Parallel Computing (EE664) Project
Synopsis - Project No. 17 (Using OpenMP)

Mohammed Adnan (150108021)
Sahil Thandra (150108046)
Souradip Pal (150102076)

A Simple Yet Efficient Evolution Strategy for Large-Scale Black-Box Optimization

1 Introduction:

Black-box optimization is the case when no information about the gradients is available. Black-box optimization finds application in many fields such as Machine Learning. There are various algorithms for optimizing black box functions such as derivative free optimization, Covariance matrix adaption (CMA-ES) etc.

CMA-ES find solution by approximating contour of function with Gaussian distribution which it maintains and update to find the optimum solution. Evolution strategy is define as the process in which potential solutions are sampled using multivariate normal distribution and then new parameters for the distribution is updates. A random vector with mean zero is added for mutation. The correlation between different variables are captured by the covariance matrix. This method of updating the covariance matrix of the distribution is known as covariance matrix adaptation (CMA) distribution and it finds application in cases when the objective function is ill conditioned.

A number of matrix can be used, among which the most basic one if the rank one model:

$$C = \alpha I + \beta v v^T \tag{1}$$

1.1 Drawbacks

Rank One model has problem in adapting mutation strength and principal search direction effectively. The authors in the paper proposed R1-ES and Rm-ES algorithm with time complexity O(n) and is also able to capture long valley problem.

2 Proposed Algorithm:

Authors instead proposed to define Co-variance matrix as C:

$$C = \alpha M + K \tag{2}$$

where M and K denotes a sparse and low rank matrix respectively. When M is identity matrix K is some matrix whose rank is equal to one, the model is rank one model. Further, if set K to be rank m, we have:

$$\mathbf{C} = \alpha \mathbf{I} + \sum_{i=1}^{m} \beta_i \mathbf{v}_i \mathbf{v}_i^T \tag{3}$$

2.1 R1-ES:

In R1-ES, K in eq(2) is a rank one matrix and the co-variance matrix is then expressed as

$$C_t = (1 - c_{cov})I + c_{cov} p_t p_t^T \tag{4}$$

where $\mathrm{p_t}$ is the principal search direction and $\mathrm{c_{cov}}$ is changing rate of co-variance matrix. Like all evolution strategy algorithm, R1-ES evolves and maintain following variables:

$\mathrm{m_t}$: The distribution mean.
$\mathrm{p_t}$: The principal search direction.
σ_{t} : The mutation strength.
$\mathrm{s_t}$: The cumulative rank rate.
$\mathrm{x_{best}}$: The best solution found so far.

The procedure to get the solution is as following:

1. λ new potential solution generated by below given equation are sampled:

$$\mathbf{x} = \mathbf{m}_t + \sigma_t \left(\sqrt{1 - c_{\mathrm{cov}}}\mathbf{z} + \sqrt{c_{\mathrm{cov}}} r \mathbf{p}_t\right) \tag{5}$$

2. Value of the objective function are calculated and are sorted in an increasing order. Best mu solution are taken to update parameters. Fitness of the solution is denoted by ($\mathrm{F_{t+1}}$).

3. Update the distribution mean for the next iteration.

4. Principal search direction can be update as:

$$\mathbf{p}_{t+1} = (1 - c)\mathbf{p}_t + \sqrt{c(2 - c)\mu_{\mathrm{eff}}}\frac{(\mathbf{m}_{t+1} - \mathbf{m}_t)}{\sigma_t} \tag{6}$$

Evolution path $\mathrm{p_{t+1}}$ is a weighted average of mean of movement.

5. Rank-Based Adaptation for Mutation Strength: This step involves calculation of mutation strength. Fitness of selection solution of current and previous iteration ($\mathrm{F_{t+1}}$ and $\mathrm{F_t}$) are sorted and denoted by $\mathrm{R_t(I)}$ and $\mathrm{R_{t+1}(I)}$ Rank diff(q) is calculated as:

$$q = \frac{1}{\mu}\sum_{i=1}^{\mu} w_i \left(R_t(i) - R_{t+1}(i)\right) \tag{7}$$

And the cumulative rank rate is computed as:

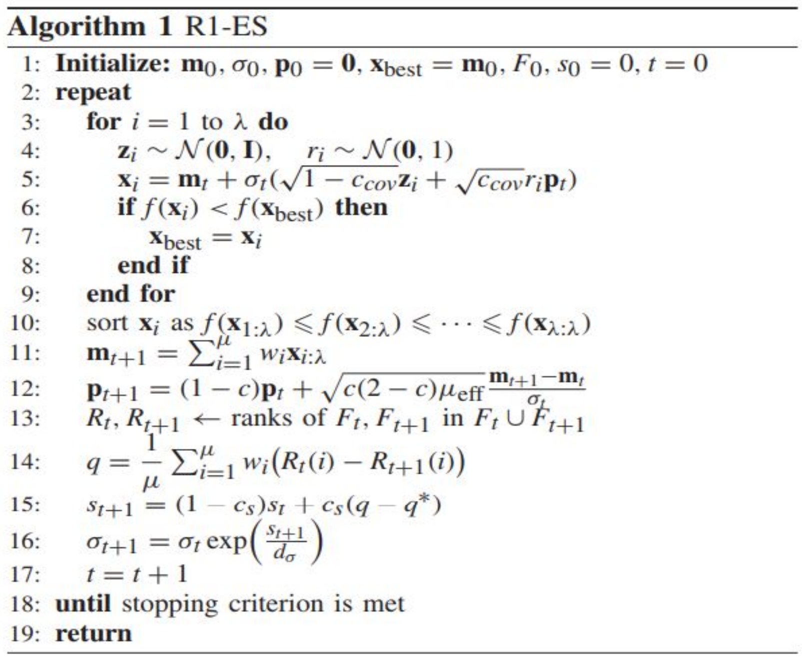
$$s_{t+1} = (1 - c_s)s_t + c_s\left(q - q^*\right) \tag{8}$$

Mutation strength is adapted as:

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{s_{t+1}}{d_\sigma}\right) \tag{9}$$

2.2 R1-ES Algorithm:

The algorithm thus can be summarized as:

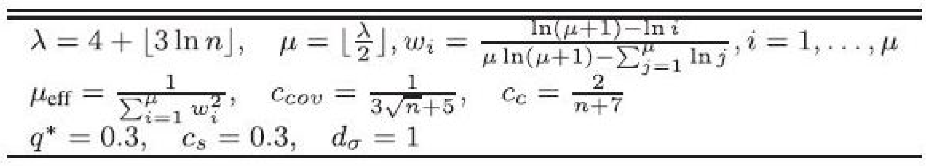


Algorithm 1 R1-ES

```
1:  Initialize: $\mathbf{m}_0, \sigma_0, \mathbf{p}_0 = \mathbf{0}, \mathbf{x}_{\text{best}} = \mathbf{m}_0, F_0, s_0 = 0, t = 0$
2:  repeat
3:      for $i = 1$ to $\lambda$ do
4:          $\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad r_i \sim \mathcal{N}(\mathbf{0}, 1)$
5:          $\mathbf{x}_i = \mathbf{m}_t + \sigma_t(\sqrt{1 - c_{cov}}\mathbf{z}_i + \sqrt{c_{cov}} r_i \mathbf{p}_t)$
6:          if $f(\mathbf{x}_i) < f(\mathbf{x}_{\text{best}})$ then
7:              $\mathbf{x}_{\text{best}} = \mathbf{x}_i$
8:          end if
9:      end for
10:     sort $\mathbf{x}_i$ as $f(\mathbf{x}_{1:\lambda}) \leqslant f(\mathbf{x}_{2:\lambda}) \leqslant \cdots \leqslant f(\mathbf{x}_{\lambda:\lambda})$
11:     $\mathbf{m}_{t+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$
12:     $\mathbf{p}_{t+1} = (1 - c)\mathbf{p}_t + \sqrt{c(2 - c)\mu_{\text{eff}}} \frac{\mathbf{m}_{t+1} - \mathbf{m}_t}{\sigma_t}$
13:     $R_t, R_{t+1} \leftarrow$ ranks of $F_t, F_{t+1}$ in $F_t \cup F_{t+1}$
14:     $q = \frac{1}{\mu} \sum_{i=1}^{\mu} w_i \big(R_t(i) - R_{t+1}(i)\big)$
15:     $s_{t+1} = (1 - c_s)s_t + c_s(q - q^*)$
16:     $\sigma_{t+1} = \sigma_t \exp\left(\frac{s_{t+1}}{d_\sigma}\right)$
17:     $t = t + 1$
18: until stopping criterion is met
19: return
```

This sequential algorithm was implemented in C++. The C++ vector library has been used to define the vectors m_0, p_0, F_0 and x_i etc. Also, Mersenne Twister pseudo-random generator(mt19937) were used to generate the random vectors $\mathbf{z}_i$ and the random variables r_i following normal distributions. For adaptation of mutation strength using rank based success rule, the C++ set data structure was used. The algorithm runs for either a predefined number of iterations or stops based on the stopping criterion that the function value reaches very close to the converging value.

The parameters used for R1-ES are given in the table below:



$\lambda = 4 + \lfloor 3 \ln n \rfloor, \quad \mu = \lfloor \frac{\lambda}{2} \rfloor, w_i = \frac{\ln(\mu+1) - \ln i}{\mu \ln(\mu+1) - \sum_{j=1}^{\mu} \ln j}, i = 1, \ldots, \mu$

$\mu_{\text{eff}} = \frac{1}{\sum_{i=1}^{\mu} w_i^2}, \quad c_{cov} = \frac{1}{3\sqrt{n}+5}, \quad c_c = \frac{2}{n+7}$

$q^* = 0.3, \quad c_s = 0.3, \quad d_\sigma = 1$

Also m_0 was initialized to a vector in the range $[-10, 10]^n$ and σ_0 to 20/3.

3 Rm-ES:

Rm-ES is an extended version of R1-ES with 'm' evolution path to generate solution. Covariance matrix is defined as:

$$\mathbf{C} = (1 - c_{\text{cov}})^m \mathbf{I} + c_{\text{cov}} \sum_{i=1}^{m} (1 - c_{\text{cov}})^{m-i} \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^T. \tag{10}$$

It maintains 'm' of evolutionary path [p] and their generation t at each iteration. This allows to adapt evolutionary paths to generate solution instead of storing co-variance matrix explicitly.

3.1 Sampling:

The low rank model $N(m_{\mathrm{t}}, {\sigma^2}_{\mathrm{t}} C_{\mathrm{t}})$ with co-variance matrix $\mathrm{C_t}$ can compute a new possible solution x as:

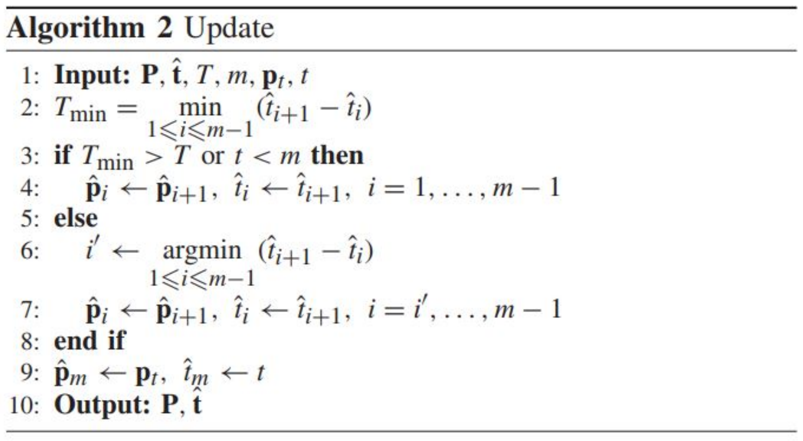
$$\mathbf{x} = \mathbf{m}_t + \sigma_t \left(a^m \mathbf{z} + b \sum_{i=1}^{m} a^{m-i} r_i \hat{\mathbf{p}}_i \right) \tag{11}$$

Time Complexity is O(mn).

3.2 Update of Direction:

Initialization of evolution path is done with zero entries. During the initial m generations, each generated paths is stored p_t. Consecutively, update of P is done. Current evolution path is used for the update. If difference between p^{i+1} and p^i is less than T, then p^{i+1} is expunged. The current evolution path is stored and the update is made accordingly.

Algorithm for update can be summarised as:

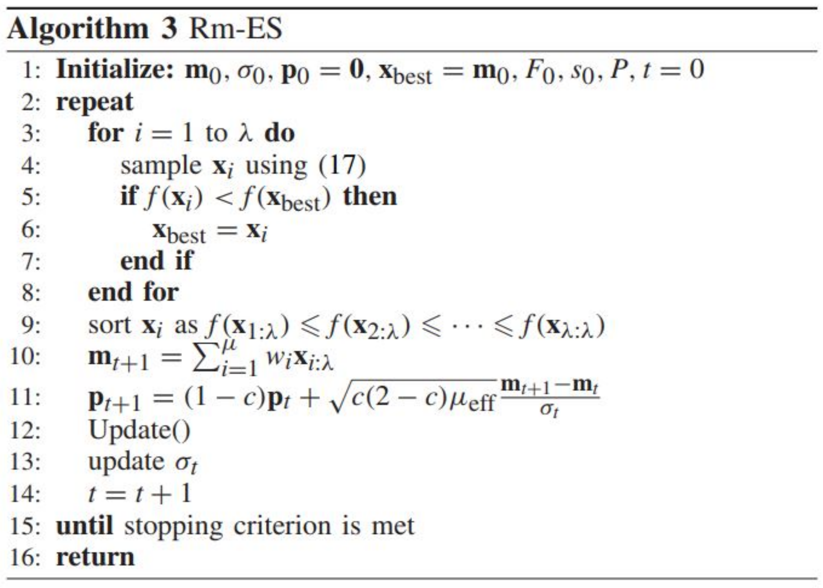


Algorithm 2 Update

```
1: Input: $\mathbf{P}, \hat{\mathbf{t}}, T, m, \mathbf{p}_t, t$
2: $T_{\min} = \min_{1 \leqslant i \leqslant m-1} (\hat{t}_{i+1} - \hat{t}_i)$
3: if $T_{\min} > T$ or $t < m$ then
4:     $\hat{\mathbf{p}}_i \leftarrow \hat{\mathbf{p}}_{i+1},\ \hat{t}_i \leftarrow \hat{t}_{i+1},\ i = 1, \ldots, m-1$
5: else
6:     $i' \leftarrow \operatorname{argmin}_{1 \leqslant i \leqslant m-1} (\hat{t}_{i+1} - \hat{t}_i)$
7:     $\hat{\mathbf{p}}_i \leftarrow \hat{\mathbf{p}}_{i+1},\ \hat{t}_i \leftarrow \hat{t}_{i+1},\ i = i', \ldots, m-1$
8: end if
9: $\hat{\mathbf{p}}_m \leftarrow \mathbf{p}_t,\ \hat{t}_m \leftarrow t$
10: Output: $\mathbf{P}, \hat{\mathbf{t}}$
```

3.3 Rm-ES Algorithm:

The algorithm for Rm-ES can be summarised as:



Algorithm 3 Rm-ES

```
1: Initialize: $\mathbf{m}_0, \sigma_0, \mathbf{p}_0 = \mathbf{0}, \mathbf{x}_{\text{best}} = \mathbf{m}_0, F_0, s_0, P, t = 0$
2: repeat
3:     for $i = 1$ to $\lambda$ do
4:         sample $\mathbf{x}_i$ using (17)
5:         if $f(\mathbf{x}_i) < f(\mathbf{x}_{\text{best}})$ then
6:             $\mathbf{x}_{\text{best}} = \mathbf{x}_i$
7:         end if
8:     end for
9:     sort $\mathbf{x}_i$ as $f(\mathbf{x}_{1:\lambda}) \leqslant f(\mathbf{x}_{2:\lambda}) \leqslant \cdots \leqslant f(\mathbf{x}_{\lambda:\lambda})$
10:    $\mathbf{m}_{t+1} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}$
11:    $\mathbf{p}_{t+1} = (1-c)\mathbf{p}_t + \sqrt{c(2-c)\mu_{\text{eff}}} \frac{\mathbf{m}_{t+1} - \mathbf{m}_t}{\sigma_t}$
12:    Update()
13:    update $\sigma_t$
14:    $t = t + 1$
15: until stopping criterion is met
16: return
```

Similar implementation was done in case of Rm-ES as that in case of R1-ES with exception of two steps i.e. sampling and update of directions as shown earlier. The step ‘update σ_t’ remains the same as R1-ES. The ‘update’ function was implemented outside the main function in C++ with the parameters m=2 and T=n.

3.4 Complexity of the Serial Algorithms:

R1-ES: Initialization takes $O(n)$. Let there be T iterations. In each iteration the for-loop in lines 3 to 9 takes $O(\lambda n)$. Sorting of array of an n-dimensional vectors of length λ using quick-sort takes $O(n\lambda log(\lambda))$ time. Updation of vectors take $O(n)$ time. Rank find algorithm takes $O(\lambda^2)$ time. Taking the value of $\lambda = 4 + 3\lfloor log(n) \rfloor$, overall time complexity becomes $O(Tnlog(n)log(log(n)))$.

Rm-ES: Here ‘Sampling’ is of the order $O(mn)$ and ‘Update’ takes $O(mn)$ time. Rest are the same. Hence overall time complexity becomes $O(mnlog(n) + Tnlog(n)log(log(n)))$.

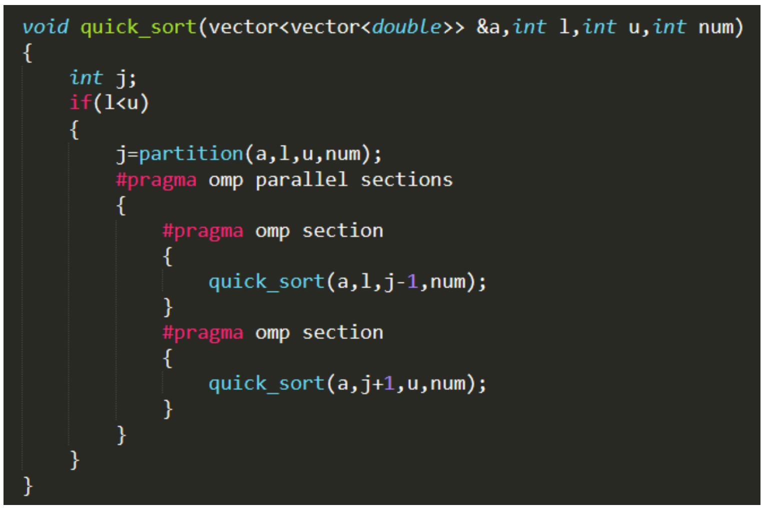
4 Parallelization using OpenMP:

We used OpenMP to parallelize various section of the algorithm. The pragma omp parallel directive is used to create additional threads to complete the process confined within the parallel construct.

4.1 Scope for Parallelization:

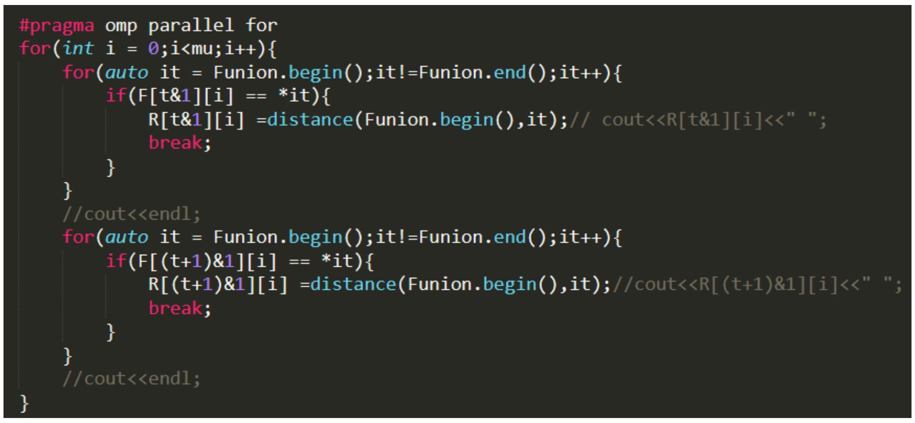
Following sections in the algorithm can be easily parallelized. In general, most of the parallelization is done over independent for loops. Also we have used reduction clauses to speed up vector summations.

1. Quick Sort: We have written a custom sort function to sort the vectors based on function values. We used ‘pragma omp section’ directive to parallelize the recursive calls within the sort function.



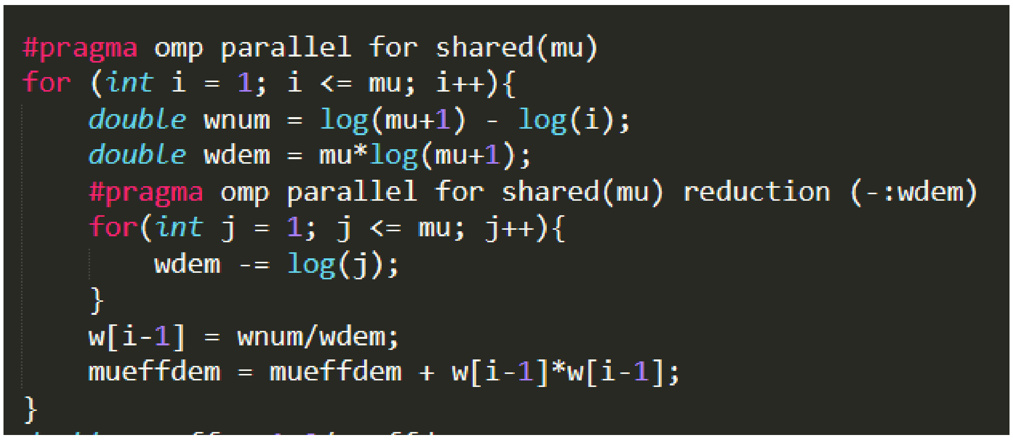
```
void quick_sort(vector<vector<double>> &a,int l,int u,int num)
{
    int j;
    if(l<u)
    {
        j=partition(a,l,u,num);
        #pragma omp parallel sections
        {
            #pragma omp section
            {
                quick_sort(a,l,j-1,num);
            }
            #pragma omp section
            {
                quick_sort(a,j+1,u,num);
            }
        }
    }
}
```

2. Find Rank: We used ‘pragma omp parallel for’ directive to parallelize the ’for’ loop.



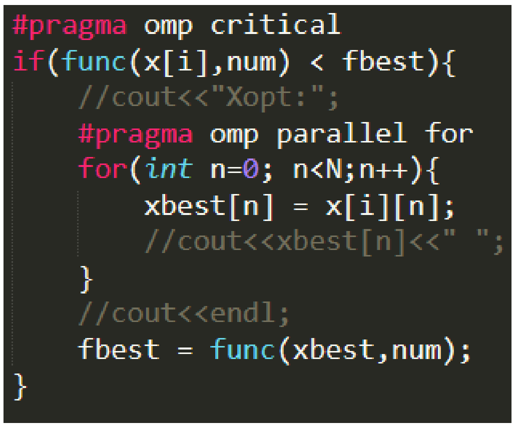
```
#pragma omp parallel for
for(int i = 0;i<mu;i++){
    for(auto it = Funion.begin();it!=Funion.end();it++){
        if(F[t&1][i] == *it){
            R[t&1][i] =distance(Funion.begin(),it);// cout<<R[t&1][i]<<" ";
            break;
        }
    }
    //cout<<endl;
    for(auto it = Funion.begin();it!=Funion.end();it++){
        if(F[(t+1)&1][i] == *it){
            R[(t+1)&1][i] =distance(Funion.begin(),it);//cout<<R[(t+1)&1][i]<<" ";
            break;
        }
    }
    //cout<<endl;
}
```

3. Initialization of variables: We used ‘pragma omp parallel for’ directive to initialize the variables in a parallel manner.



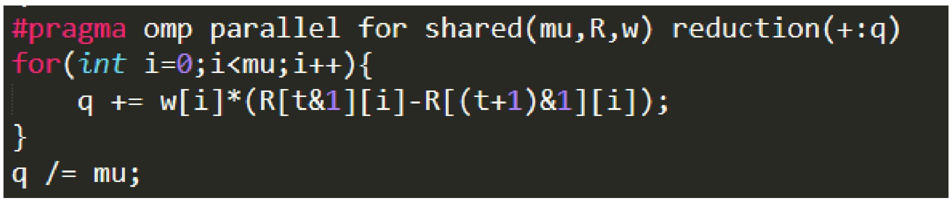
```
#pragma omp parallel for shared(mu)
for (int i = 1; i <= mu; i++){
    double wnum = log(mu+1) - log(i);
    double wdem = mu*log(mu+1);
    #pragma omp parallel for shared(mu) reduction (-:wdem)
    for(int j = 1; j <= mu; j++){
        wdem -= log(j);
    }
    w[i-1] = wnum/wdem;
    mueffdem = mueffdem + w[i-1]*w[i-1];
}
```

4. Usage of Critical Section: We used ‘pragma omp critical’ where we had to compare the function value in the current iteration with the previous best to avoid race condition.



```
#pragma omp critical
if(func(x[i],num) < fbest){
    //cout<<"Xopt:";
    #pragma omp parallel for
    for(int n=0; n<N;n++){
        xbest[n] = x[i][n];
        //cout<<xbest[n]<<" ";
    }
    //cout<<endl;
    fbest = func(xbest,num);
}
```

5. Usage of Reduction Clauses: We used ‘pragma omp parallel for’ reduction where ever there was a summation involved with independent variables.



```
#pragma omp parallel for shared(mu,R,w) reduction(+:q)
for(int i=0;i<mu;i++){
    q += w[i]*(R[t&1][i]-R[(t+1)&1][i]);
}
q /= mu;
```

4.2 Modified Algorithms:

Algorithm 1:- Parallel R1-ES:

```
1: Initialize: $m_0, \sigma_0, p_0 = 0, x_{best} = m_0, F_0, s_0 = 0, t = 0$
2: repeat
3: parallel_for i = 1 to $\lambda$ do
4:    $z_i \sim N(0, I), r_i \sim N(0, 1)$
5:    parallel update: $x_i = m_t + \sigma_t(\sqrt{1 - c_{cov}} z_i + \sqrt{c_{cov}} r_i p_t)$
6:    critical section {if $f(x_i) < f(x_{best})$ then
7:       $x_{best} = x_i$
8:    end if }
9: end for
10: parallel_sort $x_i$ as $f(x_{1:\lambda}) \leq f(x_{2:\lambda}) \leq ... \leq f(x_{\lambda:\lambda})$
11: reduce $m_{t+1} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}$
12: parallel update: $p_{t+1} = (1 - c)p_t + \sqrt{c(2 - c)\mu_{\text{eff}}} \frac{(m_{t+1} - m_t)}{\sigma_t}$
13: $R_t, R_{t+1} \leftarrow$ ranks of $F_t, F_{t+1}$ in $F_t \cup \mathrm{F}_{t+1}$
14: reduce $q = \frac{1}{\mu} \sum_{i=1}^{\mu} w_i(R_t(i) - R_{t+1}(i))$
15: $s_{t+1} = (1 - c_s)s_t + c_s(q - q^*)$
16: $\sigma_{t+1} = \sigma_t exp(\frac{s_{t+1}}{d\sigma})$
17: $t = t + 1$
18: until stopping criterion is met
19: return
```

Algorithm 2: Parallel Rm-ES

```
1: Initialize: $m_0, \sigma_0, p_0 = 0, x_{best} = m_0, F_0, s_0 = 0, P, t = 0$
2: repeat
3: parallel_for i=1 to $\lambda$ do
4:    sample $x_i$ using (11)[parallel reduction]
5:    critical section {if $f(x_i) < f(x_{best})$ then
6:       $x_{best} = x_i$
7:    end if }
8: end for
9: parallel_sort $x_i$ as $f(x_{1:\lambda}) \leq f(x_{2:\lambda}) \leq ... \leq f(x_{\lambda:\lambda})$
10: reduce $m_{t+1} = \sum_{i=1}^{\mu} w_i x_{i:\lambda}$
11: parallel update: $p_{t+1} = (1 - c)p_t + \sqrt{c(2 - c)\mu_{\text{eff}}} \frac{(m_{t+1} - m_t)}{\sigma_t}$
12: Update()
13: parallel update $\sigma_t$
14: $t = t + 1$
15: until stopping criterion is met
16: return
```

4.3 Complexity of the Parallel Algorithms:

R1-ES: Assuming negligible communication overhead and 'p' threads, initialization takes $O(n/p)$. Let there be T iterations. In each iteration, the for-loop in lines 3 to 9 takes $O(\lambda n/p)$ due to presence of critical section. Sorting of array of an n-dimensional vectors of length λ using quick-sort using 'omp sections' takes $O(n\lambda log(\lambda/p)/p)$ time. Updation of vectors take $O(n/p)$ time. Rank find algorithm takes $O(\lambda^2/p)$ time. Taking the value of $\lambda = 4 + 3\lfloor log(n) \rfloor$, overall time complexity becomes $O(Tnlog(n)log(log(n)/p)/p)$.

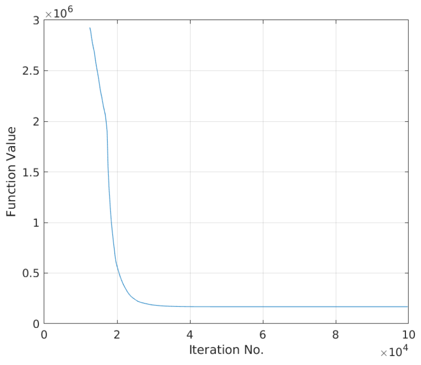
Rm-ES: Here 'Sampling' is of the order $O(mn/p)$ and 'Update' takes $O(mn)$ time. Rest are the same. Hence overall time complexity becomes $O(mnlog(n)/p + Tnlog(n)log(log(n)/p)/p)$.

5 Observation: Convergence of Test Functions

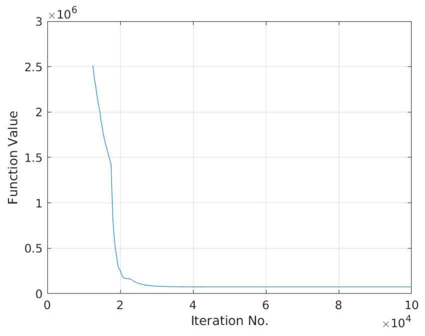
Here, testing platform was Intel Xeon i5-3rd Gen 8GB Quad-Core Processor with speed 3.2GHz.

5.1 100 Variables:

Ellipsoid: $f_{Ell}(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$

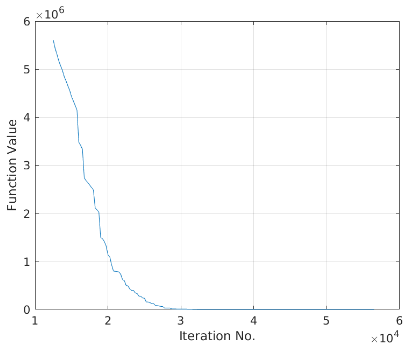


(a) Serial Version

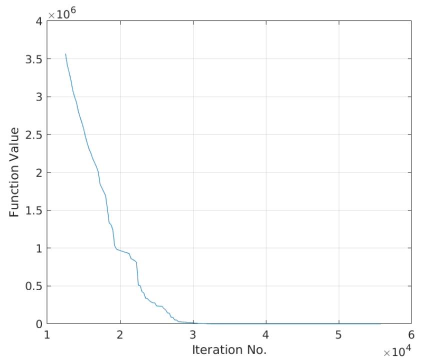


(b) Parallel Version(Num_threads = 8)

Figure 1: Plots of R1-ES serial and parallel version for Elliptic Function



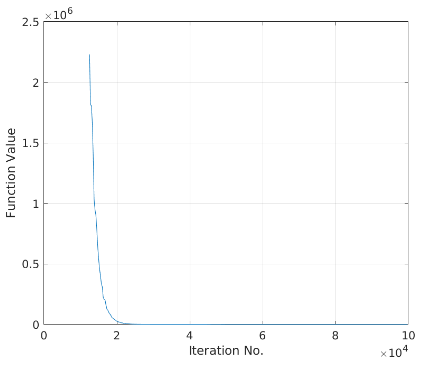
(a) Serial Version



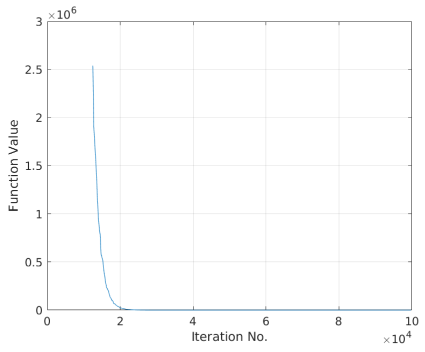
(b) Parallel Version(Num_threads = 8)

Figure 2: Plots of Rm-ES serial and parallel version for Elliptic Function

Cigar: $f_{Cig}(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$

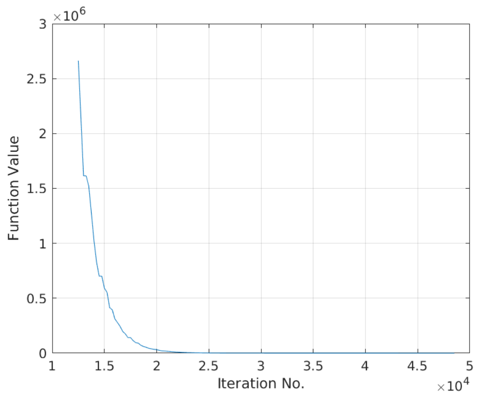


(a) Serial Version

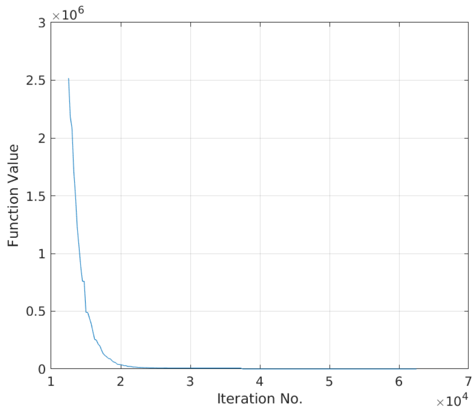


(b) Parallel Version(Num_threads = 8)

Figure 3: Plots of R1-ES serial and parallel version for Cigar Function



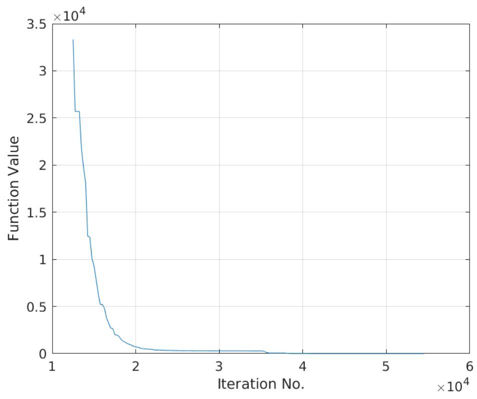
(a) Serial Version



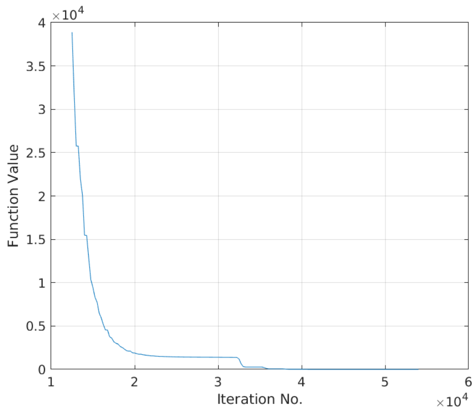
(b) Parallel Version(Num_threads = 8)

Figure 4: Plots of Rm-ES serial and parallel version for Cigar Function

Ctb: $f_{Ctb}(x) = x_1^2 + 10^6 \sum_{i=2}^{n-1} x_i^2 + x_n^2$

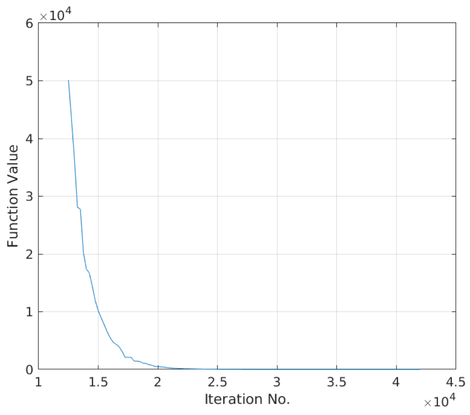


(a) Serial Version

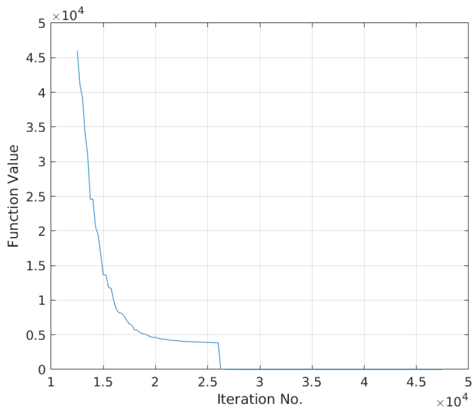


(b) Parallel Version(Num_threads = 8)

Figure 5: Plots of R1-ES serial and parallel version for CTB Function



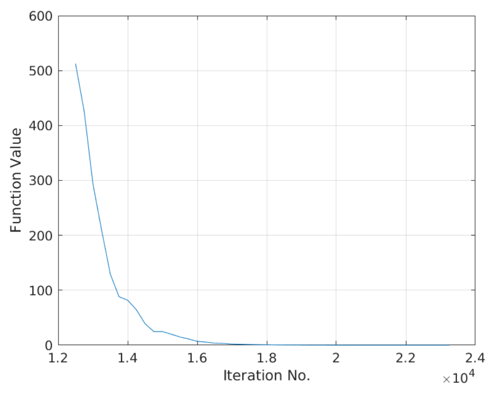
(a) Serial Version



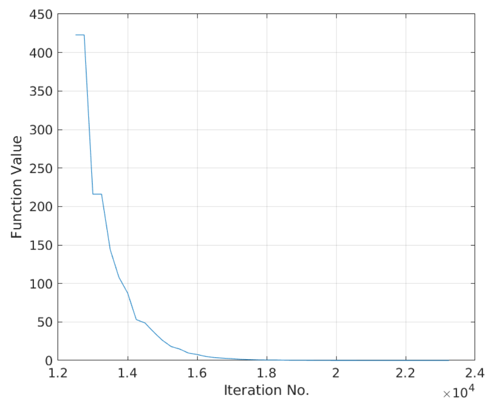
(b) Parallel Version(Num_threads = 8)

Figure 6: Plots of Rm-ES serial and parallel version for CTB Function

Schwefel: $f_{Sch}(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_i)^2$

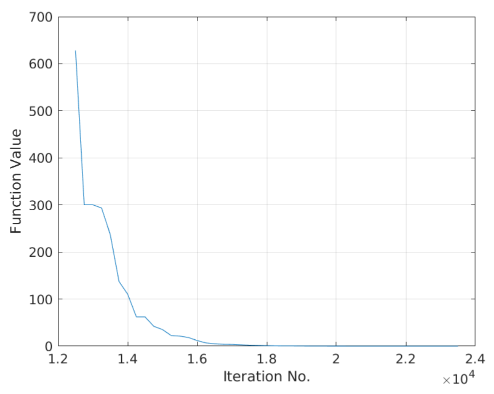


(a) Serial Version

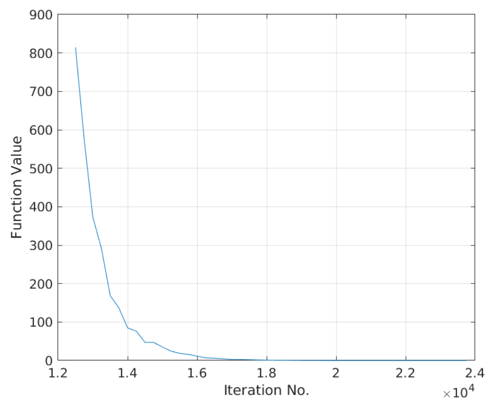


(b) Parallel Version(Num_threads = 8)

Figure 7: Plots of R1-ES serial and parallel version for Schwefel Function



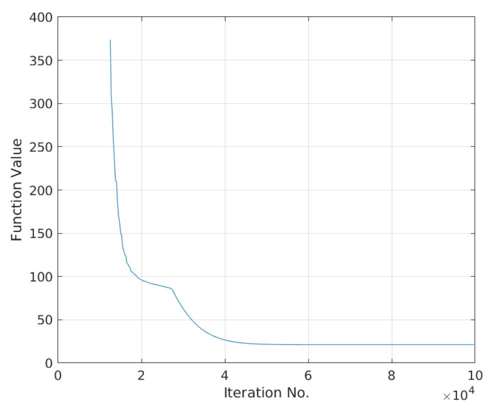
(a) Serial Version



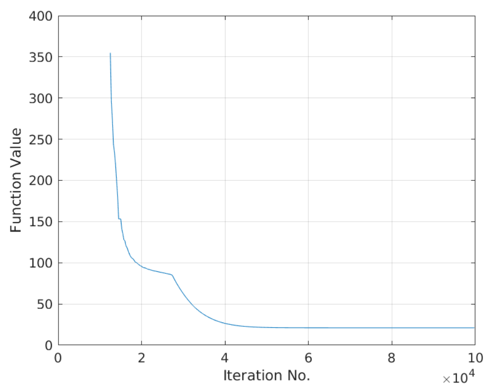
(b) Parallel Version(Num_threads = 8)

Figure 8: Plots of Rm-ES serial and parallel version for Schwefel Function

Rosenbrock: $f_{Ros}(x) = \sum_{i=1}^{n}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$

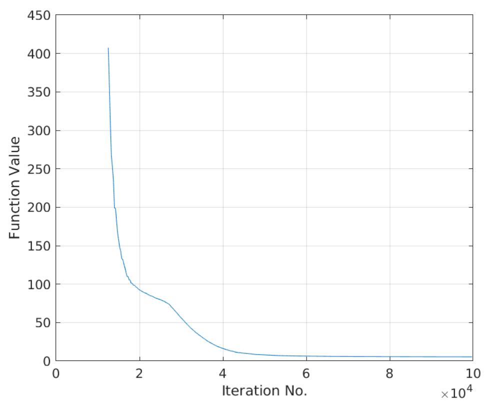


(a) Serial Version

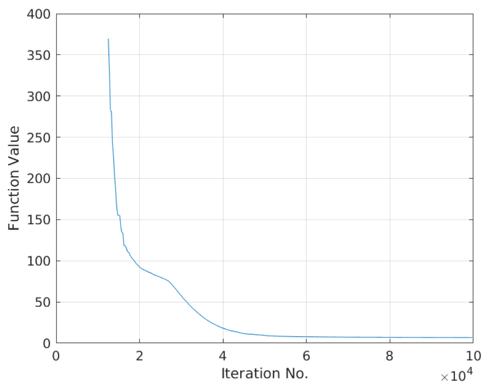


(b) Parallel Version(Num_threads = 8)

Figure 9: Plots of R1-ES serial and parallel version for Rosenbrock Function



(a) Serial Version

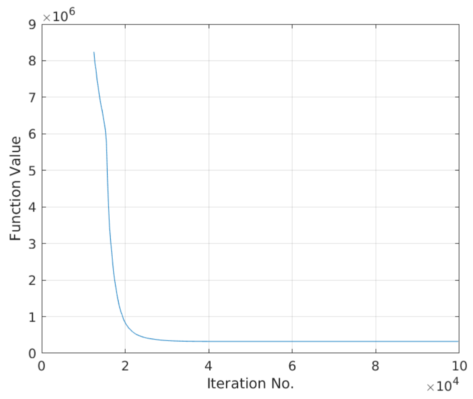


(b) Parallel Version(Num_threads = 8)

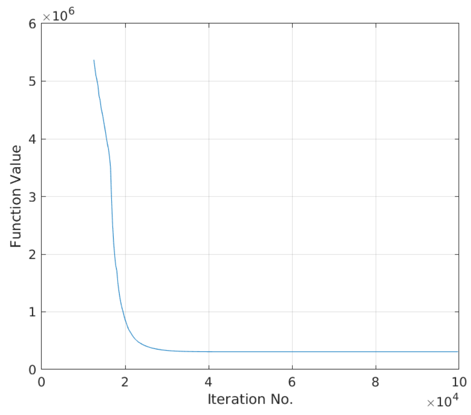
Figure 10: Plots of Rm-ES serial and parallel version for Rosenbrock Function

5.2 200 Variables:

Ellipsoid: $f_{Ell}(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$

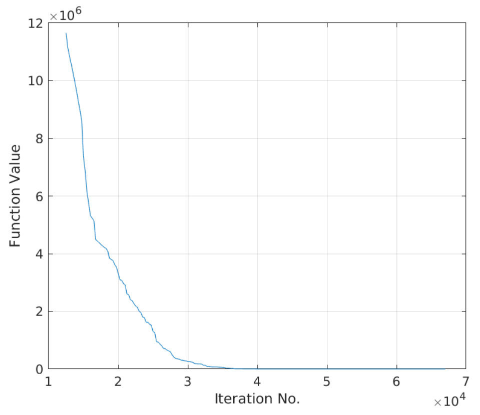


(a) Serial Version

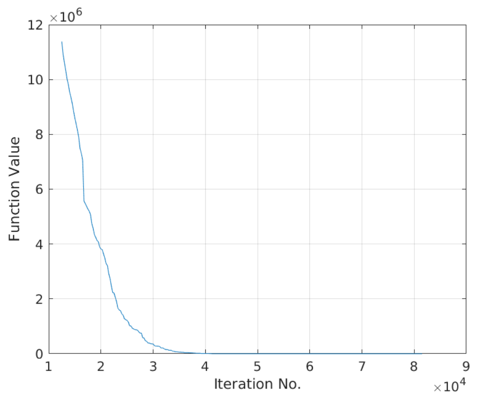


(b) Parallel Version(Num_threads = 8)

Figure 11: Plots of R1-ES serial and parallel version for Elliptic Function



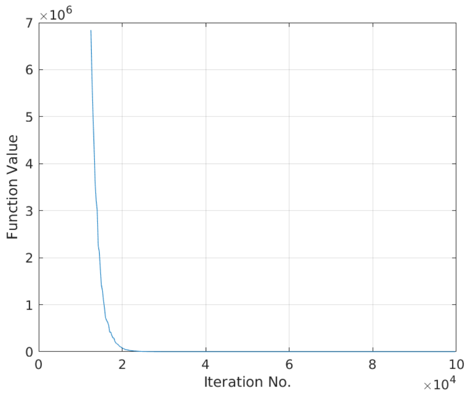
(a) Serial Version



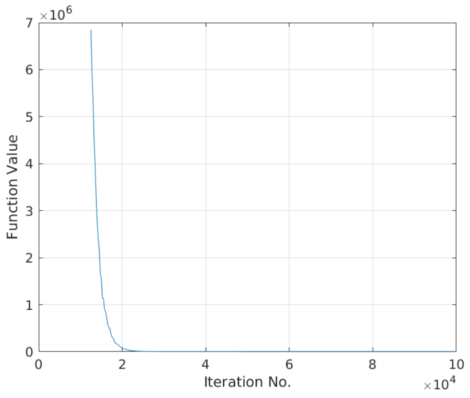
(b) Parallel Version(Num_threads = 8)

Figure 12: Plots of Rm-ES serial and parallel version for Elliptic Function

Cigar: $f_{Cig}(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$

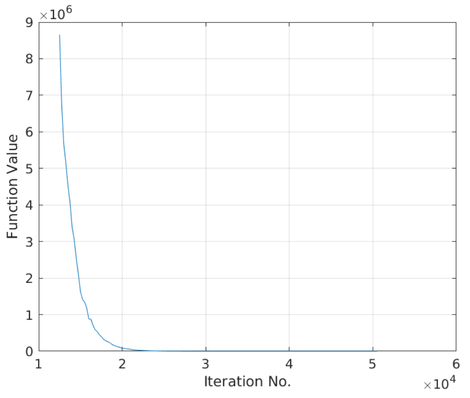


(a) Serial Version

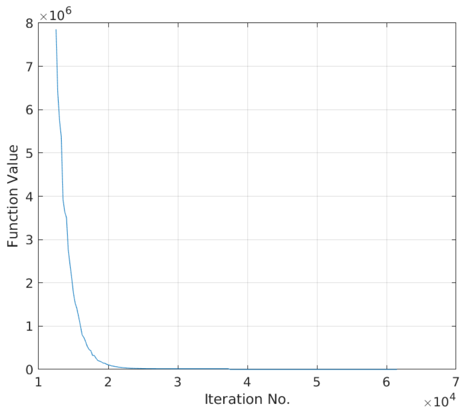


(b) Parallel Version(Num_threads = 8)

Figure 13: Plots of R1-ES serial and parallel version for Cigar Function



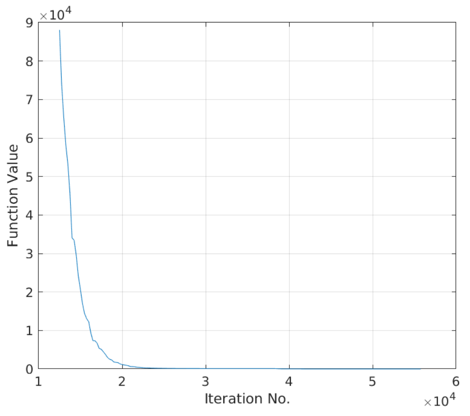
(a) Serial Version



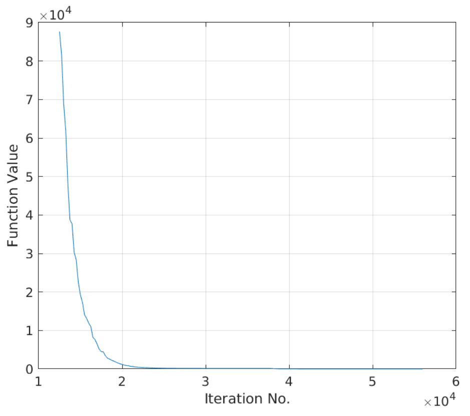
(b) Parallel Version(Num_threads = 8)

Figure 14: Plots of Rm-ES serial and parallel version for Cigar Function

Ctb: $f_{Ctb}(x) = x_1^2 + 10^6 \sum_{i=2}^{n-1} x_i^2 + x_n^2$

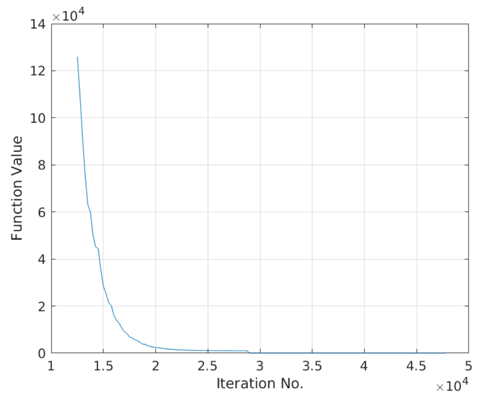


(a) Serial Version

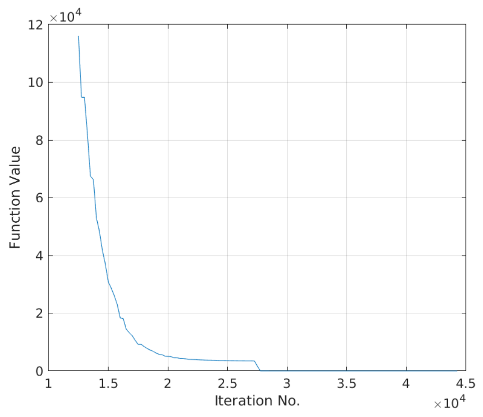


(b) Parallel Version(Num_threads = 8)

Figure 15: Plots of R1-ES serial and parallel version for CTB Function



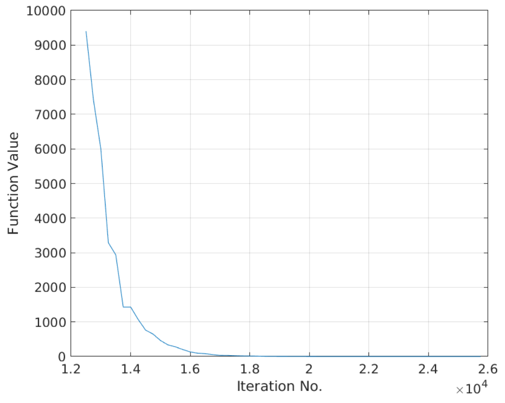
(a) Serial Version



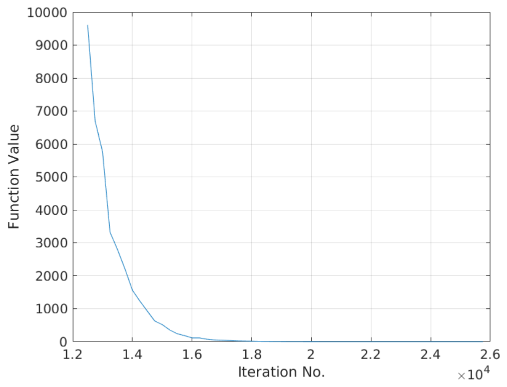
(b) Parallel Version(Num_threads = 8)

Figure 16: Plots of Rm-ES serial and parallel version for CTB Function

Schwefel: $f_{Sch}(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_i)^2$

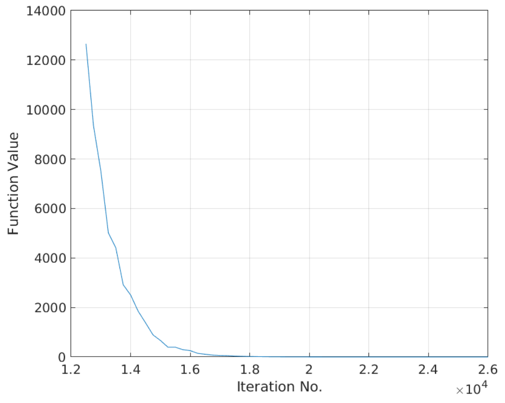


(a) Serial Version

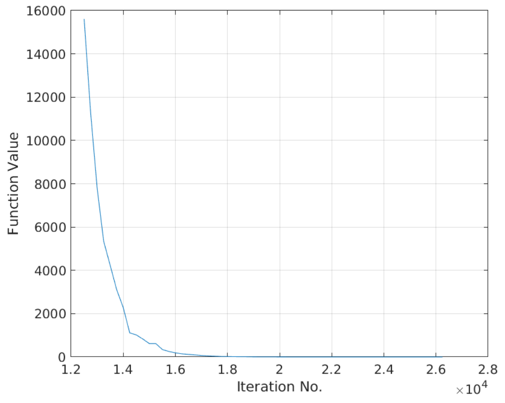


(b) Parallel Version(Num_threads = 8)

Figure 17: Plots of R1-ES serial and parallel version for Schwefel Function



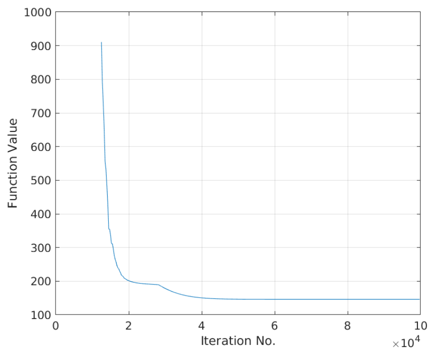
(a) Serial Version



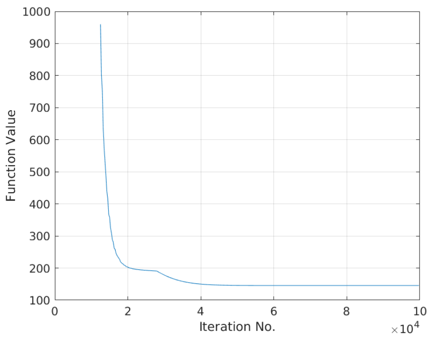
(b) Parallel Version(Num_threads = 8)

Figure 18: Plots of Rm-ES serial and parallel version for Schwefel Function

Rosenbrock: $f_{Ros}(x) = \sum_{i=1}^{n}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$

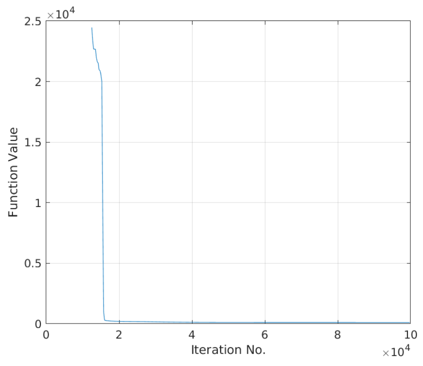


(a) Serial Version

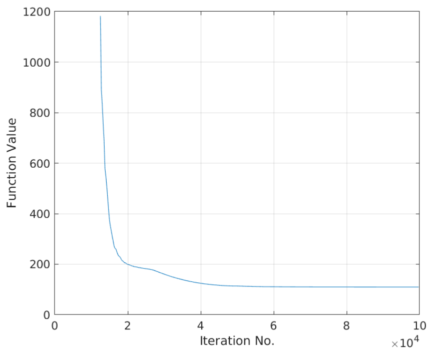


(b) Parallel Version(Num_threads = 8)

Figure 19: Plots of R1-ES serial and parallel version for Rosenbrock Function



(a) Serial Version

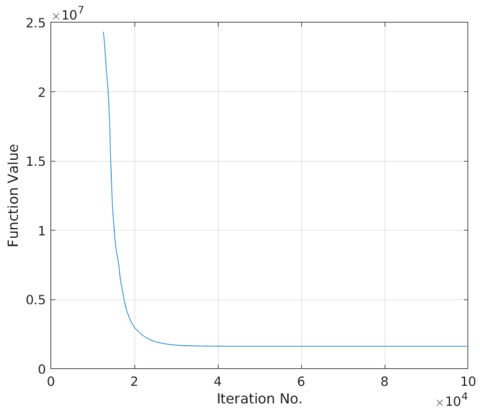


(b) Parallel Version(Num_threads = 8)

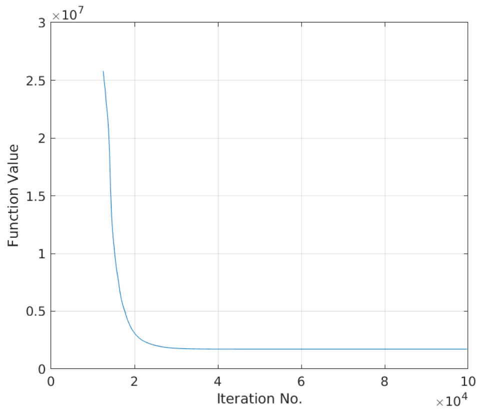
Figure 20: Plots of Rm-ES serial and parallel version for Rosenbrock Function

5.3 500 Variables:

Ellipsoid: $f_{Ell}(x) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$

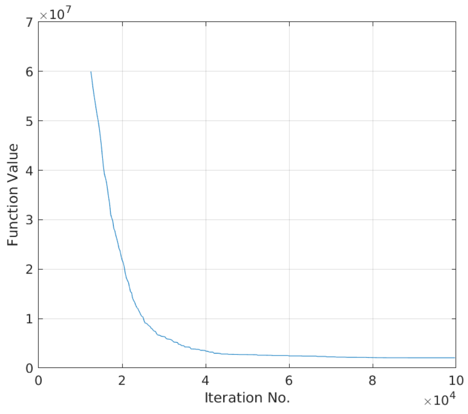


(a) Serial Version

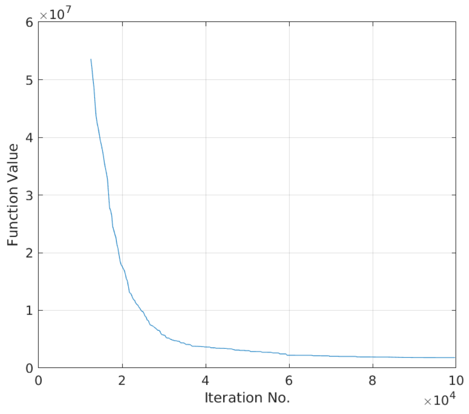


(b) Parallel Version(Num_threads = 8)

Figure 21: Plots of R1-ES serial and parallel version for Ellipsoid Function



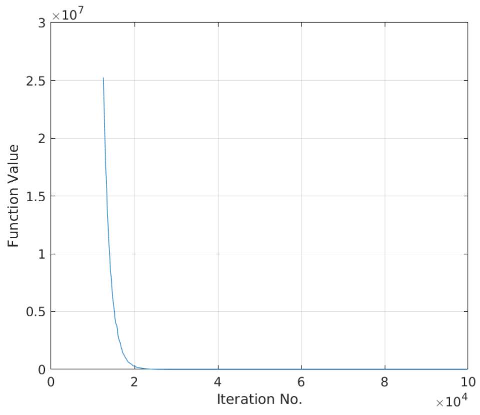
(a) Serial Version



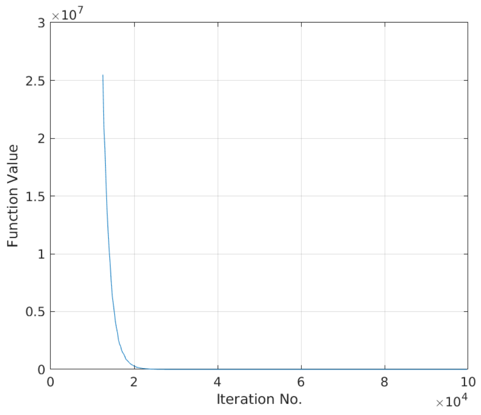
(b) Parallel Version(Num_threads = 8)

Figure 22: Plots of Rm-ES serial and parallel version for Ellipsoid Function

Cigar: $f_{Cig}(x) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$

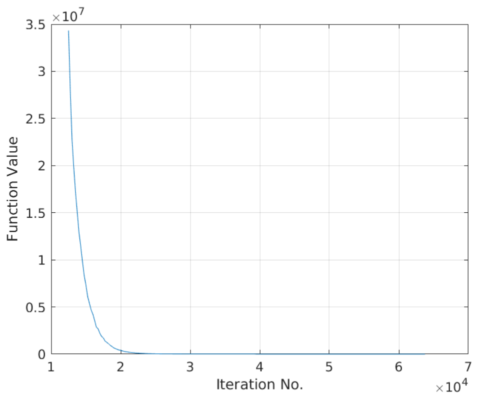


(a) Serial Version

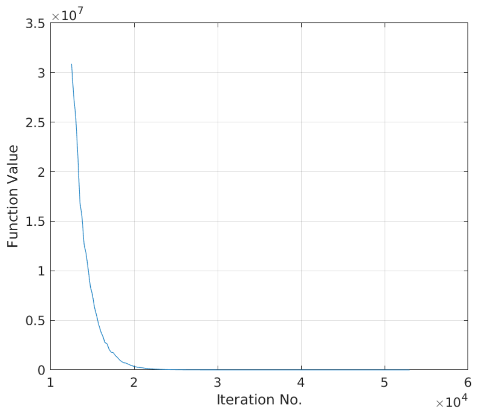


(b) Parallel Version(Num_threads = 8)

Figure 23: Plots of R1-ES serial and parallel version for Cigar Function



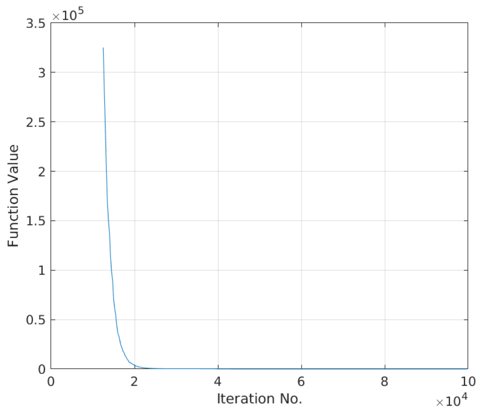
(a) Serial Version



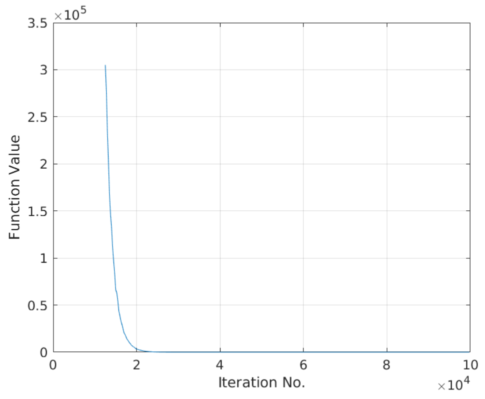
(b) Parallel Version(Num_threads = 8)

Figure 24: Plots of Rm-ES serial and parallel version for Cigar Function

Ctb: $f_{Ctb}(x) = x_1^2 + 10^6 \sum_{i=2}^{n-1} x_i^2 + x_n^2$

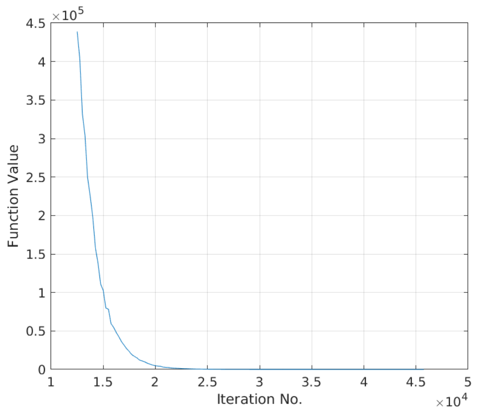


(a) Serial Version

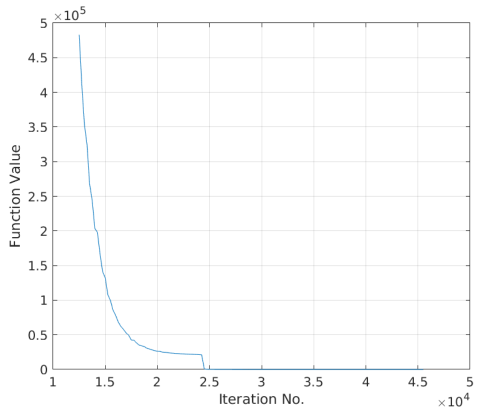


(b) Parallel Version(Num_threads = 8)

Figure 25: Plots of R1-ES serial and parallel version for CTB Function



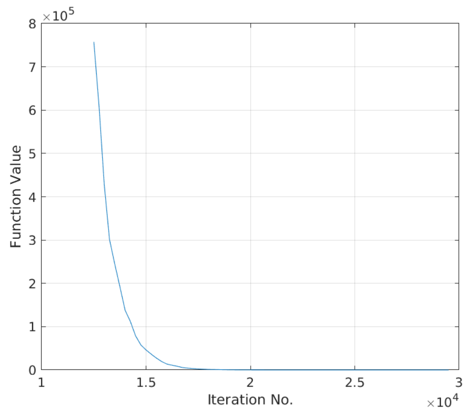
(a) Serial Version



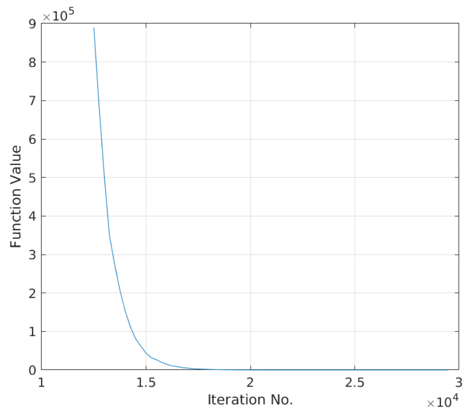
(b) Parallel Version(Num_threads = 8)

Figure 26: Plots of Rm-ES serial and parallel version for CTB Function

Schwefel: $f_{Sch}(x) = \sum_{i=1}^{n}(\sum_{j=1}^{i} x_i)^2$

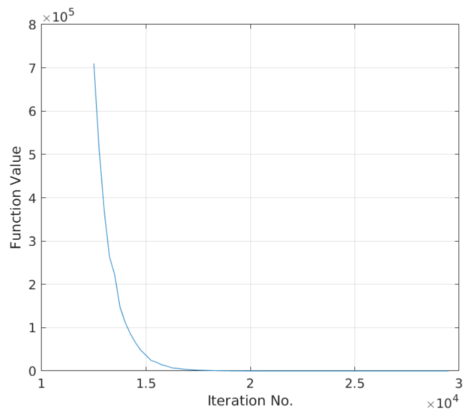


(a) Serial Version

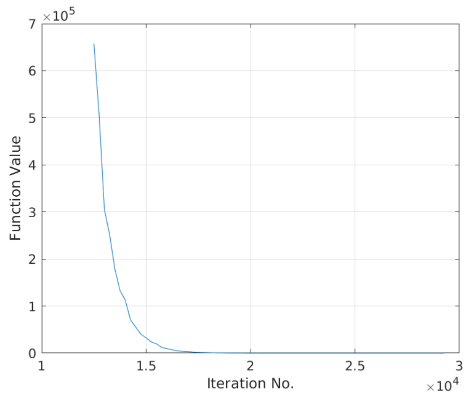


(b) Parallel Version(Num_threads = 8)

Figure 27: Plots of R1-ES serial and parallel version for Schwefel Function



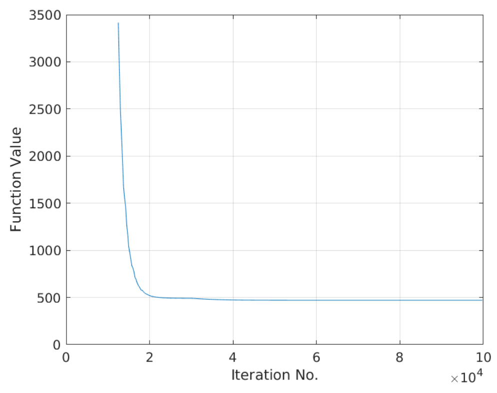
(a) Serial Version



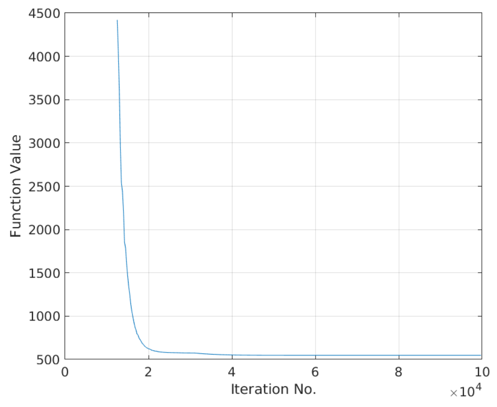
(b) Parallel Version(Num_threads = 8)

Figure 28: Plots of Rm-ES serial and parallel version for Schwefel Function

Rosenbrock: $f_{Ros}(x) = \sum_{i=1}^{n}(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$

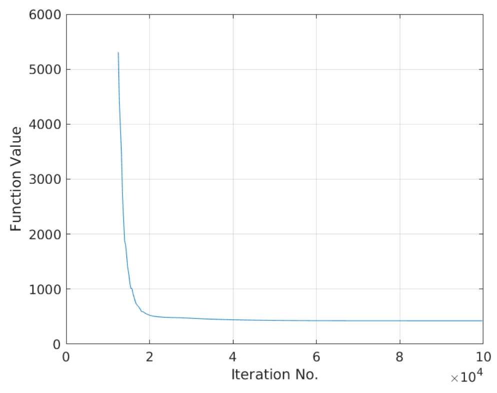


(a) Serial Version

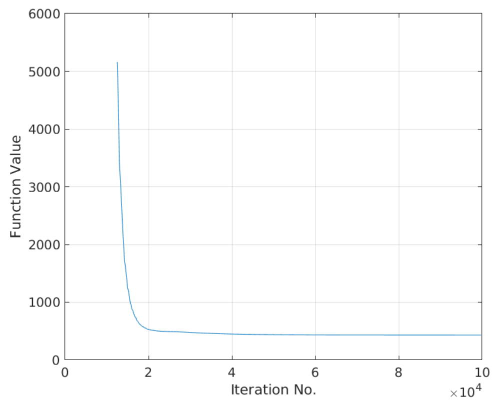


(b) Parallel Version(Num_threads = 8)

Figure 29: Plots of R1-ES serial and parallel version for Rosenbrock Function



(a) Serial Version



(b) Parallel Version(Num_threads = 8)

Figure 30: Plots of Rm-ES serial and parallel version for Rosenbrock Function

6 Speed Comparision:

6.1 100 Variables:

Table 1: Speed Comparison for N=100

(a) R1-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	113.197	123.75
f_{Cig}	52.9752	34.1505
f_{Ctb}	29.4711	18.602
f_{Sch}	74.7923	115.54
f_{Ros}	67.0875	51.7013

(b) Rm-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	85.3581	62.1638
f_{Cig}	47.7863	36.8178
f_{Ctb}	41.7953	28.672
f_{Sch}	86.7528	69.9847
f_{Ros}	112.523	75.3664

6.2 200 Variables:

Table 2: Speed Comparison for N=200

(a) R1-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	228.969	274.717
f_{Cig}	88.1392	53.0064
f_{Ctb}	49.7763	29.744
f_{Sch}	343.734	383.177
f_{Ros}	121.576	105.76

(b) Rm-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	241.016	146.11
f_{Cig}	119.691	54.9889
f_{Ctb}	85.8115	47.4142
f_{Sch}	376.932	313.638
f_{Ros}	376.932	137.271

6.3 500 Variables:

Table 3: Speed Comparison for N=500

(a) R1-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	650.866	670.531
f_{Cig}	224.147	139.555
f_{Ctb}	224.066	138.614
f_{Sch}	2908.25	2872.68
f_{Ros}	339.868	256.266

(b) Rm-ES

Function	Serial(s)	Parallel(s)
f_{Ell}	931.573	713.621
f_{Cig}	300.34	138.845
f_{Ctb}	258.407	180.2
f_{Sch}	2906.57	2652.39
f_{Ros}	607.959	506.218

7 Speed Comparison of Cigar function with No. of Variables:

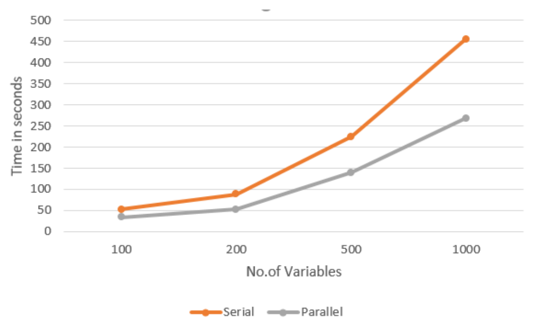
Table 4: Speed Comparison with No. of Variables[No. of Threads =8]

(a) R1-ES

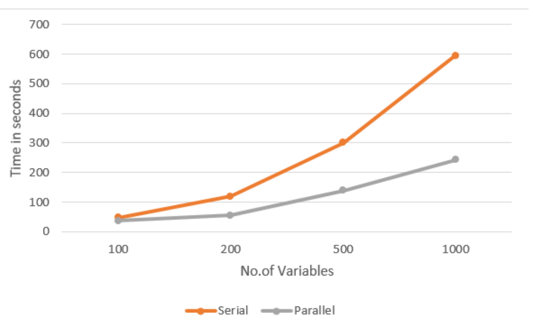
No. of Variables	Serial(s)	Parallel(s)
100	52.9752	34.1505
200	88.1392	53.0064
500	224.147	139.555
1000	455.539	267.862

(b) Rm-ES

No. of Variables	Serial(s)	Parallel(s)
100	47.7863	36.8178
200	119.691	54.9889
500	300.34	138.845
1000	594.387	243.384



(a) Serial Version



(b) Parallel Version(Num_threads = 8)

Figure 31: Trend of Cigar function with respect to number of variables.

8 Speed Comparison for Cigar function with No. of Threads:

Table 5: Speed Comparison with No. of Threads[No. of Variables =500]

(a) R1-ES

No. of Threads	Time(s)
1	259.893
2	144.797
4	98.7496
8	133.161
16	159.326
32	197.291
64	275.29
128	345.374
256	490.91
512	781.315

(b) Rm-ES

No. of Threads	Time(s)
1	355.603
2	152.096
4	119.029
8	142.013
16	132.077
32	150.791
64	193.582
128	268.785
256	355.679
512	587.937

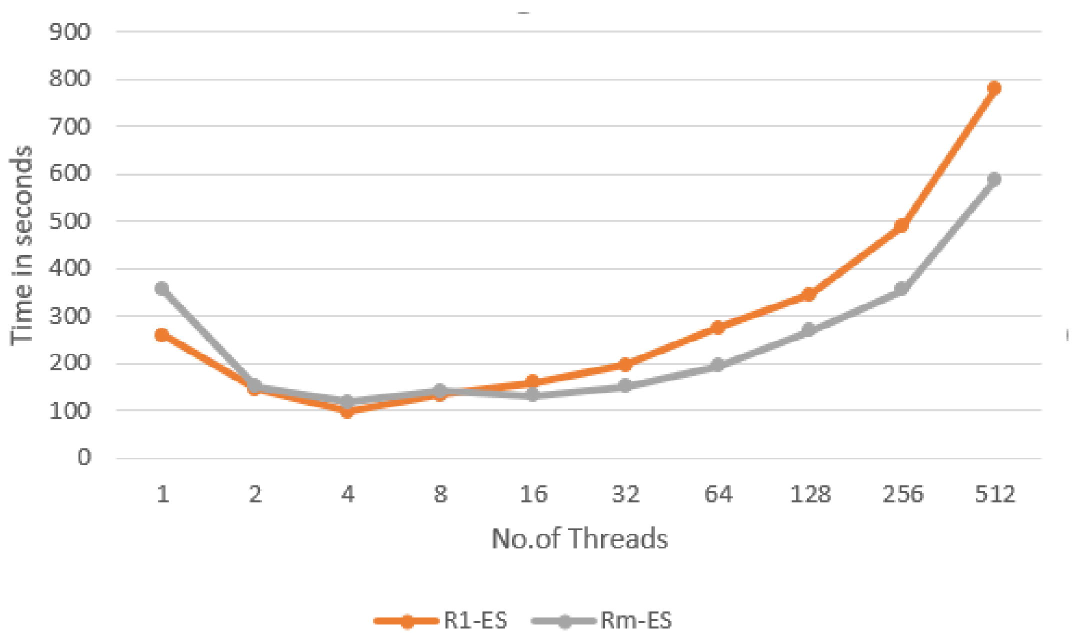


Figure 32: Trend of Cigar function with respect to number of Threads.

9 Conclusion:

The proposed R1-ES and Rm-ES algorithms use sparse and low rank decomposition of the co-variance matrix to perform computation in lesser time complexity. Single and multiple search direction are used in R1-ES and Rm-ES algorithm respectively to calculate the candidate solution. The principal search direction effectively sums up gradients in accordance to the distribution mean and thus plays a role of momentum. The algorithm used was robust, and the function value converged to a minimum in almost all the cases irrespective of the initialization. We achieved a speed-up by a factor of about 1.5 to 2 in most of the cases by parallelizing the algorithm using OpenMP.