Numerical Optimization

Dimitrios Nentidis

December 2023

1 Introduction

In this fourth assignment, the task was to implement stochastic algorithms to minimize the following function.

$$f(x) = \sum_{i=1}^{n-1} (x_i + x_{i+1} - 3)^2 + (x_i - x_{i+1} + 1)^4$$

For initial conditions of $X_0 = [2, 2, ..., 2]$.

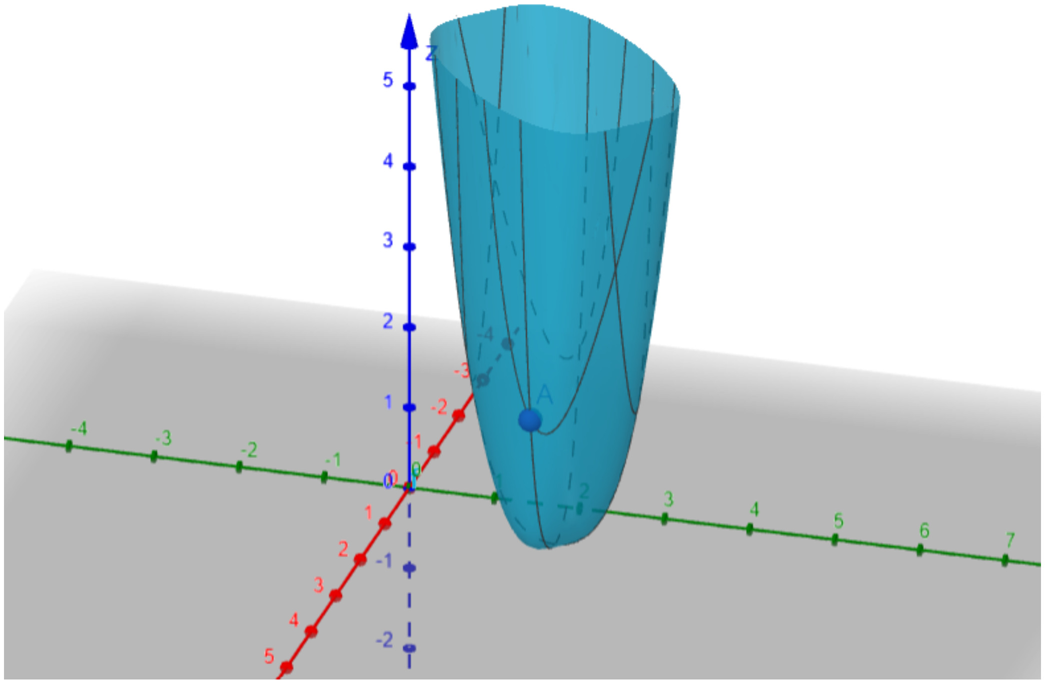

Figure 1: Generalized Tridiagonal 1 function, for n=2.

2 Deterministic approach

In the first assignment, this minimization problem was approached using deterministic, gradient-based algorithms, including Newton's method and its' modified version the Levenberg-Marquardt method. These algorithms

Method/Parameter	St. D.	CG	Newton's	L-M
Converged	$4.5881e-8$	$1.3037e-10$	No	$2.0947e-13$
Iterations	905	173	-	58
Iterations (Armijo)	3	25	-	-
Stable	Yes	Yes	-	Yes

Table 1: Performance Evaluation Table for $n = 2$.

Method/Parameter	Steepest D.	CG	Newton's	L- M
Converged	7.7487	3.3780	No	2.2787 (sym)
Iterations	990	83	-	-
Stable	Yes	No	-	-
Iterations (sym)	-	-	-	5
Stable (sym)	-	-	-	Yes

Table 2: Performance Evaluation Table for $n = 5$.

attempt to minimize the function taking steps to a certain direction, in the space where the function is defined. These steps depend on the way the function behaves in this space. The findings of this approach, found in the relevant report, are summed up in the following two tables.

3 Stochastic approach

Another way to approach this problem is using stochastic algorithms. These algorithms do not care about the dynamics of the function but rather use various techniques to evaluate the function in different, and random to a certain extent, points in the domain of interest.

Notably, these algorithms are sensitive to the area in which they are asked to find the minimum. If this area is not sufficiently bound, due to their stochastic nature they are not as effective.

4 Genetic Algorithm

In the case of the genetic algorithm, after the space in which the solution is estimated to reside is defined, an initial population of solutions is arbitrarily created. The members of the population are then evaluated and given a fit value, in this case, the value of the objective function. Then applying a similar process to evolution in nature, the fittest members of the population pair with each other, and offsprings are created.

To enable this, each member of the population must be encoded into a chromosome containing all the necessary information in binary form. The size of each chromosome is dependent on the desired accuracy and vice versa. One implication is that, the higher the dimension of the space, the more computational resources are required for the same accuracy.

This new generation of chromosomes, along with the surviving members of the previous generation, constitute the new population. Once the new generation is established, another evolution concept is applied, mutation. Here, in an unsystematic manner, a pre-defined percentage of the population is changed at a pre-defined rate. This is achieved by simply flipping the bits in a chromosome. Then the members are decoded to a new set of solutions for the problem and the circle continues until either some termination condition is met, or the number of generations is reached.

4.1 For $n = 2$

This algorithm was first implemented on a two parameter optimization problem, with an initial population size of 100 solutions. In general, there were no instances of divergence, and consistent performance was noted across multiple runs, indicating reliable convergence behavior. On average, the algorithm converged to the global minimum within 10 iterations. The order of magnitude of the solution was equal to $10^{(} - 6)$, comparable to the deterministic methods. The average time taken for the algorithm to converge to a solution was 10 seconds, significantly more than what the Levenberg-Marquardt method required.

The next two figures serve to illustrate the significance of the population size. On the left, the population size is equal to 100 potential solutions, whereas only 10 are present on the right. What can be observed is that the higher the population size the fewer iterations are required to solve the problem. Essentially the bigger population increases the possibility that some members will be in close proximity to the minimum. Computation time is not significantly affected by the increase in population, since decreasing the population increases the generations required.

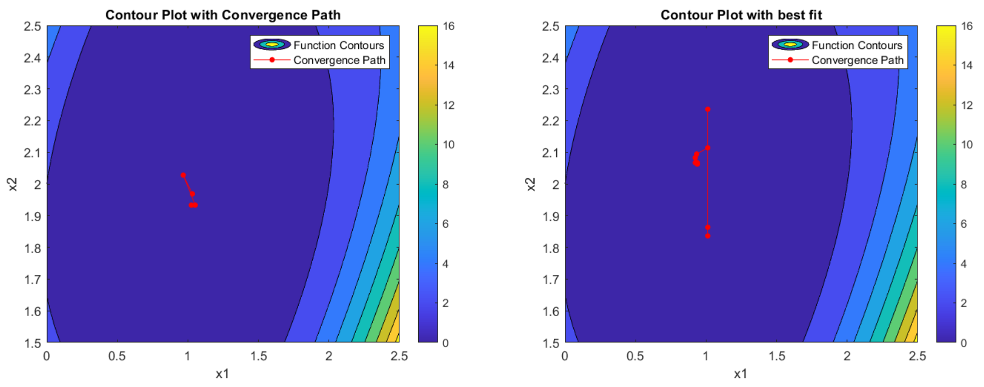

Figure 2: Contour plot and convergence path.

Another noteworthy observation is that the removal of the mutation part of the algorithm essentially pushes the best solution to the extremes. If the initial population is close to the minimum value, then it converges significantly faster, if not then the convergence takes more generations.

Lastly, it is evident that after some iterations, the objective value is very close to the minimum and it is not improving significantly.

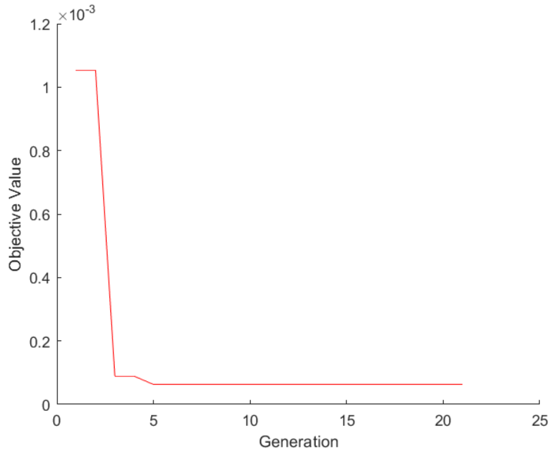

Figure 3: Histogram of Objective Value for $n = 2$.

4.2 For $n = 5$

The algorithm configured to run for five parameters remained stable. With an initial population size of 100 solutions, the routine needed 100 iterations on average to converge to a minimum of 2.2789 on point with the following coordinates $A = [1.0447, 1.3417, 1.5019, 1.6400, 2.0004, 2.2920]$. The required time was about 40 seconds on average.

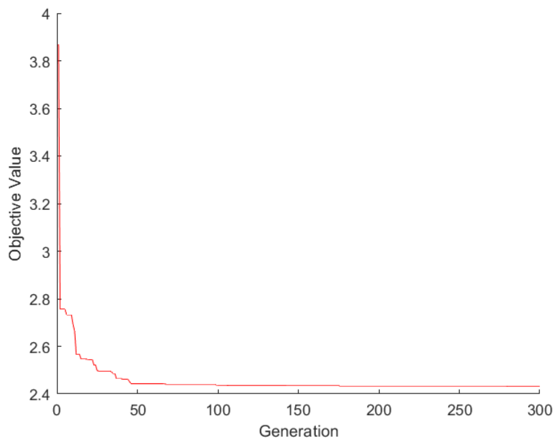

Figure 4: Histogram of Objective Value for $n = 5$.

4.3 Backstep problem

One interesting phenomenon noted during the implementation of the written Matlab code was that amidst the generations toward the best solution, there were some generations whose best member was less fit than the previous generation. This does not make sense since the fittest members of the previous population should survive in the next one. Turns out the problem was that the mutation was accidentally applied to the entire population, not only to the offsprings. Sometimes, in some new populations, the new generation did not outperform their best parent which, along with a mutation to the best parent, resulted in a worse performing best fit for the new population. Note that the average fit for the population improved with every iteration. The problem disappeared once this was changed.

5 Simulated Annealing

Simulated annealing is an optimization algorithm that draws inspiration from the physical process metals undergo during annealing. Initially, the metal atoms are in a high-energy state, moving freely and randomly. As they cool, they gradually settle into a more stable configuration. Analogously, the simulated annealing algorithm starts by exploring the solution space liberally, sometimes moving towards non-optimal states. As it progresses through iterations, it incrementally constrains this exploration, guiding the system towards an optimal solution based on predefined termination criteria.

The process begins with an initialization phase, where the algorithm sets up the initial parameters, including the starting temperature and the initial solution state. These settings are critical as they define the starting point and the conditions under which the algorithm operates.

At each iteration, the algorithm generates a new state by making small random variations to the current solution. This stochastic approach is essential for navigating the solution space and avoiding the trap of local optima. The Metropolis criteria come into play here, allowing the algorithm to accept both improvements and certain non-improving moves. This probabilistic acceptance is governed by the current temperature and the degree of worsening of the solution, thereby balancing exploration and exploitation.

The temperature of the system is systematically decreased according to a cooling schedule. This cooling influences the acceptance of new states; as the temperature drops, the algorithm becomes increasingly selective, preferring states that improve the objective function and thus inching towards convergence.

5.1 For $n = 2$

Implementing this algorithm for a two parameter optimization problem proved to be an efficient way to solve the problem. Regardless of the starting point, the routine was stable and converged to the global minimum, to an order of magnitude of $10^(- 31)$ in the maximum iterations available. This indicates that the termination criterion was not properly implemented. Despite that, the average convergence time was 18 seconds. As seen below, the solution path covers a significantly larger proportion of the solution space compared to either the genetic algorithm or any of the deterministic approaches.

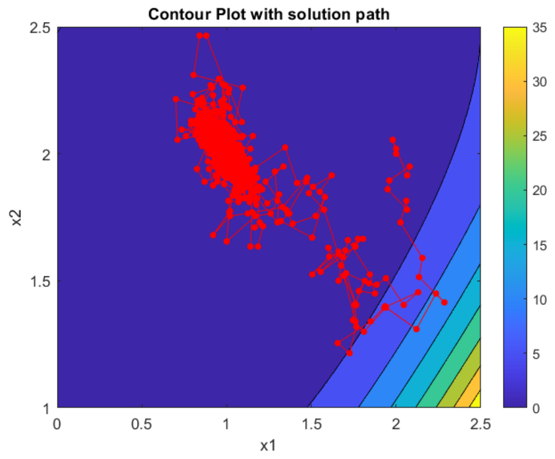

Figure 5: Solution path for Simulated Annealing and $n = 2$.

Note that the aforementioned behaviour of this approach is evident in the histogram below. The objective value is allowed to increase in the beginning, with less flexibility as the temperature drops.

5.2 For $n = 5$

A similar case is present here as well, the algorithm is stable, meaning that with initial conditions both closer and further away from the optimal value,

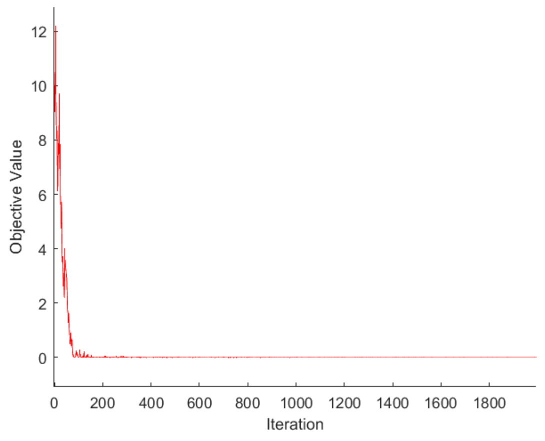

Figure 6: Objective Value Histogram $n = 2$.

it does not fail to converge. On average 750 iterations and 7.5 seconds are required for convergence. The reason this is less than the case with only two parameters is that the routine terminates early due to too many consecutive rejected moves. This method also converged to a similar point to the other five parameter solution approaches. Below the histogram can be found.

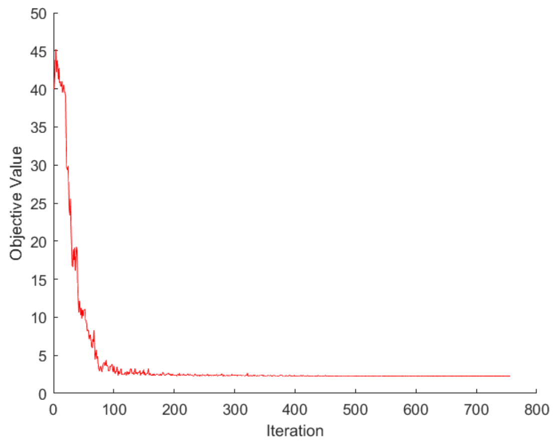

Figure 7: Objective Value Histogram $n = 5$.

It is evident that for both $n = 2andn = 5$ after a certain amount of iterations, the objective value is not significantly improving.

6 Conclusions

This report is essentially the second part of the first assignment report. Therefore the performance of these algorithms will be evaluated and compared to the deterministic approach algorithms.

Interestingly, both approaches converged to nearly identical solutions when tasked with optimizing both two-parameter and five-parameter problems. However, it became apparent that stochastic methods took considerably more time to arrive at the solution.

The primary reason for this increased time lies in the inherent design of stochastic algorithms. They operate without taking into account the gradient of the function, which is a measure of how the function behaves with changes in input values. This omission means that stochastic methods lack the nuanced guidance provided by the gradient, which deterministic methods use to direct their search for the optimum.

Despite this, stochastic algorithms are remarkably user-friendly. They require minimal mathematical background for implementation, sparing users from the complexities of calculus and derivatives. Their simplicity is very attractive when dealing with black-box functions—those where the internal workings are not known or accessible because they can adapt and search without needing explicit information about the function's behavior.

To mitigate the drawbacks of stochastic approaches, one could consider integrating behavioral insights into the evaluation process. Incorporating second-order information, such as the Hessian or the Jacobian, could provide a more informed search through the solution space, potentially reducing convergence times. This blend of stochastic flexibility and deterministic precision could lead to more efficient optimization strategies.

Lastly, it is very important to note that, the convergence of these algorithms heavily depends on the initial population or the initial point. One can not randomly select any initial point or spread the initial population in a sizable area. Doing so, will require either immense computational power, or will essentially guarantee that the solution will not be optimal. Taking this into consideration, even for black box functions, it is of decisive importance to narrow the search space down to a manageable area.