Server 4.0 Administrative Guide, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

24. Deuflhard, P., Pesch, H. J., Rentrop, P.: A modified continuation method for the numerical solution of nonlinear two-point boundary value problems by shooting techniques, Numer. Math. 26 (1975), 327-343.

25. Dennis, J. E., Schnabel, R. B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations, SIAM, Philadelphia, PA, 1996.

26. Dong, H. C., Song, B. W., Dong, Z. M., Wang, P.: Multi-start space reduction (MSSR) surrogate-based global optimization method, Struct. Multidisc. Optim. 54 (2016), 907-926.

27. Elhara, O., Varelas, K., Nguyen, D., Tusar, T., Brockhoff, D., Hansen, N., Auger, A.: COCO: The large scale black-box optimization benchmarking (bbob-largescale) Test Suite, arXiv preprint available at https://arxiv.org/abs/1903.06396, 2019.

28. Gao, W., Mi, C.: Hybrid vehicle design using global optimisation algorithms, Int. J. Electric Hybrid Veh. 1 (2007), 57-70.

29. Gropp, W., Moré, J. J.: Optimization Environments and the NEOS Server, Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., Cambridge University Press, 1997.

30. Golub, G. H., Van Loan, C. F.: Matrix Computation (4th ed.), The John Hopkins University Press, Baltimore, 2013.

31. Griewank, A. and Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, Philadelphia, 2008. https://doi.org/10.1137/1.9780898717761.

32. Hansen, N.: The CMA evolution strategy, a comparing review, In: Lozano, J. A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer, Berlin, 75-102 (2006).

33. Hansen, N.: The CMA Evolution Strategy: A Tutorial, available at https://arxiv.org/abs/1604.00772, 2010.

34. Hansen, C. H., Simpson, M. T., Cazzolato, B. S.: Active sound and vibration control: theory and applications, chapter 9: Genetic algorithms for optimising ASVC systems, pp. 185–220, No. 62 in IEE control engineering series, London, UK, 2002.

35. Hart, W. E.: Adaptive Global Optimization with Local Search, Ph.D. dissertation, University of California, San Diego, CA, USA, 1994.

36. Higham, D. J.: Trust region algorithms and timestep selection, SIAM J. Numer. Anal. 37 (1999), 194-210.

37. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search, J. Glob. Optim. 14, 331-355 (1999).

38. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems, 2nd ed., Springer-Verlag, Berlin, 1996.

39. Jackiewicz, Z.: General Linear Methods for Ordinary Differential Equations, John Wiley & Sons, Inc., Hoboken, New Jersey, 2009.

40. Kearfott, R. B.: Rigorous Global Search: Continuous Problems, Nonconvex Optimization and Applications, Kluwer Academic, Dordrecht, 1996.
68. Neumaier, A.: MCS: Global Optimization by Multilevel Coordinate Search, https://www.mat.univie.ac.at/~neum/software/mcs/. 2000.
69. NEOS Server, https://neos-server.org/neos/, 2021.
70. Nocedal, J., Wright, S. J.: Numerical Optimization, Springer-Verlag, Berlin, 1999.
71. Ortega, J. M., Rheinboldt, W. C.: Iteration Solution of Nonlinear Equations in Several Variables, SIAM, Philadelphia, PA, 2000.
72. Regis, R. G., Shoemaker, C. A.: A quasi-multistart framework for global optimization of expensive functions using response surface models, J. Glob. Optim. 56 (2013), 1719-1753.
73. Rios, L. M., Sahinidis, N. V.: Derivative-free optimization: a review of algorithms and comparison of software implementations, J. Glob. Optim. 56 (2013), 1247-1293.
74. Rosenbrock, H. H.: An automatic method for finding the greatest or least value of a function, Comput. J. 3 (1960), 175-184. Available online at http://comjnl.oxfordjournals.org/content/3/3/175.full.pdf.
75. Shampine, L. F., Gladwell, I., Thompson, S.: Solving ODEs with MATLAB, Cambridge University Press, Cambridge, 2003.
76. Sahinidis, N. V.: BARON 21.1.13: Global optimization of mixed-integer nonlinear programs, user’s manual, available at https://minlp.com/downloads/docs/baronmanual.pdf, 2021.
77. Surjanovic, S., Bingham, D.: Virtual library of simulation experiments: test functions and datasets, available at http://www.sfu.ca/~ssurjano, January 2020.
78. Sun, J., Garibaldi, J. M., Krasnogor, N., Zhang, Q.: An intelligent multi-restart memetic algorithm for box constrained global optimisation, Evol. Comput. 21 (2014), 107-147.
79. Sergeyev, Y. D., Kvasov, D. E., Mukhametzhanov, M. S.: On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget, Sci. Rep. 8 (2018), 1-9.
80. Sergeyev, Y. D., Kvasov, D. E.: Deterministic Global Optimization: An Introduction to the Diagonal Approach, Springer, 2017.
81. Sun, W.-Y., Yuan, Y.-X.: Optimization Theory and Methods: Nonlinear Programming, Springer, Berlin, 2006.
82. Tanabe, K.: Continuous Newton-Baphson method for solving an underdetermined system of nonlinear equations, Nonlinear Anal. 3 (1979), 495-503.
83. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization, Math. Program. 103 (2005), 225-249.
84. Teughels, A., Roeck, G. D., Suykens, J. A. K.: Global optimization by coupled local minimizers and its application to FE model updating, Comput. Struct. 81 (2003), 2337-2351.
85. Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., Marti, R.: Scatter search and local NLP solvers: A multistart framework for global optimization, INFORMS J. Comput. 19 (2007), 328-340.
86. Willkomm, J. and Vehreschild, A.: The ADiMat handbook, 2013. http://adimat.sc.informatik.tu-darmstadt.de/doc/.
87. Willkomm, J., Bischof, C. H., Bucker, H. M.: A new user interface for ADiMat: toward accurate and efficient derivatives of MATLAB programmes with ease of use, Int. J. Comput. Sci. Eng. 9 (2014), 408-415.
88. Xu, J., Nannariello, J., Fricke, F. R.: Optimising flat-walled multi-layered anechoic linings using evolutionary algorithms, Appl. Acoust. 65 (2004), 1009-1026.
89. Yuan, Y.-X.: Trust region algorithms for nonlinear equations, Information 1 (1998), 7-20.
90. Yuan, Y.-X.: Recent advances in trust region algorithms, Math. Program. 151 (2015), 249-281.