Journal of XXXX manuscript No.
(will be inserted by the editor)

Continuation Newton methods with deflation techniques 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 found, 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

Xin-long Luo, Corresponding author
School of Artificial Intelligence, Beijing University of Posts and Telecommunications, P. O. Box 101, Xitucheng Road No. 10, Haidian District, 100876, Beijing China, E-mail: luoxinlong@bupt.edu.cn

Hang Xiao
School of Artificial Intelligence, Beijing University of Posts and Telecommunications, P. O. Box 101, Xitucheng Road No. 10, Haidian District, 100876, Beijing China, E-mail: xiaohang0210@bupt.edu.cn

Sen Zhang
School of Artificial Intelligence, Beijing University of Posts and Telecommunications, P. O. Box 101, Xitucheng Road No. 10, Haidian District, 100876, Beijing China, E-mail: senzhang@bupt.edu.cn

1 Introduction

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

$$\min_{x\in\Re^n} f(x), \tag{1}$$

where $f:\ \Re^n \to \Re$ is a differentiable function except for a finite number of 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 method [11,22,29,47,78,93], the branch-and-bound method [10,19,82,91], the genetic evolution algorithm [51,65,69] and their memetic algorithms [39,66,84,92].

For the multi-start method, we use the local optimization method such as the trust-region method [17,28,98] or the line search method [76,89] to solve problem (1) and find its local minimum point x^*, which satisfies the following first-order optimality condition

$$g(x^*) = \nabla f(x^*) = 0. \tag{2}$$

Since the implementation of the multi-start method is simple and successful for many real-world problems such as the design of hybrid electric vehicles [31], molecular conformal optimization problems [4,8,50] and acoustic problems [72], many commercial global solvers are based on the multi-start method such as GlobalSearch.m of MATLAB R2021b [67]. 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 [78]. Hence, the multi-start method may find the same local minimum even if those searches are initiated from points that are far apart. The other multi-start method based on the deterministic procedure may have too many search branches (the number of search regions is $2^{n\times l}$ at the l-th level iteration, where n is the dimension of the problem) [22,41,74]. Hence, the computational complexity is very high when the dimension of the problem is large for the deterministic multi-start method.

The evolutionary algorithm [71,85] is another popular method due to its simplicity and attractive nature-inspired interpretations, and is used by a broad community of engineers and practitioners to solve the real-world problem such as finding the optimal location for the active control actuator [38], the optimization of acoustic absorber configuration [96], and the musical instrument [16]. However, for the evolutionary algorithm, since there is not a guaranteed convergence and its computational time is very large for the large-scale problem, it is difficult to find the global minimum accurately.

In order to reduce the computational complexity of the multi-start method and retain the advantage of the evolutionary algorithm escaping from the local minima, we use the continuation Newton method [5,58,61] with the revised deflation technique [15] to find the multiple stationary points of the objective function as the initial seeds of the quasi-genetic algorithm, which are different from the random seeds of

the known evolutionary algorithms. The practical multi-start method and the evolutionary algorithm are derivative-free methods. Namely, the gradient of the objective function does not need to be provided by the user and the derivative-free method [18, 79] is regarded as the black-box method [86].

In order to retain the usability of the derivative-free method and the fast convergence of the gradient-based method, we adopt the automatic differentiation technique [9,34,73,94,95] 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 the automatic differentiation with the reverse mode is about a constant multiple of the function cost. 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, [34]). Therefore, the computational complexity of the gradient evaluated by the automatic differentiation with the reverse mode is far less than that of the gradient estimated 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 the eigenvalue problems [52,56], linearly constrained optimization problems [57,62], the system of nonlinear equations [55,58,61], unconstrained optimization problems [53,54, 60], the linear programming problem [59], the linear complementarity problem [63] and the nonlinear equality-constrained optimization problems [64]. These continuation methods are only applicable to find a solution of nonlinear equations. In this paper, we need to confront multiple solutions of nonlinear equations. Namely, in order to obtain the global minimum of the objective function, we need to find its multiple stationary points. Therefore, we revise the deflation technique and combine it with the continuation method 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 with the trust-region updating strategy to follow the continuation Newton flow and find a stationary point of the objective function. In Section 3, we revise the deflation technique 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 found 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}^* by using the smallest L points $x_1^*, \ldots, x_L^*$ of all found stationary points $x_1^*, \ldots, x_{\mathbf{nsp}}^*$ as the evolutionary seeds. After we obtain the suboptimal point x_{ag}^* of the quasi-genetic algorithm, we refine it by using it as the initial point of the continuation Newton method to solve the system (2) of nonlinear equations and obtain a better approximation of the global minimum point. In Section 5, we give the descriptions of its implementation diagram and an example to illustrate how to use this solver. In Section 6, some promising numerical results of the proposed algorithm are reported, with comparison to other representative global optimization methods such as the multi-start method (the subroutine GlobalSearch.m of MATLAB R2021b, GLODS: Global and Local Optimization using Direct Search [22], and VRBBO: Vienna Randomized Black Box Optimization

[47]), the branch-and-bound method (a state-of-the-art open-source solver (Couenne) [10,19]), and the derivative-free algorithm (the covariance matrix adaptation evolution strategy (CMA-ES) [36,37] and the multilevel coordinate search (MCS) [41, 74]). Finally, some discussions and conclusions are also given in Section 7. $\|\cdot\|$ 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 [58,61], 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 [45,46,77] for finding a root of nonlinear equations (2) as follows:

$$x_{k+1} = x_k - \alpha_k H(x_k)^{-1} g(x_k), \tag{3}$$

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:

$$\frac{dx(t)}{dt} = -H(x)^{-1} g(x), \quad x(t_0) = x_0. \tag{4}$$

Actually, if we apply an iteration with the explicit Euler method [6,81] 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:

$$H(x)\frac{dx(t)}{dt} = -g(x), \quad x(t_0) = x_0. \tag{5}$$

The continuous Newton flow (5) is an old method and can be backtracked to Davidenko's work [23] in 1953. After that, it was investigated by Branin [12], Deuflhard et al [27], Tanabe [90] 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 1.

Proposition 1 *(Branin [12] and Tanabe [90]) Assume that $x(t)$ is a solution of the continuous Newton flow (5). Then, the energy function $E(x(t)) = \|g(x)\|^2$ 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 [12,90]. The proof of the second part can be found in references [58,61]. □

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 1 for convenience.

Definition 1 (The definition of the stiff ODE [42,49]) An ordinary differential equation $dx/dt = F(x)$ is called a stiff ODE, if $\mathrm{Re}(\lambda_i(x)) \leq 0$ $(i = 1, 2, \ldots, n)$ and the ratio $r(x) = \max_{i=1,\ldots,n} |\mathrm{Re}(\lambda_i(x))| / \min_{i=1,\ldots,n} |\mathrm{Re}(\lambda_i(x))|$ is very large, where $\mathrm{Re}(\lambda(x))$ represents the real part of $\lambda(x)$ and $\lambda_i(x)$ $(i = 1, \ldots, n)$ are the eigenvalues of the Jacobian $F'(x)$.

Remark 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 solution element $x^i(t)$ $(i = 1, 2, \ldots, n)$ of the continuous Newton flow (4) has the roughly same convergence rate, and it mitigates the stiffness of 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 ODE (5), we have

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

Thus, we obtain

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

From Definition 1, we know that ODE (6) is a non-stiff ODE. By integrating the linear ODE (6), we obtain

$$x(t) = e^{-t}x_0. \tag{7}$$

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, ODE (5) is the system of differential-algebraic equations (DAEs), and its trajectory can not be efficiently followed by the general ODE methods [6,42,43] such as the backward differentiation formulas (the built-in subroutine ode15s.m of MATLAB R2021b). 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 method, and the fast convergence rate near the stationary point x^* as the merit-function-based method. In order to achieve the two aims, we construct the special continuation Newton method with the new step $\alpha_k = \Delta t_k/(1 + \Delta t_k)$ [58,59,60,

61,62,63] and the time step Δt_k is adaptively adjusted by the trust-region updating strategy [97,98] to follow the trajectory of ODE (5).

Firstly, we apply the implicit Euler method [81] to ODE (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:

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

where H_k equals $H(x_k)$ or its approximation. For the symmetric indefinite system (9), it can be solved by the method of Aasen, which involves $n^3/3$ flops (see pp. 186-190 in [33] or reference [14]).

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 an ODE method, they are different. The damped Newton method (3) is obtained by the explicit Euler method applied to ODE (5), and its time step α_k is restricted by the numerical stability [42]. That is to say, for the linear test function $g(x) = Ax$, from equation (6), we know that its time step α_k is restricted by the stable region $|1-\alpha_k| \le 1$. Therefore, the large time step α_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 ODE (5), and its time step Δt_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 Δt_k such that the continuation Newton method (9) accurately follows the trajectory of ODE 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 Δt_k based on the trust-region updating strategies [24,40,52,54,55,56] can be described as follows. The time step Δt_{k+1} will be enlarged when the linear model $g(x_k)+H_k s_k$ approximates $g(x_k+s_k)$ well, and Δt_{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 of equation (9), we enlarge or reduce the time step Δt_{k+1} at every iteration according to the following ratio:

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

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

$$\Delta t_{k+1} = \begin{cases} c_2 \Delta t_k, & \text{if } 0 \le |1-\rho_k| \le \eta_1, \\ c_1 \Delta t_k, & \text{if } |1-\rho_k| \ge \eta_2 \text{ and } \Delta t_k \ge \overline{\Delta t}, \\ \Delta t_k, & \text{others}, \end{cases} \tag{11}$$

where the constants c_1, c_2, η_1, η_2 satisfy $0 < c_1 < 1 \le c_2$ and $0 < \eta_1 < \eta_2 < 1$, and $0 < \overline{\Delta t} < 1$ is a small threshold value. When $\rho_k \ge \eta_a$, we accept the trial step and set

$$x_{k+1} = x_k + s_k, \tag{12}$$

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

$$x_{k+1} = x_k. \tag{13}$$

Remark 2 This time-stepping selection based on the trust-region updating strategy has some advantages compared to the traditional line search strategy [53]. If we use the line search strategy and the damped Newton method (3) to track the trajectory $x(t)$ of ODE (5), in order to achieve the fast convergence rate in the steady-state phase, the time step α_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 $g(x_k) + H_k s_k^N$ may not approximate $g\left(x_k + s_k^N\right)$ well in the transient-state phase, the time step α_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, Hessian evaluations are too large if we update the Hessian matrix $H(x_k)$ at every iteration. In order to reduce the computational time of Hessian evaluations, similarly to the adaptively updating Jacobian technique [60,61], 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

$$H_{k+1} = \begin{cases} H_k, \text{ if } |1-\rho_k| \le \eta_1, \\ H(x_{k+1}), \text{ otherwise}, \end{cases} \tag{14}$$

where ρ_k is defined by equation (10). Thus, when H_k performs well, i.e. $|1-\rho_k| \le \eta_1$, 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 $f: \Re^n \to \Re$, an initial point x_0 (optional), the tolerance ε (optional), the Hessian matrix $H(x)$ of $f(x)$ (optional).

Output:

The found stationary point x^* of the objective function, the Boolean variable **success**.

Set the default $(x_0)_i = 1$ $(i = 1, 2, \ldots, n)$ and $\varepsilon = 10^{-6}$ when x_0 or ε is not provided.
Initialize the parameters: $\eta_a = 10^{-6}$, $\eta_1 = 0.25$, $\eta_2 = 0.75$, $c_1 = 0.5$, $c_2 = 2$, $\overline{\Delta t} = 10^{-7}$, $\Delta t_{\text{init}} = 10^{-2}$, maxit $= 200$.
Set $\Delta t_0 = \Delta t_{\text{init}}$, itc $= 0$, $k = 0$.
Let the Boolean variable **success** be **false**.
Let the Boolean variable **success_ap** be **true**.
Evaluate $g_0 = \nabla f(x_0)$.
Compute the residual $\text{Res}_0 = \|g_0\|_\infty$.
Set $\rho_{-1} = 0$, $s_{-1} = 0$.
while $\text{Res}_k > \varepsilon$ **do**
    Set itc $=$ itc $+ 1$.
    if itc $>$ maxit **then**
        break;
    end if
    if success_ap is **true then**
        if $|1 - \rho_{k-1}| > \eta_1$ **then**
            Evaluate $H_k = H(x_k)$.
        else
            Set $H_k = H_{k-1}$.
        end if
        Solve $H_k s_k^N = -g_k$ to obtain the Newton step s_k^N.
    end if
    Set $s_k = \Delta t_k/(1 + \Delta t_k)\, s_k^N$, $x_{k+1} = x_k + s_k$.
    Evaluate $g_{k+1} = \nabla f(x_{k+1})$.
    Compute the ratio ρ_k from equation (10).
    Adjust the time step Δt_{k+1} according to the trust-region updating strategy (11).
    if $\rho_k \geq \eta_a$ **then**
        Accept the trial point x_{k+1}. Compute the residual $\text{Res}_{k+1} = \|g_{k+1}\|_\infty$.
        Let the Boolean variable **success_ap** be **true**.
    else
        Set $x_{k+1} = x_k$, $g_{k+1} = g_k$, $s_{k+1}^N = s_k^N$, $\text{Res}_{k+1} = \text{Res}_k$.
        Let the Boolean variable **success_ap** be **false**.
    end if
    Set $k \longleftarrow k + 1$.
end while
if $\text{Res}_k \leq \varepsilon$ **then**
    Let the Boolean variable **success** be **true**.
else
    Let the Boolean variable **success** be **false**.
end if
Set $x^* = x_k$.
Output the found stationary point x^* and the Boolean variable **success**.

analysis of the global convergence and the local suplinear convergence of Algorithm 1, please refer to references [58,61].

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, η_1, η_2, $\overline{\Delta t}$ are used to adaptively adjust the time step Δt_k and they are defined by equation (11). The ratio ρ_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 η_1, η_2 $(0 < \eta_1 < 1 \leq \eta_2)$ are used to define the range of the ratio ρ_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 Δt_{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 $g(x_k + s_k)$ bad. Then, the time step Δt_{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 $g(x_k + s_k)$ neither well nor bad. Then, the time step Δt_{k+1} is kept with Δt_k. The parameter $0 < \overline{\Delta t} < 1$ is a small number and used to prevent the time step Δt_k from becoming too small and even zero, when the gradient $g(x)$ is very nonlinear and Δt_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.

In line 4, 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 5, the Boolean variable **success_ap** is used to indicate that the trial step s_k is accepted or not. When the trial step s_k is accepted, we let the Boolean variable **success_ap** be **true**. Otherwise, we let the Boolean variable **success_ap** 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 14-21, they give the procedures to solve the Newton step s_k^N. In line 9, we use the infinite norm $\text{Res}_k = \|g_k\|_\infty < \varepsilon$ as the loop termination condition, since 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 $\|g_k\| < \varepsilon$.

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 ρ_k according to equation (10). In line 25, it adaptively adjusts the time step Δt_{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 $\rho_k \geq \eta_a$ 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**.

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 multiple stationary points of $f(x)$. Then, we use these 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 those searches are initiated from points that are far apart. In order to overcome this disadvantage, we revise the deflation technique and combine it with Algorithm 1 to find multiple stationary points of $f(x)$ as possible.

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

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

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 zeros of $g(x)$. From equation (16), we know that the zero x_{k+1}^* of $G_k(x)$ is also the zero 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 [15] have proved

$$\liminf_{j \to \infty} \|G_k(x_j)\| > 0$$

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

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

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

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

holds for all $x \in B_{\delta_i}(x_i^*)$ $(i = 1, 2, \ldots, k)$, where $G_k(x)$ is defined by equation (16). That is to say, x_i^* $(i = 1, 2, \ldots, k)$ is not a zero of $G_k(x)$.

Proof. Since x_i^* is a zero of $g(x)$, according to the first-order Taylor expansion, we have

$$\begin{aligned} g(x) &= g(x_i^*) + \int_0^1 H(x_i^* + t(x - x_i^*))(x - x_i^*)dt \\ &= \int_0^1 H(x_i^* + t(x - x_i^*))(x - x_i^*)dt. \end{aligned} \tag{19}$$

According to the nonsingular assumption (17) of $H(x_i^*)$ and the continuity of $H(\cdot)$, there exists a neighborhood $B_{\delta_i}(x_i^*) = \{x : \|x - x_i^*\| \leq \delta_i\}$ of x_i^* such that

$$\|H(x) - H(x_i^*)\| \leq \frac{c_l}{2} \tag{20}$$

holds for all $x \in B_{\delta_i}(x_i^*)$. From equations (19)-(20), we have

$$\begin{aligned} &\|g(x) - H(x_i^*)(x - x_i^*)\| = \left\| \int_0^1 (H(x_i^* + t(x - x_i^*)) - H(x_i^*))(x - x_i^*)dt \right\| \\ &\leq \int_0^1 \|H(x_i^* + t(x - x_i^*)) - H(x_i^*)\| \|x - x_i^*\| dt \leq \frac{c_l}{2} \|x - x_i^*\|, \ \forall x \in B_{\delta_i}(x_i^*). \end{aligned} \tag{21}$$

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

$$\|g(x) - H(x_i^*)(x - x_i^*)\| \geq \|H(x_i^*)(x - x_i^*)\| - \|g(x)\| \geq c_l \|x - x_i^*\| - \|g(x)\|. \tag{22}$$

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

$$\|g(x)\| \geq \frac{c_l}{2} \|x - x_i^*\|, \ \forall x \in B_{\delta_i}(x_i^*). \tag{23}$$

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

$$\|G_k(x)\| \geq \frac{c_l}{2} \frac{1}{\|x - x_1^*\| \cdots \|x - x_{i-1}^*\| \|x - x_{i+1}^*\| \cdots \|x - x_k^*\|} > 0, \ \forall x \in B_{\delta_i}(x_i^*).$$

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

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 multiple stationary points of $f(x)$ as possible. According to our numerical experiments, Algorithm 1 can find more stationary points if we replace $\|x - x_i^*\|$ $(i = 1, \ldots, k)$ with $\|x - x_i^*\|_1$ $(i = 1, \ldots, k)$ in equation (16). We also notice that $\|x - x_i^*\|_1 \approx \|x_i^*\|_1 \approx n$ when x is close to zero, where n is the dimension of x_i^*. 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

$$G_k(x) = \frac{\alpha_k}{\|x - x_k^*\|_1} G_{k-1}(x) = \left(\prod_{i=1}^{k} \frac{\alpha_i}{\|x - x_i^*\|_1} \right) g(x), \tag{24}$$

where x_i^* $(i = 1, 2, \ldots, k)$ are found zeros of $g(x)$ and α_i $(i = 1, \ldots, k)$ are defined by

$$\alpha_i = \begin{cases} n, \text{ if } \|x_i^*\|_1 \leq 10^{-6}, \\ \|x_i^*\|_1, \text{ otherwise}, \end{cases} \quad i = 1, 2, \ldots, k. \tag{25}$$

Since $G_k(x)$ has the special structure and its subdifferential exists except for finite points $x_1^*, \ldots, x_k^*$, from equations (24)-(25), we compute its subdifferential $\partial G_k(x)$ as follows:

$$\begin{aligned} \partial G_k(x) &= \frac{\alpha_1 \cdots \alpha_k}{\|x - x_1^*\|_1 \cdots \|x - x_k^*\|_1} \frac{\partial g(x)}{\partial x} + \frac{\alpha_1 \cdots \alpha_k}{\|x - x_1^*\|_1 \cdots \|x - x_k^*\|_1} g(x) p_k(x)^T \\ &= \frac{\alpha_1 \cdots \alpha_k}{\|x - x_1^*\|_1 \cdots \|x - x_k^*\|_1} \left(\frac{\partial g(x)}{\partial x} + g(x) p_k(x)^T \right), \end{aligned} \tag{26}$$

where $p_k(x)$ is defined by

$$p_k(x)^T = -\left(\frac{\operatorname{sgn}(x - x_1^*)}{\|x - x_1^*\|_1} + \cdots + \frac{\operatorname{sgn}(x - x_k^*)}{\|x - x_k^*\|_1} \right). \tag{27}$$

Here, the sign function $\operatorname{sgn}(\cdot)$ is defined by

$$\operatorname{sgn}(\beta) = \begin{cases} 1, \text{ if } \beta > 0, \\ 0, \text{ if } \beta = 0, \\ -1, \text{ otherwise}. \end{cases} \tag{28}$$

Now, by using the revised deflation technique (24) of $g(x)$ and the continuation Newton method (Algorithm 1), we can find a new stationary point of $f(x)$ other than found 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 found 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 **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 **success_dt** be **true**. Otherwise, we let Boolean variable **success_dt** be **false**.

In lines 4-30, they give the pseudo codes of the continuation Newton method (C-NMTr) 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 **success_dt** be **true**. Then, this newly found stationary point is saved as x_{K+1}^*. They output

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 $f: \Re^n \to \Re$, K known stationary points $x_1^*, \ldots, x_K^*$ of $f(x)$, an initial point x_0 (optional), the tolerance ε (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 **success_dt**.

Set the default $x_0 = (1, \ldots, 1)^T$ and $\varepsilon = 10^{-6}$ when x_0 or ε is not provided.
Initialize the parameters: $\eta_a = 10^{-6}$, $\eta_1 = 0.25$, $\eta_2 = 0.75$, $c_1 = 0.5$, $c_2 = 2$, $\overline{\Delta t} = 10^{-7}$, $\Delta t_{\text{init}} = 10^{-2}$, maxit $= 200$.
Let the Boolean variable **success_dt** be **false**.
Set $\Delta t_0 = \Delta t_{\text{init}}$, itc $= 0$, $k = 0$.
Evaluate $g_0 = \nabla f(x_0)$. Then, compute $F_0 = G_K(x_0)$ from equations (24)-(25).
Set $\rho_{-1} = 0$, $s_{-1} = 0$.
Let the Boolean variable **success_ap** be **true**.
Compute the residual $\text{Res}_0 = \|F_0\|_\infty$.
while $\text{Res}_k > \varepsilon$ **do**
    itc $=$ itc $+ 1$.
    if itc $>$ maxit **then**
        break;
    end if
    if success_ap is true **then**
        Evaluate $H_k = H(x_k)$. Then, compute the Jacobian $J_k = \partial G_K(x_k)$ from equations (26)-(28).
        Solve $J_k s_k^N = -F_k$ to obtain the Newton step s_k^N.
    end if
    Set $s_k = \Delta t_k/(1+\Delta t_k)\, s_k^N$, $x_{k+1} = x_k + s_k$.
    Evaluate $g_{k+1} = \nabla f(x_{k+1})$. Then, compute $F_{k+1} = G_K(x_{k+1})$ from equations (24)-(25).
    Compute the ratio ρ_k from equation (10).
    Adjust the time step Δt_{k+1} according to the trust-region updating strategy (11).
    if $\rho_k \geq \eta_a$ **then**
        Accept the trial point x_{k+1}. Compute the residual $\text{Res}_{k+1} = \|F_{k+1}\|_\infty$.
        Let the Boolean variable **success_ap** be **true**.
    else
        Set $x_{k+1} = x_k$, $F_{k+1} = F_k$, $s_{k+1}^N = s_k^N$, $\text{Res}_{k+1} = \text{Res}_k$.
        Let the Boolean variable **success_ap** be **false**.
    end if
    Set $k \longleftarrow k+1$.
end while
if $\text{Res}_k \leq \varepsilon$ **then**
    Let the Boolean variable **success_dt** be **true**.
    Save the newly found stationary point as $x_{K+1}^* = x_k$.
    Output $K+1$ found stationary points $x_1^*, \ldots, x_{K+1}^*$ and the Boolean variable **success_dt**.
else
    Let the Boolean variable **success_dt** be **false**.
    Output K known stationary points $x_1^*, \ldots, x_K^*$ and the Boolean variable **success_dt**.
end if

$K+1$ found stationary points $x_1^*, \ldots, x_{K+1}^*$ and the Boolean variable **success_dt**. Otherwise, We let the Boolean variable **success_dt** be **false**. Then, they output K known stationary points $x_1^*, \ldots, x_K^*$ and the Boolean variable **success_dt**.

Remark 3 From equation (24), we know that $\nabla f(x_l)$ may converge to a positive number when $\lim_{l\to\infty} \|x_l - x_i^*\|_1 = \infty$, 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 multiple stationary points of $f(x)$ as possible, we use the continuation Newton method with the deflation technique (i.e. Algorithm 2) to find stationary points from several initial points that are far apart. We define the $\frac{n}{2}$-dimensional vector $e = (1, \ldots, 1)^T$ and select the following six points x_i^{init} $(i = 1, \ldots, 6)$ as the default initial points:

$$x_1^{\text{init}} = \begin{pmatrix} e \\ e \end{pmatrix}, x_2^{\text{init}} = -\begin{pmatrix} e \\ e \end{pmatrix}, x_3^{\text{init}} = \begin{pmatrix} e \\ -e \end{pmatrix}, x_4^{\text{init}} = \begin{pmatrix} -e \\ e \end{pmatrix},$$
$$x_5^{\text{init}}(i) = i,\ i = 1, 2, \ldots, n,\ x_6^{\text{init}}(i) = (n+1) - i,\ i = 1, 2, \ldots, n, \tag{29}$$

where $x_j^{\text{init}}(i)$ represents the i-th element of the n-dimensional vector x_j^{init}. Thus, we repeatedly use Algorithm 2 from multi-start points to find multiple stationary points of $f(x)$. We describe their detailed implementations 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 **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 **success** be **true**. Otherwise, we let the Boolean variable **success** be **false**. In line 4, **nsp** 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 **success** be **true**, set **nsp** $=$ **nsp** $+ 1$, save this found stationary point as x_1^*, and exit the loop. Otherwise, we let the Boolean variable **success** be **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 multiple stationary points of $f(x)$ from L default initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$. 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^{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 let **nsp** (the number of found stationary points) increase by 1 and save this newly found stationary point. Otherwise, we let the order variable i increase 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^{init}.

Algorithm 3 Continuation Newton methods with the deflation technique and multi-start points for finding multiple stationary points (CNMDTM)

Input:

the gradient $g(x) = \nabla f(x)$ of the objective function $f: \Re^n \to \Re$, the tolerance ε (optional), L initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ (optional), the Hessian matrix $H(x)$ of $f(x)$ (optional).

Output:

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

Set the default tolerance $\varepsilon = 10^{-6}$.
Set the default initial points x_i^{init} $(i = 1, \ldots, L)$ given by equation (29), where $L = 6$.
Let the Boolean variable **success** be **false**.
Set **nsp** $= 0$ (**nsp** represents the number of found stationary points of $f(x)$).
for $i = 1, \ldots, L$ **do**
    Select the i-th initial point as $x_0 = x_i^{\text{init}}$ and call Algorithm 1 to find a stationary point of $f(x)$.
    if a stationary point of $f(x)$ is found by Algorithm 1 successfully **then**
        Set **nsp** $=$ **nsp** $+ 1$.
        Let the Boolean variable **success** be **true**.
        Save this found stationary point of $f(x)$ as x_1^*.
        break;
    else
        Let the Boolean variable **success** be **false**.
    end if
end for
if the Boolean variable **success** is **true then**
    Set $i = 1$ (i represents the order of given initial points).
    while $i \le L$ **do**
        Select the i-th initial point as $x_0 = x_i^{\text{init}}$ and call Algorithm 2 to find a new stationary point.
        if a new stationary point of $f(x)$ is found successfully **then**
            Set **nsp** $=$ **nsp** $+ 1$.
            Save this newly found stationary point as $x_{\mathbf{nsp}}^*$.
            if this newly found stationary point $x_{\mathbf{nsp}}^*$ almost equals the initial point x_i^{init} **then**
                Set $i = i + 1$.
            end if
        else
            Set $i = i + 1$.
        end if
    end while
end if
if nsp > 0 **then**
    Set $f_{ap}^* = f(x_1^*)$ and $x_{ap}^* = x_1^*$.
    for $k = 1, \ldots,$ **nsp do**
        Set $f_k^* = f(x_k^*)$.
        if $f_k^* < f_{ap}^*$ **then**
            Set $x_{ap}^* = x_k^*$ and $f_{ap}^* = f_k^*$.
        end if
    end for
    Let the Boolean variable **success** be **true**.
    Output all found stationary points $x_1^*, \ldots, x_{\mathbf{nsp}}^*$ of $f(x)$, its global optimal approximation point x_{ap}^*, its optimal approximation value f_{ap}^*, and the Boolean variable **success**.
else
    Set $x_{ap}^* = x_0$ and $f_{ap}^* = f(x_0)$.
    Let the Boolean variable **success** be **false**.
    Output x_{ap}^*, f_{ap}^* and the Boolean variable **success**.
end if

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 $x_1^*, \ldots, x_{\text{nsp}}^*$, the optimal approximation point $x_{ap}^* = \arg\min\{f(x_1^*), \ldots, f(x_{\mathbf{nsp}}^*)\}$, 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 $x_1^*, \ldots, x_{\mathbf{nsp}}^*$ 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 (that would imply $f(x_{ap}^*)$ is a global minimum). Therefore, in order to find the global minimum at all possible, we use the idea of memetic algorithms [71,84] 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 $x_1^*, \ldots, x_{\mathbf{nsp}}^*$ 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. CNMTr, 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 $\frac{n}{2}$-dimensional vector $e = (1, \ldots, 1)^T$. Then, we select the following five basic seeds:

$$x_0^{bs} = 0, \quad x_1^{bs} = \begin{pmatrix} e \\ e \end{pmatrix}, \quad x_2^{bs} = \begin{pmatrix} e \\ -e \end{pmatrix}, \quad x_3^{bs} = \begin{pmatrix} -e \\ e \end{pmatrix}, \quad x_4^{bs} = -\begin{pmatrix} e \\ e \end{pmatrix}. \tag{30}$$

We denote the set S_0^{bs} as

$$S_0^{bs} = \left\{ x_1^{bs}, x_2^{bs}, x_3^{bs}, x_4^{bs} \right\}. \tag{31}$$

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

$$S_0^{\text{seed}} = \left\{ 0, 10^{-1} \times S_0^{bs}, S_0^{bs}, 10 \times S_0^{bs}, 10^2 \times S_0^{bs}, 10^3 \times S_0^{bs}, \ldots \right\}. \tag{32}$$

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

$$\bar{S}_0 = \left\{ x_1^*, \ldots, x_{\mathbf{nsp}}^*, S_0^{\text{seed}} \right\} \tag{33}$$

and denote these L points as x_i^0 $(i=1,\ldots,L)$, where the set S_0^{seed} is defined by equation (32). We denote

$$S_0 = \left\{x_1^0, \ldots, x_L^0\right\} \tag{34}$$

and adopt them as the initial seeds of the quasi-genetic algorithm.

Secondly, we select any two elements x_i^0 and x_j^0 from the seed set S_0, and use the convex crossover to generate the next offspring. Namely, we set

$$\bar{x}_k^1 = \frac{1}{2}\left(x_i^0 + x_j^0\right),\ i=1,\ldots,L,\ j=i,i+1,\ldots,L,\ \ k=1,2,\ldots,\frac{1}{2}L(L-1). \tag{35}$$

Similarly, we select the first L minimum points of $f(x)$ from the set

$$\bar{S}_1 = S_0 \bigcup \left\{\bar{x}_1^1, \ldots, \bar{x}_{L(L-1)/2}^1\right\} \tag{36}$$

and denote these L points as x_i^1 $(i=1,\ldots,L)$. We set

$$S_1 = \left\{x_1^1, \ldots, x_L^1\right\} \tag{37}$$

and regard them as the dominant population.

Repeatedly, after the l-th evolution, we generate the offspring $\left\{\bar{x}_1^{l+1}, \ldots, \bar{x}_{L(L-1)/2}^{l+1}\right\}$ and select the first L minimum points of $f(x)$ from the set

$$\bar{S}_{l+1} = S_l \bigcup \left\{\bar{x}_1^{l+1}, \ldots, \bar{x}_{L(L-1)/2}^{l+l}\right\},\ l=0,1,\ldots. \tag{38}$$

We denote these L points as x_i^l $(i=1,\ldots,L)$, and set

$$S_{l+1} = \left\{x_1^{l+1}, \ldots, x_L^{l+1}\right\},\ l=0,1,\ldots. \tag{39}$$

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_{ag}^*.

Finally, in order to improve the convergence of the quasi-genetic evolution, we call Algorithm 1 (CNMTr) from x_{ag}^* to find the stationary point of $f(x)$. We denote this found stationary point as x_{cn}^*. Then, we select the minimum point $x_{\min}^*$ of $f(x)$ between x_{cn}^* and x_{ag}^*, i.e.

$$x_{\min}^* = \arg\min\left\{f(x_{cn}^*),\ f(x_{ag}^*)\right\},$$

and output $x_{\min}^*$ 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. **nsp** represents the number of the known stationary points of $f(x)$. In line 2, L represents the number of population and its default value equals 21. It supplements L candidate seeds $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ 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 $x_1^*, \ldots, x_{\mathbf{nsp}}^*$ of $f(x)$ and L initial points $x_1^{\text{init}}, \ldots, x_L^{\text{init}}$ (optional), the tolerance ε (optional), the Hessian matrix $H(x)$ of $f(x)$ (optional).

Output:

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

1: Set the default tolerance $\varepsilon = 10^{-6}$.
2: Set $L = 21$ and select the default initial points x_i^{init} $(i = 1, \ldots, L)$ from S_0^{seed} 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 $\bar{S}_0$ defined by equation (33), and denote these L points as x_i^0 $(i = 1, \ldots, L)$.
5: Set $S_0 = \left\{x_1^0, \ldots, x_L^0\right\}$.
6: **while** $(l < N)$ **do**
7: Set $k = 1$.
8: **for** $i = 1, \ldots, L$ **do**
9: **for** $j = (i+1), \ldots,$ **do**
10: Set $\vec{x}_k^{l+1} = \frac{1}{2}\left(x_i^l + x_j^l\right)$.
11: Set $k = k + 1$.
12: **end for**
13: **end for**
14: Select the first L minimum points of $f(x)$ from the set $\bar{S}_{l+1}$ defined by equation (38), and denote these L points as x_i^{l+1} $(i = 1, \ldots, L)$.
15: Set $S_{l+1} = \left\{x_1^{l+1}, \ldots, x_L^{l+1}\right\}$.
16: Set $l \longleftarrow l + 1$.
17: **end while**
18: Set $f_{ag}^* = f\left(x_1^N\right)$ and $x_{ag}^* = x_1^N$.
19: Use x_{ag}^* as the initial point and call Algorithm 1 to find a stationary point x_{cn}^* of $f(x)$.
20: Set $x_{\min}^* = \arg\min\left\{f(x_{cn}^*),\ f(x_{ag}^*)\right\}$.
21: Output $x_{\min}^*$ as the global optimal approximation point.

set S_0^{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 $\bar{S}_0$ defined by equation (33) as the initial seeds of the evolutionary algorithm, and denotes the first L minimum points as $x_1^0, \ldots, x_L^0$. In line 5, we denote the set S_0 of the initial seeds as $S_0 = \{x_1^0, \ldots, x_L^0\}$.

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 $\left\{x_1^N, \ldots, x_L^N\right\}$. 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_{cn}^* of $f(x)$. In lines 20-21, they select the minimum from $\{f(x_{ag}^*), f(x_{cn}^*)\}$, and denote this minimum as $f(x_{\min}^*)$. Finally, it outputs $x_{\min}^*$ 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 descriptions 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 multiple stationary points of $f(x)$ from multi-start points. 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 $x^*_1, \ldots, x^*_{\mathbf{nsp}}$ as the input of the subroutine QGE.

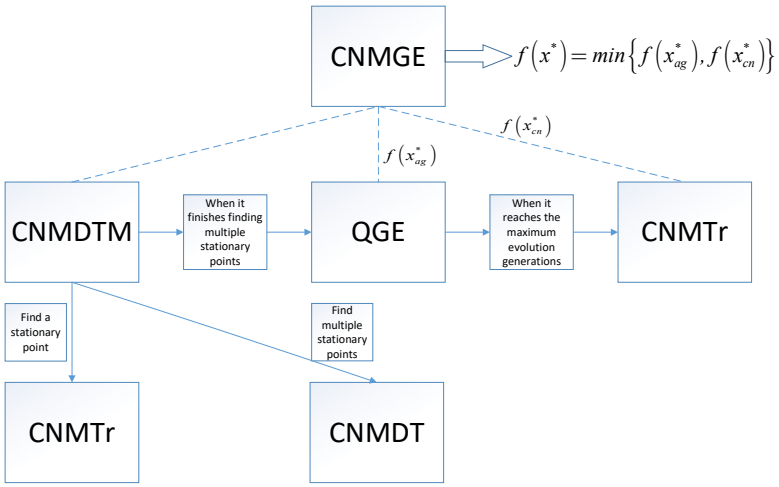

Fig. 1: The implementation diagram of CNMGE.

In order to retain the usability of derivative-free methods, we use the automatic differentiation technique [9,34,73,94,95] 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 [34]. We implement it via calling the built-in subroutine prob2struct.m of MATLAB 2021b. 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:

$$H(x_k) \approx \left[\frac{g(x_k + \varepsilon e_1) - g(x_k)}{\varepsilon}, \ldots, \frac{g(x_k + \varepsilon e_n) - g(x_k)}{\varepsilon} \right], \tag{40}$$

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 $\eta_1 = 1/4,\ \eta_2 = 3/4,\ c_1 = 1/2,\ c_2 = 2$ as the default parameters. According to our numerical experiments, these default 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 Δt_{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 [76, 89].

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 [80]

$$f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2. \tag{41}$$

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

```
n = 2;
x0 = 2 * ones(n, 1);
% Define the objective function.
x = optimvar('x', n, 1);
objfun = 100 * (x(2)^2 - x(1))^2 + (x(1) - 1)^2;
lb = -inf;
ub = inf;
unitdisk = x(1)^2 <= 1;
prob = optimproblem("Objective", objfun, "Constraints", unitdisk);
problem = prob2struct(prob,"ObjectiveDerivative","auto-reverse");
[x_opt, f_opt, CPU_time] = CNMGE(problem.objective, x0, ub, lb);
% Output the global minimum value computed by CNMGE.
fprintf("The global minimum value computed by CNMGE is %12.8f \n", f_opt);
% Output the computational time of CNMGE.
fprintf("The computational time of CNMGE is %8.4f\n", CPU_time);
```

In line 4, the subroutine **optimvar** defines an n-dimensional 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 ∞. 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. For a nonlinear least-square problem $\min f(x) = r(x)^T r(x)$, $r: \Re^n \to \Re$, **prob2struct** returns the Jacobian $J(x)$ of $r(x)$ and it is favorable to the solver of a nonlinear least-square problem such as the Gauss-Newton method (p. 279 in [76]) or the Levenberg-Marquardt method (p. 282 in [76]).

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 [67,93], GLODS: Global and Local Optimization using Direct Search [22], VRBBO: Vienna Randomized Black Box Optimization [47]), the branch-and-bound method (Couenne: Convex Over- and Under-ENvelopes for Nonlinear Estimation, one of the state-of-the-art open-source solver [10,19]), and the derivative-free algorithms (CMA-ES [36,37] and MCS [41,74]) for 148 open test problems from CUTEst [35].

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 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 [2,3,30,50,70,83] and their scales vary from dimension 1 to 1000. For problems 37, 60, 62, 67, the original test problems have the default box constraints. Since we only consider the global optimization problems, we compute these problems without the default box constraints.

In order to evaluate the effect of the algorithm gradient computed by automatic differentiation, we also compare CNMGE using the algorithm gradient computed 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

$$\|\nabla f(x_k)\|_\infty \leq 10^{-6}. \tag{42}$$

The codes are executed by a HP notebook with the Intel quad-core CPU and 8Gb memory in the MATLAB2021b environment. The numerical results are arranged in Tables 3-6 of Appendix A 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 3-6. 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 statistical results are put in Table 1.

From Table 1, we find that CNMGE and CNMGE_AG work well and almost obtain the global minima of 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 9 test problems about 13.23%. 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. 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.

Table 1: The number of failure problems (no. 1-68) computed by CNMTrM, CNMDTM, CNMGE and CNMGE_AG.

	CNMTrM	CNMDTM	CNMGE	CNMGE_AG
The number of failed problems	27	9	3	3
The probability of failure	$\frac{27}{68}$ (39.71%)	$\frac{9}{68}$ (13.23%)	$\frac{3}{68}$ (4.41%)	$\frac{3}{68}$ (4.41%)

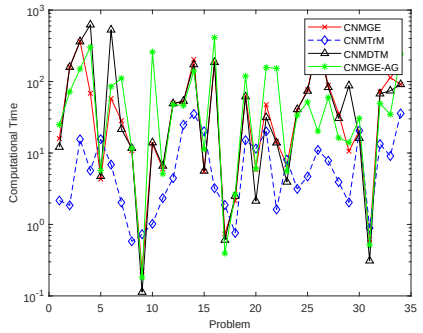

Fig. 2: CPU time of large-scale problems (no. 1-34) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

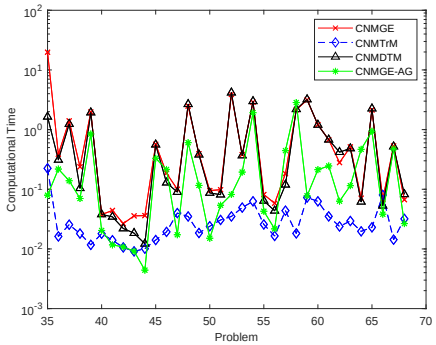

Fig. 3: CPU time of small-scale problems (no. 35-68) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

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, GLODS [22], and VRBBO [47]), the branch-and-bound method (Couenne, one of the state-of-the-art open-source solver [10,19]), and the derivative-free algorithms (CMA-ES [36,37] and MCS [41,74]). 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 [21,25,32,75].

GlobalSearch is an efficient multi-start method for global optimization problems and it is widely used in engineering fields. GLODS (Global and Local Optimization using Direct Search [22]) and VRBBO (Vienna Randomized Black Box Optimization, its MATLAB code can be downloaded from [47]) 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 [20]) and the multilevel coordinate search (MCS, its MATLAB code can be downloaded from [68]) are two representative derivative-free algorithms. Therefore, we select these six methods (GlobalSearch, GLODS, VRBBO, Couenne, CMA-ES and MCS) as the basis for comparison.

In order to test those seven methods (CNMGE, GlobalSearch, GLODS, VRBBO, Couenne, CMA-ES and MCS) more sufficiently, we not only test them for 68 unconstrained optimization problems from the references [2,3,30,50,70,83], but also test them for additional 80 unconstrained optimization problems from a larger publicly available test set (CUTEst [35]). The numerical results are arranged in Tables 7-24 of Appendix A. Since the global minima of some test problems are unknown, we mark their global minima of those problems with the notation "\" in Tables 7-24. 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 7-24, respectively.

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 CNMGE possibly find the global minimum of $f(x)$ for the difficult problem. We illustrate this result via Problem 1. Problem 1 [50] is a molecular potential energy problem and its objective function has the following form:

$$f(\omega) = \sum_{i=1}^{n} \left(1 + cos(3\omega_i) + \frac{(-1)^i}{\sqrt{10.60099896 - 4.141720682(cos\omega_i)}} \right), \quad (43)$$

where ω_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 [44,50] or the multi-start methods (GlobalSearch, GLODS, VRBBO) are difficult to find its global minimum. Couenne, CMA-ES and MCS also fail to solve this problem.

In order to evaluate and compare those seven methods (CNMGE, GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO) fairly, we also adopt the performance profile as a evaluation tool [26,87]. The performance profile for a solver is the (cumulative) distribution function for a performance metric, which is the ratio of the computing time of the solver versus the best time of all of the solvers as the performance metric. If the solver fails to solve a problem, we let its ratio be a bigger number such as 999. Figure 4 is the performance profiles for seven global optimization solvers (CNMGE, GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO). We also count the statistic number of failed problems and fasted problems computed by CNMGE, GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO. The statistical results are put in Table 2.

From Tables 7-24, Table 2 and Figure 4, we find that CNMGE works well for unconstrained optimization problems and finds their global minima efficiently. GlobalSearch, Couenne, CMA-ES and MCS all work well for small-scale problems. From Table 4, we find that CNMGE fails to find the global minima of test problems about $14/148$ (9.46%). GlobalSearch, CMA-ES, MCS and VRBBO fail to find the global minima of test problems about $50/148$ (33.78%). Couenne can solve the most problems efficiently and it can not solve efficiently about $35/148$ (23.65%) (Couenne fails to solve 11 test problems and 24 problems are solved by it about 8 hours). Therefore,

CNMGE is more robust than other representative global optimization methods such as GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO.

From Table 2, we also find that Couenne is significantly faster than the other six global optimization solvers (CNMGE, GlobalSearch, CMA-ES, MCS, GLODS and VRBBO). One reason is due to the more gain of the complied language than that of the interpreted language, since Couenne is written by C++ language (a complied language), and the other six solvers ((CNMGE, GlobalSearch, CMA-ES, MCS, GLODS and VRBBO) are written by MATLAB language (an interpreted language). The other reason is that Couenne is executed by a high performance (NEOS Server [75], `https://neos-server.org/neos/`) and the other six solvers ((CNMGE, GlobalSearch, CMA-ES, MCS, GLODS and VRBBO) are executed by a HP notebook with the Intel quad-core CPU and 8Gb memory.

Table 2: The number of failed problems for 148 test problems computed by CNMGE, GlobalSearch, Couenne, CMA-ES, MCS, GLODS and VRBBO.

	CNMGE	GlobalSearch	Couenne	CMA-ES
The number of failed problems	14	50	11	50
The number of problems computed by 8 hours	\	\	24	5
The probability of failure	$\frac{14}{148}$ (9.46%)	$\frac{50}{148}$ (33.78%)	$\frac{35}{148}$ (23.65%)	$\frac{55}{148}$ (37.16%)
The number of fasted problems	13	13	72	3
The probability of fasted problem	$\frac{13}{148}$ (8.78%)	$\frac{13}{148}$ (8.78%)	$\frac{72}{148}$ (48.65%)	$\frac{3}{148}$ (2.03%)
	MCS	GLODS	VRBBO	\
The number of failed problems	45	64	50	\
The number of problems computed by 8 hours	10	\	\	\
The probability of failure	$\frac{55}{148}$ (37.16%)	$\frac{64}{148}$ (43.24%)	$\frac{50}{148}$ (33.78%)	\
The number of fasted problems	5	2	40	
The probability of fasted problem	$\frac{5}{148}$ (3.38%)	$\frac{2}{148}$ (1.35%)	$\frac{40}{148}$ (27.03%)	

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

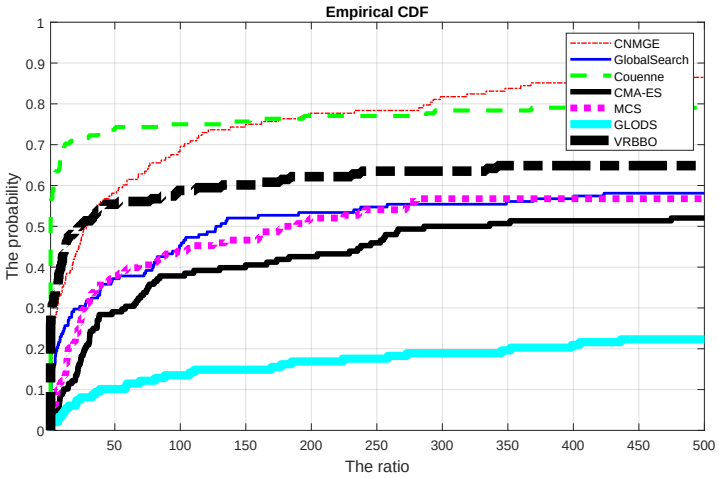

Fig. 4: Performance profile for global optimization solvers.

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.

We only compare CNMGE with some state-of-the-art open source methods. And we do not compare CNMGE with commercial solvers such as BARON [82] or non-open source solvers. In the future, it is worthy to compare CNMGE with some no-open source state-of-the-art methods such as the methods, which are proposed by Y. D. Sergeyev and D. E. Kvasov [48,88], or the commercial solvers such as BARON, which is written by N. V. Sahinidis [82]. Furthermore, CNMGE is worthy to be explored further for non-smooth optimization problems by combining the technique of estimating the Lipschitz constant [48,86,88]. And the computational efficiency of CNMGE can be improved further if its parameters are selected elaborately or it combines the state-of-the-art acceleration technique [13].

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 authors are grateful to two anonymous referees for their comments and suggestions which greatly improve the presentation of this paper.

Declarations

Funding: This work was supported in part by Grants 61876199 and 62376036 from National Natural Science Foundation of China, Grant YBWL2011085 from Huawei Technologies Co., Ltd., and Grant YJCB2011003HI from the Innovation Research Program of Huawei Technologies Co., Ltd..

Conflicts of interest / Competing interests: Not applicable.

Ethical Approval: 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 MINIPACK-2 Test Problem Collection*, Mathematics and Computer Science Division, Agronne 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. Brezinski, C., Redivo-Zaglia, M., Saad, Y: *Shanks sequence transformations and Anderson acceleration*, SIAM Rev. **60** (2018), 646-649.
14. Brezinski, C., Meurant, G., Redivo-Zaglia, M.: *A Journey through the History of Numerical Linear Algebra*, SIAM, Philadelphia, 2022.
15. Brown, K. M., Gearhart, W. B.: *Deflation techniques for the calculation of further solutions of a nonlinear system*, Numer. Math. **16** (1971), 334-342.
16. Braden, A.: *Optimisation techniques for solving design problems in modern trombones*, In Forum Acusticum 2005, 557-662.
17. Conn, A. R., Gould, N., Toint, Ph. L.: *Trust-Region Methods*, SIAM, Philadelphia, PA, 2000.
18. Conn, A. R., Scheinberg, K., Vicente, L. N.: *Introduction Derivative-free Optimization*, SIAM, Philadelphia, PA, 2009.

19. Couenne, *a solver for non-convex MINLP problems*, available at https://www.coin-or.org/Couenne/, February 2020.
20. CMA-ES, *The covariance matrix adaptation evolution strategy*, available at http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaes.m, 2012.
21. Czyzyk, J., Mesnier, M. P., Moré, J. J.: *The NEOS Server*, IEEE Comput. Sci. Eng. **5** (1998), 68-75.
22. 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.
23. 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.
24. Deuflhard, P.: *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms*, Springer-Verlag, Berlin, 2004.
25. Dolan, E. D.: *The NEOS Server 4.0 Administrative Guide*, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.
26. Dolan, E. D., Moré, J. J.: *Benchmarking optimization software with performance profiles*, Math Program **91** (2002), 201-213.
27. 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.
28. Dennis, J. E., Schnabel, R. B.: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Philadelphia, PA, 1996.
29. 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.
30. 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.
31. Gao, W., Mi, C.: *Hybrid vehicle design using global optimisation algorithms*, Int. J. Electric Hybrid Veh. **1** (2007), 57-70.
32. 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.
33. Golub, G. H., Van Loan, C. F.: *Matrix Computation* (4th ed.), The John Hopkins University Press, Baltimore, 2013.
34. Griewank, A. and Walther, A.: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, 2008. https://doi.org/10.1137/1.9780898717761.
35. Gould, N. I. M, Orban, D., and Toint, Ph. L.: *CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization*, Comput Optim Appl **60** (2015), 545-557, https://www.cuter.rl.ac.uk/mastsif.html.
36. Hansen, N.: *The CMA evolution strategy: a comparing reivew*, In: Lozano, J. A., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer, Berlin, 75-102 (2006).
37. Hansen, N.: *The CMA Evolution Strategy: A Tutorial*, available at https://arxiv.org/abs/1604.00772, 2010.
38. 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.
39. Hart, W. E.: *Adaptive Global Optimization with Local Search*, Ph.D. dissertation, University of California, San Diego, CA, USA, 1994.
40. Higham, D. J.: *Trust region algorithms and timestep selection*, SIAM J. Numer. Anal. **37** (1999), 194-210.
41. Huyer, W., Neumaier, A.: *Global optimization by multilevel coordinate search*, J. Glob. Optim. **14**, 331-355 (1999).
42. Hairer, E., Wanner, G.: *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*, 2nd ed., Springer-Verlag, Berlin, 1996.
43. Jackiewicz, Z.: *General Linear Methods for Ordinary Differential Equations*, John Wiley & Sons, Inc., Hoboken, New Jersey, 2009.
44. Kearfott, R. B.: *Rigorous Global Search: Continuous Problems*, Nonconvex Optimization and Applications, Kluwer Academic, Dordrecht, 1996.
45. Kelley, C. T.: *Solving Nonlinear Equations with Newton's Method*, SIAM, Philadelphia, PA, 2003.
46. Kelley, C. T.: *Numerical methods for nonlinear equations*, Acta Numer. **27** (2018), 207-287.

47. Kimiaei, M., Neumaier, A.: *Efficient unconstrained black box optimization*, Math. Program. Comput. **14** (2022), 365-414. https://doi.org/10.1007/s12532-021-00215-9. Software available at https://arnold-neumaier.at/software/VRBBO/.
48. Kvasov, D. E., Sergeyev, Y. D. : *Lipschitz gradients for global optimization in a one-point-based partitioning scheme*, J. Comput. Appl. Math. **236** (2012), 4042-4054.
49. Lambert, J. D.: *Computational Methods in Ordinary Differential Equations*, John Wiley, 1973.
50. Lavor, C., Maculan, N.: *A function to test methods applied to global minimization of potential energy of molecules*, Numer. Algorithms **35** (2004), 287-300.
51. Leung, Y.-W., Wang, Y. P.: *An orthogonal genetic algorithm with quantization for global numerical optimization*, IEEE Trans. Evol. Comput. **5** (2001), 41-53.
52. Liu, S.-T., Luo, X.-L.: *A method based on Rayleigh quotient gradient flow for extreme and interior eigenvalue problems*, Linear Algebra Appl. **432** (2010), 1851-1863.
53. Luo, X.-L.: *Singly diagonally implicit Runge-Kutta methods combining line search techniques for unconstrained optimization*, J. Comput. Math. **23**, 153-164 (2005)
54. Luo, X.-L., Kelley, C. T., Liao, L.-Z., Tam, H.-W.: *Combining trust region techniques and Rosenbrock methods to compute stationary points*, J. Optim. Theory Appl. **140** (2009), 265-286.
55. Luo, X.-L.: *A second-order pseudo-transient method for steady-state problems*, Appl. Math. Comput., **216** (2010), 1752-1762.
56. Luo, X.-L.: *A dynamical method of DAEs for the smallest eigenvalue problem*, J. Comput. Sci. **3** (2012), 113-119.
57. Luo, X.-L., Lv, J.-H., Sun, G.: *Continuation method with the trusty time-stepping scheme for linearly constrained optimization with noisy data*, Optim. Eng. **23** (2022), 329-360. http://doi.org/10.1007/s11081-020-09590-z.
58. Luo, X.-L., Xiao, H., Lv, J.-H.: *Continuation Newton methods with the residual trust-region time-stepping scheme for nonlinear equations*, Numer. Algorithms **89** (2022), 223-247. http://doi.org/10.1007/s11075-021-01112-x.
59. Luo, X.-L., Yao, Y. Y.: *Primal-dual path-following methods and the trust-region strategy for linear programming with noisy data*, J. Comput. Math. **40** (2022), 760-780. http://doi.org/10.4208/jcm.2101-m2020-0173.
60. Luo, X.-L., Xiao, H., Lv, J.-H., Zhang, S.: *Explicit pseudo-transient continuation and the trust-region updating strategy for unconstrained optimization*, Appl. Numer. Math. **165** (2021), 290-302. http://doi.org/10.1016/j.apnum.2021.02.019.
61. Luo, X.-L., Xiao, H.: *Generalized continuation Newton methods and the trust-region updating strategy for the underdetermined system*, J. Sci. comput. **88** (2021), article 56, 1-22. http://doi.org/10.1007/s10915-021-01566-0.
62. Luo, X.-L., Xiao, H.: *The regularization continuation method with an adaptive time step control for linearly constrained optimization problems*, Appl. Numer. Math. **181** (2022), 255-276. https://doi.org/10.1016/j.apnum.2022.06.008.
63. Luo, X.-L., Zhang, S., Xiao, H.: *Regularization path-following methods with the trust-region updating strategy for linear complementarity problems*, arXiv preprint available at http://arxiv.org/abs/2205.10727, pp. 1-30, May 21, 2022.
64. Luo, X.-L., Xiao, H., Zhang, S.: *The regularization continuation method for optimization problems with nonlinear equality constraints*, arXiv preprint available at http://arxiv.org/abs/2303.14692, pp. 1-41, March 28, 2023.
65. Man, K. F., Tang, K. S., Kwong, S.: *Genetic Algorithms: Concepts and Designs*, Springer, Berlin, 1999.
66. Macêdo, M. J. F. G., Karas, E. W., Costa, M. F. P., Rocha, A. M. A. C.: *Filter-based stochastic algorithm for global optimization*, J. Glob. Optim. **77** (2020), 777-805.
67. MATLAB R2021b, The MathWorks Inc., http://www.mathworks.com, 2021.
68. MCS, *The multilevel coordinate search*, available at https://www.mat.univie.ac.at/~neum/software/mcs/, 2000.
69. Mitchell, M.: *An Introduction to Genetic Algorithms*, MIT press, Cambridge, MA, 1996.
70. Moré, J. J., Garbow, B. S., Hillstrom, K. E.: *Testing unconstrained optimization software*, ACM Trans. Math. Soft. **7** (1981), 17-41.
71. Moscato, P.: *On evolution, search, optimization, gas and martial arts: Toward memetic algorithms*, Technical report, Caltech Concurrent Computation Program 158-79, California Institute of Technology, Pasadena, California, 1989.

72. Morgans, R. C., Howard, C. Q., Zander, A. C., Hansen, C. H., Murphy, D. J.: *Derivative free optimisation in engineering and acoustics*, 14th International Congress on Sound & Vibration, 2007, 1-8.
73. Neidinger, R. D.: *Introduction to automatic differentiation and MATLAB object-oriented programming*, SIAM Rev. **52** (2010), 545-563. https://doi.org/10.1137/080743627.
74. Neumaier, A.: *MCS: Global Optimization by Multilevel Coordinate Search*, https://www.mat.univie.ac.at/~neum/software/mcs/, 2000.
75. NEOS Server, https://neos-server.org/neos/, 2021.
76. Nocedal, J., Wright, S. J.: *Numerical Optimization*, Springer-Verlag, Berlin, 1999.
77. Ortega, J. M., Rheinboldt, W. C.: *Iteration Solution of Nonlinear Equations in Several Variables*, SIAM, Philadelphia, PA, 2000.
78. 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.
79. 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.
80. Rosenbrock, H. H.: *An automatic method for finding the greatest or least value of a function*, Comput. J. **3** (1960), 175-184. Aailable online at http://comjnl.oxfordjournals.org/content/3/3/175.full.pdf.
81. Shampine, L. F., Gladwell, I., Thompson, S.: *Solving ODEs with MATLAB*, Cambridge University Press, Cambridge, 2003.
82. 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.
83. Surjanovic, S., Bingham, D.: *Virtual library of simulation experiments: test functions and datasets*, available at http://www.sfu.ca/~ssurjano, January 2020.
84. Sun, J., Garibaldi, J. M., Krasnogor, N., Zhang, Q.: *An intelligent muti-restart memetic algorithm for box constrained global optimisation*, Evol. Comput. **21** (2014), 107-147.
85. 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.
86. Sergeyev, Y. D., Kvasov, D. E.: *Deterministic Global Optimization: An Introduction to the Diagonal Approach*, Springer, 2017.
87. Sergeyev, Y. D., Kvasov, D. E., Mukhametzhanov, M. S.: *Operational zones for comparing metaheuristic and deterministic one-dimensional global optimization algorithms*, Math. Comput. Simul. **141** (2017), 96-109.
88. Sergeyev, Y. D., Kvasov, D. E.: *A deterministic global optimization using smooth diagonal auxiliary functions*, Commun. Nonlinear Sci. **21** (2015), 99-111.
89. Sun, W.-Y, Yuan, Y.-X.: *Optimization Theory and Methods: Nonlinear Programming*, Springer, Berlin, 2006.
90. Tanabe, K.: *Continuous Newton-Raphson method for solving an underdetermined system of nonlinear equations*, Nonlinear Anal. **3** (1979), 495-503.
91. Tawarmalani, M., Sahinidis, N.V.: *A polyhedral branch-and-cut approach to global optimization*, Math. Program. **103** (2005), 225-249.
92. Teughels, A., Roeck, G. De, Suykens, J. A. K.: *Global optimization by coupled local minimizers and its application to FE model updating*, Comput. Sturct. **81** (2003), 2337-2351.
93. 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.
94. Willkomm, J. and Vehreschild, A.: *The ADiMat handbook*, 2013. http://adimat.sc.informatik.tu-darmstadt.de/doc/.
95. Willkomm, J., Bischof, C. H., Bücker, 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.
96. Xu, J., Nannariello, J., Fricke, F. R.: *Optimising flat-walled multi-layered anechoic linings using evolutionary algorithms*, Appl. Acoust. **65** (2004), 1009-1026.
97. Yuan, Y.-X.: *Trust region algorithms for nonlinear equations*, Information **1** (1998), 7-20.
98. Yuan, Y.-X.: *Recent advances in trust region algorithms*, Math. Program. **151** (2015), 249-281.
99. Žilinskas, A., Gillard, J., Scammell, M., Zhiglijavsky, A.: *Multistart with early termination of descents*, J. Glob. Optim. **79** (2021), 447-462.

A Tables of Numerical Results

Table 3: Numerical results of large-scale problems (no. 1-17) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

Problem	CNMTrM min $f(x)$ (CPU time (s))	CNMDTM min $f(x)$ (CPU time (s))	CNMGE min $f(x)$ (CPU time (s))	CNMGE_AG min $f(x)$ (CPU time (s))	Global minimum
1. Molecular Energy Problem (n = 1000) [50]	-2.8789E-01 (2.16E+00) (failed)	-41.1183E+00 (1.21E+01)	-41.1183E+00 (1.59E+01)	-41.1183E+00 (2.49E+01)	-4.1118E+01
2. Ackley Function (n = 1000) [83]	3.5744E+0 (1.85E+0) (failed)	3.5744E+00 (1.59E+02) (failed)	8.8817E-16 (1.60E+02)	8.8817E-16 (71.79)	0
3. Levy Function (n = 1000) [83]	4.4674E+01 (1.55E+01) (failed)	1.1513E-15 (3.6220E+02)	1.4997E-32 (3.67E+02)	1.4998E-32 (1.50E+02)	0
4. Schwefel Function (n = 1000) [83]	2.1713E+05 (5.67E+00) (failed)	1.2728E-02 (6.23E+01)	1.2728E-02 (6.83E+01)	1.2728E-02 (3.03E+02)	1.2728E-02
5. Rastrigin Function (n = 1000) [83]	9.9495E+02 (1.57E+01) (failed)	0 (4.72E+00)	0 (4.33E+00)	0 (5.70E+00)	0
6. Styblinski-Tang Function (n = 1000) [83]	-3.9126E+04 (6.87E+00)	-3.9166E+04 (5.29E+01)	-3.9166E+04 (5.78E+01)	-3.9166E+04 (8.50E+01)	-3.9166E+04
7. Trid Function (n = 1000) [83]	-1.7000E+08 (2.03E+00)	-1.6716E+08 (2.15E+01)	-1.6716E+08 (2.84E+01)	-1.6716E+08 (1.11E+02)	-1.6716E+08
8. Sum Squares Function (n = 1000) [83]	1.2528E-15 (5.84E-01)	1.2528E-15 (1.17E+01)	0 (1.08E+01)	0 (1.16E+01)	0
9. Sphere Function (n = 1000) [83]	6.8029E-14 (7.3459E-01)	6.8029E-14 (1.13E-01)	0 (2.23E-01)	0 (1.82E-01)	0
10. Rotated Hyper-Ellipsoid Function (n = 1000) [83]	1.2528E-15 (1.02E+00)	1.2528E-15 (1.40E+01)	0 (1.34E+01)	0 (2.59E+02)	0
11. Zakharov Function (n = 1000) [83]	1.3555E-19 (2.33E+00)	1.3555E-19 (6.58E+00)	0 (6.00E+00)	0 (5.11E+00)	0
12. Dixon-Price Function (n = 1000) [83]	1.8058E-15 (4.45E+00)	9.8100E-14 (4.92E+01)	6.2737E-14 (4.96E+01)	6.2737E-14 (47.60)	0
13. Rosenbrock Function (n = 1000) [70, 83]	5.8935E-19 (2.48E+01)	5.8935E-19 (5.33E+01)	0 (5.17E+01)	0 (4.61E+01)	0
14. Powell Function (n = 1000) [70, 83]	9.8784E-09 (3.50E+01)	1.0364E-07 (1.73E+02)	0 (2.03E+02)	0 (1.40E+02)	0
15. Quartic With Noise Function (n = 1000) [3]	5.0000E-01 (2.01E+01)	5.0000E-01 (5.61E+00)	5.0000E-01 (5.53E+00)	5.0000E-01 (1.13E+01)	5.0000E-01
16. Schubert Function (n = 1000) [3]	3.4943E+03 (3.22E+00) (failed)	-1.2031E+04 (1.86E+02)	-1.2031E+04 (1.87E+02)	-1.2031E+04 (4.12E+02)	-1.2031E+04
17. Raydan 1 Function (n = 1000) [2]	5.0050E+04 1.87E+00)	5.0050E+04 (6.05E-01)	5.0050E+04 (7.35E-01)	5.0050E+04 (3.95E-01)	\

Table 4: 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)	5.9607E-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 (2.00E+01)	5.1228E-18 (3.14E+01)	5.1228E-18 (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 (5.59E+00)	\
24. Extended BD1 Function (n = 1000) [2]	2.3356E-14 (3.14E+00)	3.3241E-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 (51.64E+01)	\
26. Perturbed quadratic diagonal Function (n = 1000) [2]	1.9102E-15 (1.10E+01)	2.5170E-15 (3.59E+02)	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 (8.76E+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+01) (failed)	-4.9575E+05 (1.67E+01) (failed)	-5.0050E+05 (2.04E+01)	-5.0050E+05 (3.05E+01)	\
31. Diagonal 5 Function	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	-5.0031E+02 (1.33E+01)	-5.0031E+02 (6.75E+01)	-5.0031E+02 (7.20E+01)	-5.0031E+02 (4.94E+01)	\
33. EG2 Function (n = 1000) [2]	-9.9884E+02 (9.06E+00)	-9.9850E+02 (7.42E+01)	-9.9894E+02 (1.14E+02)	-9.9950E+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 5: Numerical results of small-scale problems (no. 35-50) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

Problem	CNMTrM min $f(x)$ (CPU time (s))	CNMDTM min $f(x)$ (CPU time (s))	CNMGE min $f(x)$ (CPU time (s))	CNMGE_AG min $f(x)$ (CPU time (s))	Global minimum
35. Griewank Function (n = 10) [70,83]	1.0030E+00 (2.24E-01) (failed)	2.7755E-15 (1.64E+00)	0 (1.98E+00)	0 (1.95E-02)	0
36. Levy Function N. 13 (n = 2) [83]	9.7371E-01 (1.62E-02) (failed)	5.9152E-22 (3.11E-01)	1.3497E-31 (3.87E-01)	1.3497E-31 (2.18E-01)	0
37. Hosaki Function (n = 2) [3]	-1.1277E+00 (2.55E-02) (failed)	-2.7434E+15 (1.24E+00)	-2.7434E+15 (1.40E+00)	-1.2200E+07 (1.39E-01) (failed)	-∞
38. Beale Function (n = 2) [70,83]	2.4143E-16 (1.81E-02)	3.8283E-13 (1.04E-01)	4.5343E-14 (2.38E-01)	4.3287E-14 (7.00E-02)	0
39. Easom Function (n = 2) [83]	-1.3953E-07 (1.17E-02) (failed)	-9.70E-01 (1.93E+00)	-1.0000E+00 (1.97E+00)	-1.0000E+00 (8.44E-01)	-1.0000E+00
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)	0
41. Branin Function (n = 2) [83]	3.9788E-01 (1.40E-02)	3.9788E-01 (3.45E-02)	3.9788E-01 (4.42E-02)	3.9788E-01 (1.17E-02)	3.9788E-01
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)	0
43. Booth Function (n = 2) [83]	3.4010E-16 (9.12E-03)	1.3604E-15 (1.84E-02)	1.3604E-15 (3.58E-02)	1.3604E-15 (9.17E-03)	0
44. Matyas Function (n = 2) [83]	1.8783E-14 (1.01E-02)	1.8783E-14 (1.22E-02)	0 (3.61E-02)	0 (4.41E-03)	0
45. McCormick Function (n = 2) [83]	-1.9132E+00 (1.41E-02)	-1.9132E+00 (5.59E-01)	-1.9132E+00 (5.67E-01)	-1.9132E+00 (3.30E-01)	-1.9132E+00
46. Power Sum Function (n = 4) [83]	3.7740E-09 (1.93E-02)	4.2909E-04 (1.29E-01) (failed)	4.2909E-04 (1.86E-01) (failed)	2.1824E-11 (2.15E-01)	0
47. Colville Function (n = 4) [83]	7.7098E-18 (3.99E-02)	1.0131E-17 (8.98E-02)	0 (1.00E-01)	0 (1.73E-02)	0
48. Schaffer Function N. 2 (n = 2) [83]	9.9754E-01 (3.50E-02) (failed)	4.6240E-14 (2.66E+00)	0 (2.48E+00)	0 (6.04E-01)	0
49. Bohachevsky Function (n = 2) [83]	3.5183E+00 (1.85E-02) (failed)	4.12E-01 (3.80E-01) (failed)	0 (3.93E-01)	0 (1.16E-01)	0
50. Three-Hump Camel Function (n = 2) [83]	2.9863E+00 (2.40E-02) (failed)	1.1706E-14 (8.68E-02)	0 (9.51E-02)	0 (1.51E-02)	0

Table 6: Numerical results of small-scale problems (no. 51-68) computed by CNMGE, CNMTrM, CNMDTM and CNMGE_AG.

Problem	CNMTrM min $f(x)$ (CPU time (s))	CNMDTM min $f(x)$ (CPU time (s))	CNMGE min $f(x)$ (CPU time (s))	CNMGE_AG min $f(x)$ (CPU time (s))	Global minimum
51. Six-Hump Camel Function (n = 2) [83]	5.4371E-01 (3.06E-02) (failed)	-1.0316E+00 (8.12E-02)	-1.0316E+00 (9.84E-02)	-1.0316E+00 (5.39E-02)	-1.0316E+00
52. Drop-Wave Function (n = 2) [83]	-8.6245E-15 (3.50E-02) (failed)	-1.0000E+00 (4.16E+00)	-1.0000E+00 (3.88E+00)	-1.0000E+00 (8.18E-02)	-1.0000E+00
53. Perm Function 0, d, β (n = 4) [83]	0 (4.93E-02)	0 (3.68E-01)	0 (3.82E-01)	0 (1.94E-01)	0
54. Hartmann 3-D Function (n = 3) [83]	-2.8713E-08 (6.28E-02) (failed)	-3.8627E+00 (2.98E+00)	-3.8627E+00 (2.89E+00)	-3.8627E+00 (1.92E+00)	-3.8627E+00
55. Trefethen 4 Function (n = 2) [3]	-1.8895E+00 (2.57E-02) (failed)	-2.4275E+00 (6.40E-02) (failed)	-3.1407E+00 (8.16E-02)	-2.6634E+00 (4.28E-02)	-3.3068E+00
56. Zettl Function (n = 2) [3]	3.7912E-03 (1.65E-02)	3.7912E-03 (4.36E-02)	3.7912E-03 (5.82E-02)	3.7912E-03 (2.17E-02)	3.7912E-03
57. Exp2 Function (n = 2) [3]	1.9476E+00 (4.38E-02) (failed)	1.9476E+00 (1.19E-01) (failed)	3.6953E-14 (1.83E-01)	3.6953E-14 (4.45E-01)	0
58. Hansen Function	-2.6977E+01 (1.82E-02) (failed)	-1.7654E+02 (2.19E+00)	-1.7654E+02 (2.15E+00)	-1.7654E+02 (2.84E+00)	-1.7654E+02
59. Schaffer Function N. 4 (n = 2) [83]	9.9687E-01 (7.26E-02) (failed)	2.9257E-01 (3.20E+00)	2.9257E-01 (3.14E+00)	2.9257E-01 (7.76E-02)	2.9257E-01
60. Holder Table Function (n = 2) [83]	-1.7329E+00 (6.27E-02) (failed)	-∞ (1.20E+00)	-6.5535E+04 (1.21E+00) (failed)	-∞ (2.13E-01)	-∞
61. Gramacy & Lee (2012) Function (n = 1) [83]	4.7641E-01 (3.51E-02) (failed)	-8.6901E-01 (6.77E-01)	-8.6901E-01 (6.77E-01)	-8.6901E-01 (2.46E-01)	-8.6901E-01
62. Eggholder Function (n = 2) [3]	-9.3533E+02 (2.38E-02) (failed)	-2.50410E+03 (4.19E-01) (failed)	-3.7274E+03 (3.74E-01)	-2.0408E+03 (6.31E-02) (failed)	\
63. Michalewicz Function (n = 2) [83]	-2.0869E-08 (2.94E-02) (failed)	-8.0130E-01 (4.83E-01) (failed)	-1.3061E+00 (5.12E-01)	-1.8013E+00 (1.15E-01)	-1.8013E+00
64. Box-Betts Exponential Quadratic Function (n = 3) [3]	8.3869E-17 (1.97E-02)	8.3869E-17 (6.17E-02)	0 (7.09E-02)	0 (4.67E-01)	0
65. Cross-in-Tray Function (n = 2) [83]	-2.0626E+00 (2.30E-02)	-2.0626E+00 (2.25E+00)	-2.0626E+00 (2.13E+00)	-2.0626E+00 (9.39E-01)	-2.0626E+00
66. Himmeblau Function (n = 2) [83]	6.7719E+01 (8.03E-02) (failed)	6.4197E-14 (5.30E-02)	6.9206E-16 (6.80E-02)	6.9206E-16 (3.82E-02)	0
67. Forrester et al. (2008) Function (n = 1) [83]	1.5908E+01 (1.43E-02) (failed)	-1.1490E+05 (5.20E-01) (failed)	-9.1005E+06 (5.32E-01) (failed)	-1.8288E+07 (4.78E-01) (failed)	-∞
68. Goldstein-Price Function (n = 2) [83]	3 (3.19E-02)	3 (8.22E-02)	3 (6.69E-02)	3 (2.66E-02)	3

Table 7: Numerical results of large-scale problems (no.1-17) 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
1. Molecular Energy Problem (n = 1000) [50]	-4.1118E+01 (1.59E+01)	-2.8761E-01 (1.82E+01) (failed)	-4.1118E+01 (2.68E+04) (too long)	1.0008E+01 (2.98E+02) (failed)	-4.1118E+01
2. Ackley Function (n = 1000) [83]	8.8817E-016 (1.60E+02)	8.8817E-16 (1.81E+01)	6.7078E-06 (1.77E+03)	3.6253 (7.71E+01) (failed)	0
3. Levy Function (n = 1000) [83]	1.4997E-32 (3.67E+02)	0.3651 (3.74E+00) (failed)	1.1997E-15 (1.43E-01)	1.4998E-32 (1.51E+02)	0
4. Schwefel Function (n = 1000) [83]	1.2728E-02 (6.83E+01)	4.7903E+03 (4.34E+01) (failed)	3.1688E+03 (2.42E+04) (failed)	1.7096E+05 (2.45E+02) (failed)	1.2728E-02
5. Rastrigin Function (n = 1000) [83]	0 (4.33E+00)	0 (3.48E+00)	8.5893E-26 (2.82E+01)	1.0000E+03 (1.84E+02) (failed)	0
6. Styblinski-Tang Function (n = 1000) [83]	-3.9166E+04 (5.78E+01)	-2.5013e+04 (4.52E+00) (failed)	-3.9166E+04 (2.55E+04) (too long)	-8.1869E+03 (1.69E+02) (failed)	-3.9166E+04
7. Trid Function (n = 1000) [83]	-1.6717E+08 (2.84E+01)	-2.4920e+04 (4.34E+00) (failed)	-1.6716E+08 (2.64E+04) (too long)	-9.9900E+02 (1.84E+02) (failed)	-1.6716E+08
8. Sum Squares Function (n = 1000) [83]	0 (1.00E+01)	0 (3.84E+00)	3.7885E-44 (4.14E-01)	2.3517E-16 (6.20E+01)	0
9. Sphere Function (n = 1000) [83]	0 (2.23E-01)	3.9800E-11 (3.22E+00)	1.4569E-31 (2.66E-01)	2.9222E-16 (5.91E+01)	0
10. Rotated Hyper-Ellipsoid Function (n = 1000) [83]	0 (1.34E+01)	0 (1.54E+01)	2.1437E-45 (3.26E-01)	3.7091E-16 (6.64E+02)	0
11. Zakharov Function (n = 1000) [83]	0 (6.00E+00)	0 (5.17E+00)	1.3612E-14 (3.20E+00)	1.5541E+04 (1.61E+02) (failed)	0
12. Dixon-Price Function (n = 1000) [83]	6.2737E-14 (4.96E+01)	7.2931E-01 (4.03E+00) (failed)	1.0905E-15 (2.13E-01)	6.6667E-01 (9.77E+01) (failed)	0
13. Rosenbrock Function (n = 1000) [70, 83]	0 (5.17E+01)	3.0630E+01 (3.82E+00) (failed)	9.8899E+02 (1.07E+02) (failed)	9.7702E+02 (1.77E+02) (failed)	0
14. Powell Function (n = 1000) [70, 83]	0 (2.03E+02)	0 (4.72E+00)	5.6167E-10 (9.05E-01)	4.9855E-03 (2.63E+02) (failed)	0
15. Quartic With Noise Function (n = 1000) [3]	5.0000E-01 (5.53E+00)	5.0000E-01 (3.89E+00)	5.0000E-01 (9.90E-02)	5.0000E-01 (5.75E+01)	5.0000E-01
16. Schubert Function (n = 1000) [3]	-1.2031E+04 (1.87E+02)	-9.4947E+03 (6.30E+01) (failed)	-1.2031E+04 (2.62E+04) (too long)	-4.2396E+03 (1.66E+02) (failed)	-1.2031E+04
17. Raydan 1 Function (n = 1000) [2]	5.0050E+04 (7.35E-01)	5.0050E+04 (1.03E+01)	5.0050E+04 (1.15E-01)	5.0050E+04 (2.83E+02)	\

Table 8: Numerical results of large-scale problems (no.18-34) 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
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)	1.3083E+03 (2.93E+02) (failed)	\
19. Extended Tridiagonal 1 Function(n = 1000) [2]	5.9698E-19 (6.29E+01)	2.9700E+02 (4.51E+00) (failed)	9.9721E+02 (5.61E+02) (failed)	1.0000E+03 (2.66E+02) (failed)	\
20. Extended quadratic penalty	3.9900E+03 (1.15E+01)	3.9962E+03 (3.30E+00) (failed)	3.9900E+03 (2.58E+04) (too long)	3.9900E+03 (1.05E+02)	\
21. Extended quadratic penalty QP2 Function (n = 1000) [2]	5.1228E-18 (4.75E+01)	1.0000E+04 (3.66E+00) (failed)	3.6937E-22 (2.48E+00)	1.3031E-01 (6.64E+01) (failed)	\
22. Quadratic QF2 Function (n = 1000) [2]	-1.0001E+00 (1.44E+01)	-1.0001E+00 (3.66E+00)	-1.0001E+00 (4.10E+01)	-1.0001E+00 (5.25E+01)	\
23. Extended PSC1 Function (n = 1000) [2]	3.8660E+02 (6.83E+00)	5.0000E+02 (3.32E+00) (failed)	3.8660E+02 (2.82E+04) (too long)	3.8660E+02 (9.50E+01)	\
24. Extended BD1 Function (n = 1000) [2]	0 (3.98E+01)	0 (2.50E+00)	5.0327E-23 (3.49E-01)	1.0616E-15 (9.98E+01)	\
25. Extended Cliff Function (n = 1000) [2]	9.9893E+01 (7.66E+01)	2.3779E+02 (5.44E+00) (failed)	9.9893E+01 (1.85E+01)	1.0179E+02 (2.44E+02)	\
26. Perturbed quadratic diagonal Function (n = 1000) [2]	0 (3.92E+02)	0 (3.17E+00)	1.1750E-42 (1.23E+00)	1.3102E-14 (8.59E+01)	\
27. Extended Hiebert Function (n = 1000) [2]	5.4874E-19 (8.85E+01)	1.2500E+12 (2.21E+01) (failed)	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+01)	1.2796E+03 (3.65E+00)	1.2796E+03 (2.67E+04) (too long)	1.2796E+03 (1.75E+02)	\
29. Diagonal 1 Function (n = 1000) [2]	-2.7068E+06 (1.06E+00)	-2.2288E+06 (3.54E+00) (failed)	-2.7068E+06 (5.33E+01)	-2.7068E+06 (2.52E+02)	\
30. Diagonal 3 Function (n = 1000) [2]	-5.0050E+05 (2.04E+01)	-4.8990E+05 (1.83E+01) (failed)	-5.0050E+05 (7.34E-01)	-4.8991E+05 (2.17E+02) (failed)	\
31. Diagonal 5 Function (n = 1000) [2]	6.9315E+02 (6.20E-01)	6.9315E+02 (1.16E+01)	6.9315E+02 (2.81E+04) (too long)	6.9315E+02 (1.65E+02)	\
32. Extended Maratos Function (n = 1000) [2]	-5.0031E+02 (7.20E+01)	-5.0026E+02 (3.63E+00)	-5.0031E+02 (1.60E+01)	-1.5082E+02 (1.74E+02) (failed)	\
33. EG2 Function (n = 1000) [2]	-9.9950E+02 (1.14E+02)	-9.2965E+02 (2.71E+00) (failed)	-9.9950E+02 (5.76E-01)	-8.0038E+02 (6.47E+01) (failed)	\
34. SINQUAD Function (n = 1000) [2]	0 (3.57E+01)	0 (3.30E+00)	2.4405E-13 (1.12E+00)	0 (2.05E+02)	\

Table 9: Numerical results of small-scale problems (no.35-51) computed by CN-MGE, 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
35. Griewank Function (n = 10) [70,83]	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) [83]	1.3497E-31 (3.87E-01)	1.0981E-01 (2.85E+00) (failed)	2.8774E-16 (8.48E-03)	1.3498E-31 (6.78E-01)	0
37. Hosaki Function (n = 2) [3]	-inf (1.40E+00)	-1.1277E+00 (2.62E+00) (failed)	-2.3458E+00 (2.80E+04) (too long)	-2.3458E+00 (5.70E-01)	-2.3458E+00
38. Beale Function (n = 2) [70,83]	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) [83]	-1.0000E+00 (1.97E+00)	-1.0000E+00 (2.64E+00)	-1.0000E+00 (2.55E-03)	-3.0308E-05 (3.71E-02) (failed)	-1.0000E+00
40. Price Function (n = 2) [3]	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) [83]	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) [3]	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) [83]	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) [83]	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) [83]	-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) [83]	4.2909E-04 (1.86E-01) (failed)	2.0472E-07 (4.19E+00)	4.2680E-22 (4.76E-03)	5.4319E-12 (4.46E+00)	0
47. Colville Function (n = 4) [83]	0 (1.00E-01)	5.3436E-13 (3.80E+00)	0 (2.39E-01)	0 (1.09E+00)	0
48. Schaffer Function N. 2 (n = 2) [83]	0 (2.48E+00)	0 (4.17E+00)	0 (1.10E-03)	2.9258E-01 (1.32E+00)	0
49. Bohachevsky Function (n = 2) [83]	0 (3.93E-01)	0 (2.43E+00)	-5.5511E-17 (2.43E-02)	0 (6.54E-01)	0
50. Three-Hump Camel Function (n = 2) [83]	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) [83]	-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 10: 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) [83]	-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) [83]	0 (3.82E-01)	6.3025E+02 (5.79E+00) (failed)	5.3868E-10 (2.10E-01)	3.0872E-16 (1.11E+00)	0
54. Hartmann 3-D Function (n = 3) [83]	-3.8627E+00 (2.89E+00)	-3.8628E+00 (2.77E+00)	-3.8628E+00 (4.50E-01)	-3.8628E+00 (8.20E-01)	-3.8628E+00
55. Trefethen 4 Function (n = 2) [3]	-3.1407E+00 (8.16E-02)	-2.8376E+00 (5.91E+00)	-3.3069E+00 (7.88E-01)	-3.2965E+00 (8.23E-01)	-3.3069E+00
56. Zettl Function (n = 2) [3]	-3.7912E-03 (2.82E-02)	-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]	3.6945E-14 (1.83E-01)	4.8793E-14 (2.81E+00)	8.0737E-20 (7.61E-03)	3.0512E-18 (6.26E-01)	0
58. Hansen Function (n = 2) [3]	-1.7654E+02 (2.15E+00)	-1.7654E+02 (2.98E+00)	-1.7654E+02 (9.89E-01)	-1.7654E+02 (8.64E-01)	-1.7654E+02
59. Schaffer Function N. 4 (n = 2) [83]	2.9257E-01 (3.14E+00)	2.9258E-01 (4.60E+00)	2.9258E-01 (1.49E-01)	2.9387E-01 (9.21E-01)	2.9258E-01
60. Holder Table Function (n = 2) [83]	-∞ (1.21E+00)	-1.9209E+01 (3.36E+00) (failed)	-1.9209E+01 (3.81E-01) (fail)	-1.9209E+01 (6.05E-01) (fail)	-∞
61. Gramacy & Lee (2012) Function (n = 1) [83]	-8.6901E-01 (6.77E-01)	-8.6901E-01 (5.49E-01)	-8.6901E-01 (2.31E-03)	-8.6901E-01 (5.67E-01)	-8.6901E-01
62. Eggholder Function (n = 2) [3]	-3.7274E+03 (3.74E-01) (failed)	-9.3533E+02 (3.53E+00) (failed)	-9.5964E+02 (2.98E-02) (failed)	-9.3432E+02 (8.48E-01) (failed)	-∞
63. Michalewicz Function (n = 2) [83]	-1.3061E+00 (5.12E-01)	-1.8013E+00 (4.32E+00)	-1.8013E+00 (3.62E-01)	-1.8013E+00 (5.97E-01)	-1.8013E+00
64. Box-Betts Exponential Quadratic Function (n = 3) [3]	0 (7.09E-02)	6.8000E-17 (2.76E+00)	6.3148E-16 (6.52E-03)	1.1671E-18 (8.48E-01)	0
65. Cross-in-Tray Function (n = 2) [83]	-2.0626E+00 (2.13E+00)	-2.0626E+00 (3.88E+00)	-2.0626E+00 (7.46E-03)	-2.0626E+00 (5.69E-01)	-2.0626E+00
66. Himmeblau Function (n = 2) [83]	6.9206E-16 (6.80E-02)	0 (2.43E+00)	2.2376E-25 (2.32E-03)	1.0669E-18 (5.95E-01)	0
67. Forrester et al. (2008) Function (n = 1) [83]	-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
68. Goldstein-Price Function (n = 2) [83]	3.0000E+00 (4.73E-01)	3.0000E+00 (2.71E+00)	3.0000E+00 (3.61E+00)	3.0000E+00 (7.32E-01)	3.0000E+00

Table 11: Numerical results of large-scale problems (no.1-17) computed by CNMGE, MCS, GLODS and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
1. Molecular Energy Problem (n = 1000) [50]	-4.1118E+01 (1.59E+01)	-3.6903E-01 (2.01E+02) (failed)	6.2332E-04 (1.21E+02) (failed)	-2.8789E-01 (1.51E+01) (failed)	-4.1118E+01
2. Ackley Function (n = 1000) [83]	8.8817E-16 (1.60E+02)	0 (1.28E+04) (too long)	8.8817E-16 (1.14E+02)	4.8252E-07 (4.81E+00)	0
3. Levy Function (n = 1000) [83]	1.4997E-32 (3.67E+02)	2.2419E-09 (1.10E+02)	8.9970E+01 (1.48E+02) (failed)	7.3494E-12 (1.43E+01)	0
4. Schwefel Function (n = 1000) [83]	1.2728E-02 (6.83E+01)	1.1844E+05 (2.60E+02) (failed)	2.9976E-01 (1.35E+02) (failed)	2.1713E+05 (1.36E+01) (failed)	1.2728E-02
5. Rastrigin Function (n = 1000) [83]	0 (4.33E+00)	0 (1.25E+02)	0 (1.14E+02)	2.8867E-09 (4.26E+00)	0
6. Styblinski-Tang Function (n = 1000) [83]	-3.9166E+04 (5.78E+01)	-3.9166E+04 (3.17E+02)	-9.4000E+02 (1.27E+02) (failed)	-2.5029E+04 (3.82E+01) (failed)	-3.9166E+04
7. Trid Function (n = 1000) [83]	-1.6717E+08 (2.84E+01)	-9.9900e+02 (1.84E+02) (failed)	-9.6100E+02 (1.53E+02) (failed)	-2.3258E+07 (3.81E+00) (failed)	-1.6716E+08
8. Sum Squares Function (n = 1000) [83]	0 (1.00E+01)	0 (1.26E+04) (too long)	0 (1.13E+02)	5.9371E-09 (3.59E+00)	0
9. Sphere Function (n = 1000) [83]	0 (2.23E-01)	0 (1.26E+04) (too long)	0 (1.13E+02)	1.4551E-09 (3.27E+00)	0
10. Rotated Hyper-Ellipsoid Function (n = 1000) [83]	0 (1.34E+01)	0 (1.43E+04) (too long)	0 (1.30E+02)	8.6292E-09 (1.71E+02)	0
11. Zakharov Function (n = 1000) [83]	0 (6.00E+00)	1.7016E+04 (1.44E+02) (failed)	0 (1.13E+02)	8.4247E-03 (3.60E+00) (failed)	0
12. Dixon-Price Function (n = 1000) [83]	6.2737E-14 (4.96E+01)	6.6667E-01 (8.45E+02) (failed)	6.6796E-01 (1.19E+02) (failed)	9.4552E+01 (9.98E-01) (failed)	0
13. Rosenbrock Function (n = 1000) [70, 83]	0 (5.17E+01)	2.1796E+02 (2.18E+02) (failed)	9.9815E+02 (1.30E+02) (failed)	7.2905E-09 (4.16E+00)	0
14. Powell Function (n = 1000) [70, 83]	0 (2.03E+02)	2.1627E+01 (2.05E+02) (failed)	0 (1.22E-01)	2.1787E-04 (1.95E+01) (failed)	0
15. Quartic With Noise Function (n = 1000) [3]	5.0000E-01 (5.53E+00)	5.0000E-01 (1.27E+04) (too long)	5.0000E-01 (1.14E+02)	5.0000E-01 (5.77E+00)	5.0000E-01
16. Schubert Function (n = 1000) [3]	-1.2031E+04 (1.87E+02)	-2.6399E+03 (3.34E+02) (failed)	-1.2031E+04 (8.91E+01)	-1.0629E+04 (3.93E+01) (failed)	-1.2031E+04
17. Raydan 1 Function (n = 1000) [2]	5.0050E+04 (7.35E-01)	5.0050E+04 (1.22E+04) (too long)	5.0050E+04 (8.91E+01)	5.0050E+04 (3.93E+01)	\

Table 12: Numerical results of large-scale problems (no.18-34) computed by CNMGE, MCS, GLODS and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
18. Raydan 2 Function (n = 1000) [2]	1.0000E+03 (2.15E+00)	1.0000E+03 (1.31E+04) (too long)	1.0000E+03 (5.16E+01)	1.3083E+03 (5.08E+00)	\
19. Extended Tridiagonal 1 Function(n = 1000) [2]	5.9698E-19 (6.29E+01)	4.0306E-01 (1.59E+02) (failed)	9.8000E+02 (1.47E+02) (failed)	9.0949E-13 (4.76E+00)	\
20. Extended quadratic penalty QP1 Function (n = 1000) [2]	3.9900E+03 (1.15E+01)	3.9912E+03 (7.88+01)	3.9991E+03 (1.30E+02)	3.9900E+03 (3.36E+00)	\
21. Extended quadratic penalty QP2 Function (n = 1000) [2]	5.1228E-18 (4.75E+01)	6.6117E+02 (6.79E+01) (failed)	3.74E+02 (1.44E+02) (failed)	3.66E-02 (8.23E+00) (failed)	\
22. Quadratic QF2 Function (n = 1000) [2]	-1.0001E+00 (1.44E+01)	-1.0001E+00 (7.79E+01)	2.5022E+05 (1.53E+02) (failed)	-1.0001E+00 (3.69E+00)	\
23. Extended PSC1 Function (n = 1000) [2]	3.8660E+02 (6.83E+00)	4.0143E+02 (9.22E+01) (failed)	4.9849E+02 (1.49E+02) (failed)	3.8660E+02 (5.50E+00)	\
24. Extended BD1 Function (n = 1000) [2]	0 (3.98E+01)	1.2762E-23 (9.52E+01)	2.0637E+03 (6.67E+01) (failed)	7.2759E-15 (4.05E+00)	\
25. Extended Cliff Function (n = 1000) [2]	9.8933E+01 (7.66E+01)	2.9165E+02 (7.05E+01) (failed)	4.9595E+02 (1.67E+02) (failed)	9.9906E+01 (7.88E+00)	\
26. Perturbed quadratic diagonal Function (n = 1000) [2]	0 (3.92E+02)	0 (1.27E+04) (too long)	0 (1.36E+02)	1.1973E+03 (3.38E+00) (failed)	\
27. Extended Hiebert Function (n = 1000) [2]	5.4874E-19 (8.85E+01)	6.5817E+06 (1.41E+02) (failed)	1.2475E+12 (1.81E+02) (failed)	2.5105E+08 (3.34E+00) (failed)	\
28. Extended TET Function (n = 1000) [2]	1.2796E+03 (3.58E+01)	1.2796E+03 (2.17E+02)	\ (0) (failed)	1.2796E+03 (7.51E+00)	\
29. Diagonal 1 Function (n = 1000) [2]	-2.7068E+06 (1.06E+00)	-2.6997E+06 (7.52E+01) (failed)	7.3936E+02 (7.52E+01) (failed)	-2.7068E+06 (2.52E+02)	\
30. Diagonal 3 Function (n = 1000) [2]	-5.0050E+05 (2.04E+01)	-4.9623E+05 (2.79E+02) (failed)	-5.0049E+05 (5.93E+01)	-4.9575E+05 (1.18E+01) (failed)	\
31. Diagonal 5 Function (n = 1000) [2]	6.9315E+02 (6.20E-01)	6.9315E+02 (1.25E+04) (too long)	\ (0) (failed)	6.9315E+02 (7.98E+00)	\
32. Extended Maratos Function (n = 1000) [2]	-5.0031E+02 (7.20E+01)	-5.0026E+02 (8.24E+01)	4.9596E+04 (1.45E+02) (failed)	-4.9736E+02 (3.43E+00)	\
33. EG2 Function (n = 1000) [2]	-9.9950E+02 (1.14E+02)	-9.9917E+02 (1.01E+02)	-9.9946E+02 (1.29E+02)	-9.9949E+02 (1.50E+01)	\
34. SINQUAD Function (n = 1000) [2]	0 (3.57E+01)	5.5349E-01 (1.70E+02) (failed)	7.5168E-01 (1.30E+02) (failed)	8.1854E-12 (3.98E+00)	\

Table 13: Numerical results of small-scale problems (no.35-51) computed by CNMGE, MCS, GLODS and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
35. Griewank Function (n = 10) [70,83]	1.9318E-14 (1.98E+00)	0 (6.38E-01)	0 (9.01E+00)	3.2362E-09 (2.63E-01)	0
36. Levy Function N. 13 (n = 2) [83]	1.3497E-31 (3.87E-01)	1.3498E-31 (7.01E-01)	1.3497E-31 (5.80E+01)	5.4136E-06 (3.74E-02)	0
37. Hosaki Function (n = 2) [3]	-∞ (1.40E+00)	-2.3457E+00 (3.26E-01) (failed)	-2.3458E+00 (4.89E-01) (failed)	-1.0000E+50 (2.36E-02)	-∞
38. Beale Function (n = 2) [70,83]	4.5343E-14 (2.38E-01)	1.5473E-22 (2.82E-01)	2.0990E-15 (5.82E+02)	4.8504E-07 (4.92E-01)	0
39. Easom Function (n = 2) [83]	-1.0000E+00 (1.97E+00)	-1.0000E+00 (6.13E-01)	-8.1102E-05 (5.25E+02) (failed)	-9.9991E-01 (3.71E-02)	-1.0000E+00
40. Price Function (n = 2) [3]	0 (3.76E-02)	0 (1.42E-01)	0 (5.48E+01)	2.9208E-06 (2.91E-02)	0
41. Branin Function (n = 2) [83]	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
42. Trecanni Function (n = 2) [3]	0 (2.66E-02)	0 (4.46E-01)	0 (5.36E+01)	2.9796E-07 (1.79E-02)	0
43. Booth Function (n = 2) [83]	1.3604E-15 (3.85E-02)	0 (2.48E-01)	4.4408E-16 (5.25E+01)	9.5258E-06 (1.85E-01)	0
44. Matyas Function (n = 2) [83]	0 (3.61E-02)	0 (9.18E-02)	0 (5.23E+01)	4.0101E-10 (3.77E-02)	0
45. McCormick Function (n = 2) [83]	-1.9132E+00 (5.67E-01)	-1.9132E+00 (2.30E-01)	-1.9132E+00 (7.99E-01)	1.2283E+00 (1.87E-02) (failed)	-1.9132E+00
46. Power Sum Function (n = 4) [83]	4.2909E-04 (1.86E-01)	1.4325E-06 (1.07E+00)	0 (2.24E+01)	2.4072E-09 (3.31E-02)	0
47. Colville Function (n = 4) [83]	0 (1.00E-01)	3.8343E-19 (5.95E-01)	1.4701E-05 (2.00E+01)	1.4566E-09 (2.99E-02)	0
48. Schaffer Function N. 2 (n = 2) [83]	0 (5.67E-01)	0 (6.62E-01)	0 (5.37E+01)	5.0000E-01 (2.03E-02) (failed)	0
49. Bohachevsky Function (n = 2) [83]	0 (3.93E-01)	0 (1.66E-01)	-5.5511E-17 (5.39E+01)	2.8554E-06 (2.13E-02)	0
50. Three-Hump Camel Function (n = 2) [83]	0 (9.51E-02)	0 (1.15E-01)	0 (5.37E+01)	2.3841E-07 (2.26E-02)	0
51. Six-Hump Camel Function (n = 2) [83]	-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 14: Numerical results of small-scale problems (no.52-68) computed by CNMGE, MCS, GLODS and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global Minimum
52. Drop-Wave Function (n = 2) [83]	-1.0000E+00 (3.88E+00)	-1.0000E+00 (5.65E-01)	-1.0000E+00 (5.32E+02)	-7.8506E-01 (1.74E-01)	-1.0000E+00
53. Perm Function 0, d, β (n = 4) [83]	0 (3.82E-01)	1.0130E+02 (9.28E-01) (failed)	1.3286E+00 (2.16E+01) (failed)	7.3526E+02 (4.43E-02) (failed)	0
54. Hartmann 3-D Function (n = 3) [83]	-3.8627E+00 (2.89E+00)	-3.8628E+00 (8.20E-01)	-3.8628E+00 (3.23E+01)	-3.0897E+00 (3.76E-02)	-3.8628E+00
55. Trefethen 4 Function (n = 2) [3]	-3.1407E+00 (8.16E-02)	-2.7612E+00 (7.37E-01)	-2.4279E+00 (5.38E+01)	-3.0626E+00 (1.82E-01)	-3.3069E+00
56. Zettl Function (n = 2) [3]	-3.7912E-03 (2.82E-02)	-3.7912E-03 (2.57E-01)	-3.7912E-03 (5.39E+02)	-3.7870E-03 (1.84E-01)	-3.7912E-03
57. Exp2 Function (n = 2) [3]	3.6945E-14 (1.83E-01)	2.8282E-21 (2.44E-01)	2.6936E+02 (5.26E+01) (failed)	2.6936E+02 (2.07E-02) (failed)	0
58. Hansen Function (n = 2) [3]	-1.7654E+02 (2.15E+00)	1.4548E+02 (7.22E-01) (failed)	-1.16E+02 (5.33E+01) (failed)	-2.43E+01 (2.44E-02) (failed)	-1.7654E+02
59. Schaffer Function N. 4 (n = 2) [83]	2.9257E-01 (3.14E+00)	4.9289E-01 (9.21E-01)	5.0000E-01 (5.22E+01)	5.0000E-01 (4.63E-02)	2.9258E-01
60. Holder Table Function (n = 2) [83]	-∞ (3.00E+00)	-1.9209E+01 (5.86E-01) (failed)	-1.9209E+01 (1.92E+00) (failed)	-1.0000E+50 (2.17E-02)	-∞
61. Gramacy & Lee (2012) Function (n = 1) [83]	-8.6901E-01 (6.77E-01)	-6.6328E-01 (4.56E-01)	-6.6327E-01 (4.27E-01)	-5.2660E-01 (2.57E-02)	-8.6901E-01
62. Eggholder Function (n = 2) [3]	-3.7274E+03 (3.74E-01) (failed)	-5.5737E+02 (4.70E-01) (failed)	-1.2893E+32 (5.26E+01)	-2.4663E+37 (2.35E-02)	-∞
63. Michalewicz Function (n = 2) [83]	-1.3061E+00 (5.12E-01)	-1.8013E+00 (3.11E-01)	-1.8013E+00 (5.29E+01)	-1.8013E+00 (2.27E-02)	-1.8013E+00
64. Box-Betts Exponential Quadratic Function (n = 3) [3]	0 (7.09E-02)	0 (1.59E-01)	0 (3.28E+01)	1.9491E-07 (3.01E-02)	0
65. Cross-in-Tray Function (n = 2) [83]	-2.0626E+00 (2.13E+00)	-1.8899E+00 (3.40E-01)	-2.0626E+00 (1.32E+00)	-2.0626E+00 (2.31E-02)	-2.0626E+00
66. Himmeblau Function (n = 2) [83]	6.9206E-16 (6.80E-02)	3.4597E-20 (4.00E+00)	0 (5.43E+01)	5.6383E-04 (2.21E-02) (failed)	0
67. Forrester et al. (2008) Function (n = 1) [83]	-9.1005E+06 (5.32E-01)	-6.0207E+00 (2.65E-01)	-6.0207E+00 (3.00E-01)	-8.4445E+03 (2.04E-02)	-6.0207E+00
68. Goldstein-Price Function (n = 2) [83]	3.0000E+00 (4.73E-01)	3.0000E+00 (7.32E-01)	3.0000E+00 (1.86E+00)	3.0000E+00 (2.69E-02)	3.0000E+00

Table 15: Numerical results of CUTEst problems [35] (no. 69-84) 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
69. Allinitu (n = 4)	5.7444 (4.50E-01)	5.7444 (2.19E+00)	5.7444 (4.36E-03)	5.7444 (2.07E+00)	\
70. cliff (n = 2)	1.9979E-01 (2.63E+00)	1.9979E-01 (3.78E-01)	2.0724E-01 (4.72E-03)	1.9992E-01 (1.66E+00)	\
71. dixon3dq (n = 10)	0 (1.17E-01)	4.8186E-14 (1.81E-01)	2.2326E-32 (2.28E-03)	3.6187E-16 (2.01E+00)	\
72. edensch (n = 2000)	1.2003E+04 (8.66E+02)	1.2003E+04 (2.49E+00)	1.2003E+04 (2.67E+04) (too long)	1.2003E+04 (2.10E+02)	\
73. fletcher (n = 100)	0 (1.66E+00)	0 (4.04E-01)	5.2005E-18 (2.80E-02)	1.0922E-14 (3.70E+01)	\
74. genrose (n = 500)	1 (2.14E+01)	1 (6.45E-01)	1 (1.64E+00)	3.0296E+02 (2.26E+03) (failed)	\
75. hairy (n = 2)	2.0000E+01 (1.16E+01)	2.0000E+01 (3.36E-01)	2.0000E+01 (1.19E-02)	2.0000E+01 (1.23E+00)	\
76. himmelbb (n = 2)	0 (2.98E-01)	0 (1.48E-01)	2.1369E-21 (2.77E-03)	2.2419E-18 (7.07E-01)	\
77. himmelbg (n = 2)	0 (5.00E-01)	0 (3.16E-01)	3.6330E-22 (1.36E-03)	1.7970E-19 (1.24E+00)	\
78. indef (n = 1000)	-2.0924E+07 (7.14E+01)	-1.3857E+07 (1.71E+01)	-4.6265E+18 (2.78E+04) (too long)	-1.0005E+06 (1.29E+02)	\
79. jensmp (n = 2)	1.2436E+02 (3.51E-01)	2.0200E+03 (2.91E-01) (failed)	1.2436E+02 (3.60E-01)	1.2436E+02 (1.68E+00)	\
80. liarwhd (n = 1000)	0 (6.72E+01)	0 (8.94E-01)	5.9184E-17 (1.37E+02)	4.9470E-12 (6.71E+01)	\
81. loghairy (n = 2)	1.8232E-01 (3.02E-01)	1.8232E-01 (1.50E+00)	1.8232E-01 (4.42E-01)	1.8232E-01 (1.45E+00)	\
82. maratosb (n = 2)	-1 (1.44E-01)	-1 (1.15E+00)	-1 (3.60E-02)	-1 (4.90E+00)	\
83. mexhat (n = 2)	-4.0100E-02 (1.81E-01)	-4.0100E-02 (1.85E-01)	-4.0100E-02 (3.26E-02)	-4.0100E-02 (1.55E+00)	\
84. nondia (n = 1000)	0 (2.33E+01)	0 (1.05E+00)	9.6962E-16 (2.33E-01)	1.0406E+00 (9.03E+01) (failed)	\

Table 16: Numerical results of CUTEst problems [35] (no.85-100) 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
85. nondquar (n = 1000)	0.0000E+00 (1.04E+02)	4.8650E+03 (2.32E+00) (failed)	4.1323E-11 (2.89E-01)	8.4830E+04 (1.83E+02) (failed)	\
86. penalty1 (n = 1000)	2.5000E-03 (6.63E+01)	6.2500E+04 (1.12E+00) (failed)	6.4395E+00 (3.75E+01) (failed)	1.2410E-02 (9.04E+01) (failed)	\
87. power (n = 1000)	0.0000E+00 (8.30E+00)	1.3910E+08 (1.24E+00) (failed)	1.4823E-43 (4.33E-02)	8.8121E-14 (4.01E+01)	\
88. arglinb (n = 10)	4.6341E+00 (1.89E+00)	4.6341E+00 (1.86E-01)	4.6341E+00 (1.08E-02)	4.6341E+00 (2.73E+00)	\
89. arglinc (n = 10)	5.1351E+00 (1.72E+00)	7.1351E+00 (2.62E-01)	6.1351E+00 (1.53E-02)	7.1351E+00 (2.54E+00)	\
90. arwhead (n = 1000)	0.0000E+00 (8.40E+01)	1.3910E+08 (1.04E+00) (failed)	-2.6645E-15 (9.05E-01)	5.2295E-12 (3.44E+01)	\
91. bard (n = 3)	8.2149E-03 (2.61E-01)	8.2149E-03 (5.00E+00)	8.2149E-03 (2.89E+04) (too long)	8.2149E-03 (1.53E+00)	\
92. bdexp (n = 1000)	-∞ (1.33E+02)	8.6778E-01 (5.47E-01) (failed)	2.6452E-07 (2.71E+03) (failed)	-1.6337E+99 (1.36E+02)	\
93. bdqrtic (n = 1000)	3.9838E+03 (9.47E+01)	1.7996E+05 (1.07E+00) (failed)	3.9838E+03 (3.67E+02)	3.9838E+03 (3.56E+01)	\
94. biggs6 (n = 6)	0.0000E+00 (5.06E-01)	8.7426E-17 (5.07E+00)	5.9850E-21 (5.92E-03)	1.2583E-17 (1.99E+00)	\
95. box3 (n = 3)	5.2994E-12 (5.18E-02)	1.0813E-17 (7.74E-01)	1.6902E-25 (7.43E-03)	2.3991E-16 (1.52E+00)	\
96. brkmcc (n = 2)	1.6904E-01 (4.30E-01) (failed)	1.6904E-01 (1.40E-01) (failed)	1.6904E-01 (9.34E-02) (failed)	1.6904E-01 (1.10E+00) (failed)	$-\infty$
97. brownal (n = 10)	0.0000E+00 (1.16E+00)	4.4894E-15 (1.28E-01)	1.4956E-16 (3.99E-03)	2.5739E-16 (2.95E+00)	\
98. brownbs (n = 2)	0.0000E+00 (1.83E-01)	1.1085E-04 (1.39E-01)	0.0000E+00 (5.03E-03)	9.9999E+11 (2.25E+00) (failed)	\
99. brownden (n = 4)	8.5822E+04 (1.67E-01)	8.5822E+04 (1.69E-01)	8.5822E+04 (1.74E-01)	1.9995E+06 (2.77E+00) (failed)	\
100. broydn7d (n = 1000)	1.2333E+00 (4.01E+02) (failed)	2.3090E+03 (4.26E+00) (failed)	-1.0000E+50 (2.77E+04) (too long)	3.0375E+02 (2.03E+02) (failed)	\

Table 17: Numerical results of CUTEst problems [35] (no.101-116) 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
101. chainwoo (n = 1000)	1.0000E+00 (9.19E+01)	1.0000E+00 (3.26E+00)	8.1471E+00 (3.35E-01) (failed)	2.1209E+03 (1.11E+02) (failed)	\
102. chnrosnb (n = 50)	0 (2.29E+00)	0 (8.04E-01)	1.2858E-16 (8.12E-03)	3.5330E-15 (9.41E+00)	\
103. cosine (n = 1000)	-9.9900E+02 (6.0439E+01)	-9.7179E+02 (7.6399E+00)	-9.9900E+02 (2.19E-02)	-8.4689E+02 (8.81E+01) (failed)	\
104. cragglvy (n = 1000)	3.3642E+02 (2.45E+02)	6.1552E+02 (2.04E+00) (failed)	3.3642E+02 (2.74E+04) (too long)	5.4802E+05 (1.30E-01) (failed)	\
105. cube (n = 2)	0 (4.55E-02)	0 (3.33E-01)	2.1257E-19 (8.64E-03)	3.2470E-16 (1.50E+00)	\
106. curly10 (n = 1000)	failed (failed)	-1.0013E+08 (2.54E+03)	-1.0032E+05 (6.96E+02)	-9.9550E+06 (4.14E+03)	\
107. dqdrtic (n = 1000)	0 (2.07E+01)	9.6144E+00 (3.34E+00) (failed)	2.4552E-50 (4.45E-02)	1.0137E-13 (4.52E+01)	\
108. dqrtic (n = 1000)	0 (8.76E+01)	1.9498E+14 (1.0509E+01) (failed)	8.2828E+00 (1.3589E-01) (failed)	1.4700E-18 (4.12E+01)	\
109. eg2 (n = 1000)	-9.9895E+02 (1.23E+01)	-9.9013E+02 (1.58E+01)	-9.9895E+02 (1.03E+03)	1.3589E-01 (4.54E+01) (failed)	\
110. engval1 (n = 1000)	1.1082E+03 (7.48E+01)	1.1570E+03 (1.11E+00) (failed)	1.1082E+03 (1.06E+02)	1.1082E+03 (2.90E+01)	\
111. engval2 (n = 3)	0 (1.00E+00)	4.6149E-11 (3.62E-01)	1.0896E-23 (3.00E-03)	2.8460E-16 (2.18E+00)	\
112. errinros (n = 50)	3.9904E+01 (5.74E+00)	4.0666E+01 (5.46E-01)	3.9904E+01 (8.79E-01)	3.9904E+01 (6.08E+00)	\
113. extrosnb (n = 10)	0 (3.89E-01)	0 (5.85E-01)	2.1451E-13 (9.35E-01)	0 (7.67E+00)	\
114. fletcbv3 (n = 1000)	-4.5167E-02 (3.02E+01) (failed)	-5.7006E-06 (1.45E+01) (failed)	-6.7083E+04 (2.3639E+04) (too long)	-1.0013E-04 (9.39E+01) (failed)	$-\infty$
115. genhumps (n = 5)	0 (4.78E-01)	1.0775E-12 (2.42E-01)	7.0857E-24 (2.95E-03)	2.0008E-16 (1.71E+00)	\
116. gulf (n = 3)	failed (failed) (failed)	3.2835E+01 (3.81E-01) (failed)	4.0067E-28 (1.71E-01)	3.0959E+01 (3.09E-01) (failed)	\

Table 18: Numerical results of CUTEst problems [35] (no.117-132) 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
117. hilbertb (n = 50)	0 (4.14E-01)	2.8783E-13 (3.55E-01)	3.0878E-29 (2.82E+04) (too long)	6.3562E-16 (3.89E+00)	\
118. himmelbh (n = 2)	-1.0000E+10 (1.75E-01)	-2.6987E+25 (2.08E+00)	-1.0000E+00 (2.81E-01) (failed)	-1.0000E+00 (1.19E+00) (failed)	\
119. kowosb (n = 4)	1.1037E-03 (1.42E+00)	3.0751E-04 (2.15E+00)	3.0751E-04 (2.88E+04) (too long)	3.0751E-04 (1.94E+00)	\
120. meyer3 (n = 3)	3.6710E+06 1.74E+01) (failed)	7.6672E+02 (6.47E-01) (failed)	8.7946e+01 (2.89E+04) (too long)	3.8895E+09 (1.60E+00) (failed)	\
121.noncvxu2 (n = 1000)	2.3168E+03 (1.57E+02)	3.3265E+03 (1.52E+00) (failed)	2.3169E+03 (2.86E+04) (too long)	9.1465E+04 (1.87E+02) (failed)	\
122.osbornea (n = 5)	5.4650E-05 (8.08E-01)	5.0572E-02 (1.85E+00) (failed)	5.4649E-05 (2.84E+04) (too long)	5.4649E-05 (3.06E+00)	\
123. penalty2 (n = 100)	9.7096E+04 (3.17E+00)	9.7119E+04 (5.21E-01)	9.7096E+04 (2.84E+04) (too long)	9.7096E+04 (1.34E+01)	\
124. quartc (n = 1000)	0 (7.61E+01)	1.9498E+14 (8.57E+00) (failed)	8.2828E+00 (8.94E-02) (failed)	2.0887E-18 (4.31E+01)	\
125. rosenbr (n = 1000)	0 (1.24E-01)	0 (1.54E+00)	1.9165E-22 (2.60E-03)	2.1342E-15 (2.59E+00)	\
126. scosine (n = 1000)	-8.3699E+01 (1.40E+02) (failed)	-7.72204E+01 (1.71E+01) (failed)	-9.9900E+02 (7.35E-01)	-2.8522E+02 (6.08E+01) (failed)	\
127. scurly10 (n = 100)	-9.9690E+03 (8.86E+02)	3.9372E+16 (1.26E+00) (failed)	-1.0031E+04 (2.85E+04) (too long)	-1.0000E+04 (2.99E+02)	\
128. sineval (n = 2)	0 (6.13E-01)	2.4503E-21 (4.75E-01)	6.6837E-27 (5.05E-03)	7.0103E-17 (3.26E+00)	\
129. sinquad (n = 1000)	0.0000E+00 (1.14E+02)	0.0000E+00 (9.78E-01)	3.5072E-13 (1.22E+00)	3.9372E+16 (1.49E+02) (failed)	\
130. sisser (n = 2)	0 (3.62E-02)	3.1554E-30 (1.30E-01)	8.1216E-13 (1.15E-01)	1.5832E-23 (6.84E-01)	\
131. srosenbr (n = 1000)	0 (6.25E+01)	0 (8.58E-01)	6.1274E-17 (8.68E-01)	1.2100E+04 (1.11E+02) (failed)	\
132. tridia (n = 1000)	0 (4.56EE+01)	6.8863E+04 (1.05E+00) (failed)	1.0364E-213 (4.57E-01)	5.5544E-13 (5.04E+01)	\

Table 19: Numerical results of CUTEst problems [35] (no.133-148) 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
133. vardim (n = 100)	0 (1.46E-01)	0 (1.85E+00)	2.4591E-09 (1.16E-02)	1.16E-02 (1.03E+01) (failed)	\
134. watson (n = 31)	1.8000E-07 (7.18E+00)	2.4192E-08 (1.42E+00)	1.0331E-13 (1.36E-01)	2.0126E-08 (3.20E+01)	\
135. woods (n = 1000)	0 (5.82E+01)	0 (8.76E-01)	8.5519E-13 (1.95E-01)	9.1626E+02 (1.11E+02) (failed)	\
136. zangwil2 (n = 2)	-8.4267E+01 (1.91E-01)	-1.8200E+01 (1.18E-01)	-1.8200E+01 (4.65E-02)	3.0867E+01 (1.39E+00) (failed)	\
137. fletchbv (n = 1000)	-2.3742E+12 (4.19E+02)	-5.0080E+16 (4.49E+01)	-6.6040E+12 (5.37E+02)	-2.0046E+12 (1.29E+02)	\
138. heart6ls (n = 6)	4.2957E-01 (5.60E+00) (failed)	3.7406E-01 (7.29E-01) (failed)	3.1141E-29 (3.54E-01)	2.0119E+01 (2.82E+00) (failed)	\
139. heart8ls (n = 8)	1.5893E+02 (2.32E+00) (failed)	1.2026E-12 (5.63E-01)	1.6754E-29 (3.72E-02)	9.7151E-16 (2.41E+00)	\
140. hilberta (n = 10)	0 (2.28E-01)	2.2431E-10 (1.73E-01)	2.0882E-20 (2.89E+04) (too long)	2.1124E-12 (2.64E+00)	\
141. himmelbf (n = 4)	3.7348E+02 (5.39E-01)	3.1857E+02 (3.74E-01)	3.1857E+02 (2.86E+04) (too long)	1.3680E+04 (2.30E+00) (failed)	\
142. humps (n = 2)	0 (4.39E-01)	3.9968E-17 (5.47E-01)	1.5767E-28 (1.68E-01)	1.2879E-18 (1.79E+00)	\
143. methanb8 (n = 31)	7.3987E+03 (3.13E+01) (failed)	1.5417E+01 (5.27E-01) (failed)	2.6011E-26 (2.20E-02)	1.5953E+02 (3.80E+01) (failed)	\
144. nasty (n = 2)	0 (1.89E+00)	2.5531E-10 (1.62E-01)	3.9842E-28 (2.78E-02)	5.0000E-01 (1.91E+00) (failed)	\
145. osborneb (n = 11)	3.4543E-01 (1.17E+00) (failed)	1.7898E+01 (1.38E+00) (failed)	4.0138E-02 (2.84E+04) (too long)	1.4742E+00 (6.47E+00) (failed)	\
146. yfitu (n = 3)	3.5106E+03 (2.73E+00) (failed)	7.5667E-08 (2.79E-01)	6.6697E-13 (1.20E-02)	2.4737E+03 (2.29E+00) (failed)	\
147. denschna (n = 2)	0 (3.50E-01)	2.0365E-14 (1.39E-01)	4.6923E-24 (3.56E-03)	4.1868E-17 (2.07E+00)	\
148. dixmaank (n = 2)	1 (3.11E+02)	2.7556E+03 (4.68E+00) (failed)	1 (2.62E+04) (too long)	1 (4.15E+03)	\

Table 20: Numerical results of CUTEst problems [35] (no.69-84) computed by CNMGE, MCS, GLODS, and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
69. Allinitu (n = 4)	5.7444E+00 (4.50E-01)	5.7444E+00 (5.48E-01)	5.7444E+00 (2.83E+01)	5.7444E+00 (1.43E-01)	\
70. cliff (n = 2)	1.9979E-01 (2.63E+00)	1.8377E-01 (1.84E-01)	2.0026E-01 (5.37E+01)	1.5985E+02 (5.82E-02) (failed)	\
71. dixon3dq (n = 10)	0 (1.17E-01)	0 (6.21E-01)	4.7326E-05 (1.38E+01)	7.8000E-03 (7.59E-02) (failed)	\
72. edensch (n = 2000)	1.2003E+04 (8.66E+02)	1.2003E+04 (6.83E+02)	3.3948E+04 (7.8000E-03) (failed)	1.2003E+04 (5.42E+01)	\
73. fletcher (n = 100)	0 (1.66E+00)	7.1882E+01 (2.27E+01) (failed)	3.8000E+03 (6.11E+00) (failed)	1.9598E-04 (1.17E+00)	\
74. genrose (n = 500)	1 (2.14E+01)	4.9496E+02 (1.61E+04) (failed)	4.9985E+02 (4.32E+01) (failed)	4.7956E+02 (1.91E+00) (failed)	\
75. hairy (n = 2)	2.0000E+01 (1.16E+01)	2.0000E+01 (2.69E-01)	2.0000E+01 (5.95E+01)	2.0000E+01 (4.71E-01)	\
76. himmelbb (n = 2)	0 (2.98E-01)	0 (5.23E-01)	failed (failed) (failed)	4.9585E-09 (2.77E-02)	\
77. himmelbg (n = 2)	0 (5.00E-01)	0 (1.83E-01)	0 (6.30E+01)	5.6048E-08 (4.95E-02)	\
78. indef (n = 1000)	-2.0924E+07 (7.14E+01)	-1.0001E+06 (2.19E+02)	-1.0000E+35 (1.16E+02)	-1.0000E+50 (1.99E+01)	\
79. jensmp (n = 2)	1.2436E+02 (3.51E-01)	1.2436E+02 (3.56E-01)	1.2436E+02 (5.95E+01)	1.2957E+02 (4.41E-02)	\
80. liarwhd (n = 1000)	0 (6.72E+01)	1.8869E+02 (6.03E+01) (failed)	9.9550E+02 (1.45E+02) (failed)	4.2000E-03 (3.69E+00) (failed)	\
81. loghairy (n = 2)	1.8232E-01 (3.02E-01)	1.8232E-01 (1.84E-01)	1.8232E-01 (6.35E+01)	1.1440E+00 (2.89E-02) (failed)	\
82. maratosb (n = 2)	-1 (1.44E-01)	-7.3938E-01 (4.35E-01) failed	-1 (6.22E+01)	-1 (3.07E-02)	\
83. mexhat (n = 2)	-4.0100E-02 (1.81E-01)	-4.0100E-02 (3.30E-01)	-4.0100E-02 (5.06E+01)	-4.0100E-02 (2.80E-02)	\
84. nondia (n = 1000)	0 (2.33E+01)	9.9992E-01 (6.67E+01) (failed)	1.0000E+00 (1.15E+02) (failed)	5.8208E-09 (4.48E+00)	\

Table 21: Numerical results of CUTEst problems [35] (no.85-100) computed by CNMGE, MCS, GLODS, and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
85. nondquar (n = 1000)	0 (1.04E+02)	0 (7.74E+03)	0 (1.17E+02)	2.6618E-04 (2.23E+01)	\
86. penalty1 (n = 1000)	2.5000E-03 (6.63E+01)	4.4289E-03 (6.04E+01)	1.2020E-02 (1.41E+02) (failed)	1.2600E-02 (4.09E+00) (failed)	\
87. power (n = 1000)	0 (8.30E+00)	1.1733E-18 (1.26E+04) (too long)	0 (1.16E+02)	2.4224E-06 (4.21E+00)	\
88. arglinb (n = 10)	4.6341E+00 (1.89E+00)	4.6341E+00 (5.70E-01)	4.6341E+00 (1.05E+01)	4.6341E+00 (9.22E-01)	\
89. arglinc (n = 10)	5.1351E+00 (1.72E+00)	7.1351E+00 (5.37E-01)	7.1351E+00 (1.00E+01)	7.1351E+00 (1.11E-01)	\
90. arwhead (n = 1000)	0 (8.40E+01)	0 (6.17E+01)	2.9670E+03 (1.51E+02) (failed)	5.4570E-12 (4.55E+00)	\
91. bard (n = 3)	8.2149E-03 (2.61E-01)	8.2149E-03 (4.89E-01)	5.7220E+59 (3.84E+01) (failed)	8.2149E-03 (1.04E-01)	\
92. bdexp (n = 1000)	-∞ (1.33E+02)	-2.3349E+99 (2.18E+02)	0.0000E+00 (1.17E+02) (failed)	0.0000E+00 (6.07E+00) (failed)	\
93. bdqrtic (n = 1000)	3.9838E+03 (9.47E+01)	3.9838E+03 (7.2427E+01)	8.9367E+03 (1.45E+02) (failed)	3.9838E+03 (3.98E+03)	\
94. biggs6 (n = 6)	0 (5.06E-01)	6.4095E-31 (6.43E-01)	3.8664E-19 (1.72E+01)	5.2994E-12 (5.18E-02)	\
95. box3 (n = 3)	5.2994E-12 (5.18E-02)	0 (1.82E-01)	0 (3.78E+01)	7.4821E-04 (2.63E-02)	\
96. brkmcc (n = 2)	1.6904E-01 (4.30E-01) (failed)	-1.266E+03 (5.92E-01)	1.6904E-01 (6.12E+01) (failed)	1.6904E-01 (1.69E-01) (failed)	$-\infty$
97. brownal (n = 10)	0 (1.16E+00)	1.2508E-28 (7.93E-01)	failed (failed) (failed)	2.9211E-04 (4.07E-01) (failed)	\
98. brownbs (n = 2)	0 (1.83E-01)	9.9999E+11 (2.75E-01) (failed)	1.0526E-05 (6.28E+01)	1.0832E-05 (6.61E-01)	\
99. brownden (n = 4)	8.5822E+04 (1.67E-01)	1.9995E+06 (2.27E-01) (failed)	8.5822E+04 (2.83E+01)	8.5947E+04 (3.18E-01)	\
100. broydn7d (n = 1000)	1.2333E+00 (4.01E+02) (failed)	6.3867E+02 (2.14E+02) (failed)	9.9851E+02 (1.24E+02) (failed)	4.0145E+01 (3.76E+01) (failed)	\

Table 22: Numerical results of CUTEst problems [35] (no.101-116) computed by CNMGE, MCS, GLODS, and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
101. chainwoo (n = 1000)	1 (9.19E+01)	1.8866E+03 (1.05E+02) (failed)	2.0928E+04 (1.43E+02) (failed)	1.1987E+03 (5.12E+00) (failed)	\
102. chnrosnb (n = 50)	0 (2.29E+00)	5.4636E-06 (6.52E+00)	4.0289E+01 (6.80E+00) (failed)	1.1332E-04 (7.85E-01)	\
103. cosine (n = 1000)	-9.9900E+02 (6.04E+01)	-9.9068E+02 (2.73E+02)	-9.9899E+02 (1.17E+02)	-9.9900E+02 (7.30E+00)	\
104. cragglvy (n = 1000)	3.3642E+02 (2.45E+02)	4.2166E+02 (1.44E+02) (failed)	9.9189E+02 (6.05E+1) (failed)	3.4398E+02 (3.58E+01)	\
105. cube (n = 2)	0 (4.55E-02)	1.7485E-18 (4.24E-01)	1.0673E-04 (5.28E+01)	3.0215E-09 (6.26E-02)	\
106. curly10 (n = 1000)	failed (failed) (failed)	-9.9550E+06 (1.28E+04) (too long)	-9.9550E+35 (5.14E+02)	-1.0000E+50 (3.99E+03)	\
107. dqdrtic (n = 1000)	0 (2.07E+01)	0 (7.54E+03)	0 (1.15E+02)	3.1000E-03 (4.95E+00) (failed)	\
108. dqrtic (n = 1000)	0 (8.76E+01)	1.9231E-09 (2.11E+02)	2.0046E+14 (1.73E+02) (failed)	3.9000E-03 (1.91E+01) (failed)	\
109. eg2 (n = 1000)	-9.9895E+02 (1.23E+01)	-9.9991E+02 (7.25E+01)	-9.9947E+02 (1.17E+02)	8.5037E-091 (1.96E+01) (failed)	\
110. engval1 (n = 1000)	1.1082E+03 (7.48E+01)	1.1118E+03 (4.63E+01)	2.9760E+03 (1.53E+02) (failed)	1.1082E+03 (4.16E+00)	\
111. engval2 (n = 3)	0 (1.00E+00)	6.5480E-15 (8.41E-01)	5.5558E-04 (3.29E+01)	5.0457E-04 (7.37E-02)	\
112. errinros (n = 50)	3.9904E+01 (5.74E+00)	3.9904E+01 (5.32E+00)	4.7803E+01 (5.88E+00)	4.0353E+01 (2.87E-01)	\
113. extrosnb (n = 10)	0 (3.89E-01)	7.9637E-05 (1.47E+00)	3.0713E-02 (1.50E+01) (failed)	4.5475E-10 (1.13E-01)	\
114. fletcbv3 (n = 1000)	-4.5167E-02 (3.02E+01) (failed)	-1.0018E-04 (1.91E+04) (failed)	-1.5969E-03 (2.20E+02) (failed)	-1.3585E+01 (2.21E+01) (failed)	$-\infty$
115. genhumps (n = 5)	0 (4.78E-01)	0 (4.79E-01)	0 (2.07E+01)	2.2100E+01 (3.00E-02) (failed)	\
116. gulf (n = 3)	failed (failed) (failed)	1.5279E-01 (1.16E+00) (failed)	3.2835E+01 (3.62E+01) (failed)	3.2835E+01 (1.64E-01) (failed)	\

Table 23: Numerical results of CUTEst problems [35] (no.117-132) computed by CNMGE, MCS, GLODS, and VRBBO.

Problem	CNMGE $\min f(x)$ (CPU time (s))	MCS $\min f(x)$ (CPU time (s))	GLODS $\min f(x)$ (CPU time (s))	VRBBO $\min f(x)$ (CPU time (s))	Global minimum
117. hilbertb (n = 50)	0 (4.14E-01)	0 (4.44E+00)	0 (3.93EE+00)	1.64E-01 (1.54E+001) (failed)	\
118. himmelbh (n = 2)	-1.0000E+10 (1.75E-01)	-1.0000E+00 (1.60E-01) (failed)	-1.0000E+96 (6.16E+01)	-1.0000E+00 (3.19E-02) (failed)	\
119. kowosb (n = 4)	1.1037E-03 (1.42E+00)	3.0751E-04 (1.77E+00)	3.0751E-04 (2.53E+01)	3.0752E-04 (5.80E-02)	\
120. meyer3 (n = 3)	3.6710E+06 (1.74E+01) (failed)	3.8895E+09 (2.97E-01) (failed)	9.1344E+07 (3.25E+01) (failed)	8.1664E+04 (7.3.E-01) (failed)	\
121.noncvxu2 (n = 1000)	2.3168E+03 (1.57E+02)	2.4861E+03 (9.95E+01)	3.9642E+03 (1.45E+02) (failed)	5.1019E+04 (1.39E+01) (failed)	\
122.osbornea (n = 5)	5.4650E-05 (8.08E-01)	5.4652E-05 (1.34E+00)	1.1529E+00 (2.06E+01) (failed)	5.5846E-04 (4.61E-02)	\
123. penalty2 (n = 100)	9.7096E+04 (3.17E+00)	9.7119E+04 (3.12E+00)	9.7096E+04 (3.80E+00)	9.7096E+04 (6.04-01)	\
124. quartc (n = 1000)	0 (7.61E+01)	1.9231E-09 (2.88E+02)	2.0046E+14 (1.74E+02) (failed)	2.1112E-08 (1.98E+01)	\
125. rosenbr (n = 1000)	0 (1.24E-01)	2.7140E-20 (4.58E-01)	0 (5.22E+01)	1.3128E-05 (1.74E-02)	\
126. scosine (n = 1000)	-8.3699E+01 (1.40E+02) (failed)	-8.6657E+02 (1.56E+02) (failed)	-7.0061E+01 (1.17E+02) (failed)	-9.9576E+02 (2.47E+01)	\
127. scurly10 (n = 100)	-9.9690E+03 (8.86E+02)	-9.9975E+03 (3.49E+01)	-2.0351E+03 (1.01E+01) (failed)	3.4050E+17) (1.96E+00) (failed)	\
128. sineval (n = 2)	0 (6.13E-01)	0 (4.01E-01)	0 (6.18E+01)	3.8282E+00 (6.54E-02) (failed)	\
129. sinquad (n = 1000)	0 (1.14E+02)	2.7484E-02 (8.2956E+01) (failed)	7.5169E-01 (1.1788E+02) (failed)	5.2932E+00 (5.2932E+00) (failed)	\
130. sisser (n = 2)	0 (3.62E-02)	0 (9.18E-01)	0 (6.13E+01)	6.6534E-12 (2.89E-02)	\
131. srosenbr (n = 1000)	0 (6.25E+01)	3.4391E+01 (7.61E+01) (failed)	4.9963E+02 (1.42E+02) (failed)	1.0910E+02 (3.67E+00) (failed)	\
132. tridia (n = 1000)	0 (4.56EE+01)	1.9438E-16 (7.3388E+01)	1.5625E-01 (1.36E+02) (failed)	1.1600E-02 (4.55E+00)	\

Table 24: Numerical results of CUTEst problems [35] (no.133-148) computed by CNMGE, MCS, GLODS, and VRBBO.

Problem	CNMGE min $f(x)$ (CPU time (s))	MCS min $f(x)$ (CPU time (s))	GLODS min $f(x)$ (CPU time (s))	VRBBO min $f(x)$ (CPU time (s))	Global minimum
133. vardim (n = 100)	0 (1.46E-01)	7.8388E+04 (8.78E+01) (failed)	2.0284E+07 (9.38E+00) (failed)	2.7162E+05 (3.75E-01) (failed)	\
134. watson (n = 31)	1.8000E-07 (7.18E+00)	5.6003E-07 (1.20E+01)	5.6725E-04 (8.04E+00)	5.0000E-03 (1.80E+00)	\
135. woods (n = 1000)	0 (5.82E+01)	2.3472E+01 (7.78E+01) (failed)	1.0475E+04 (1.44E+02) (failed)	1.8114E+02 (4.08E+00) (failed)	\
136. zangwil2 (n = 2)	-8.4267E+01 (1.91E-01)	3.0867E+01 (1.04E-01) (failed)	-1.8200E+01 6.21E+01 (failed)	-1.8200E+01 3.40E-02 (failed)	\
137. fletchbv (n = 1000)	-2.3742E+12 (4.19E+02)	-2.0056E+12 (1.72E+04) (too long)	-3.1937E+13 2.24E+02	-4.0908E+19 1.78E+01	\
138. heart6ls (n = 6)	4.2957E-01 (5.60E+00) (failed)	2.3780E-01 (1.50E+00) (failed)	9.8816E+01 (2.16E+01) (failed)	5.5834E+01 (1.37E+00) (failed)	\
139. heart8ls (n = 8)	1.5893E+02 (2.32E+00) (failed)	1.4290E+00 (2.23E+00) (failed)	1.8154E+00 (1.52E+01) (failed)	1.1073E+00 (1.23E-01) (failed)	\
140. hilberta (n = 10)	0 (2.28E-01)	0 (4.52E-01)	0 (1.03E+01)	2.6152E-05 (1.17E-01)	\
141. himmelbf (n = 4)	3.7348E+02 (5.39E-01)	1.3680E+04 (8.28E-01) (failed)	3.4717E+02 (2.62E+01)	3.5038E+02 (4.51E-02)	\
142. humps (n = 2)	0 (4.39E-01)	0 (6.55E-01)	0 (6.30E+01)	4.3893E-09 (4.89E-02)	\
143. methanb8 (n = 31)	7.3987E+03 (3.13E+01) (failed)	1.9658E+02 (3.03E+00) (failed)	4.1851E+03 (5.72E+00) (failed)	3.3589E-04 (2.18E+01)	\
144. nasty (n = 2)	0 (1.89E+00)	0 (9.64E-02)	0 (6.25E+01)	2.9802E+12 (5.85E-01) (failed)	\
145. osborneb (n = 11)	3.4543E-01 (1.17E+00) (failed)	2.5176E-01 (2.11E+00) (failed)	1.4742E+00 (9.56E+00) (failed)	4.0100E-02 (2.10E-01)	\
146. yfitu (n = 3)	3.5106E+03 (2.73E+00) (failed)	2.2396E+03 (9.10E-01) (failed)	5.9757E+03 (3.79E+01) (failed)	5.3270E-01 (5.33E-01) (failed)	\
147. denschna (n = 2)	0 (3.50E-01)	0 (1.39E-01)	0 (6.18E+01)	2.9801E-07 (4.47E-02)	\
148. dixmaank (n = 2)	1 (3.11E+02)	1.0065E+00 (2.48E+03)	failed (failed) (failed)	1 (1.35E+02)	\