به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس: بهینه‌سازی و یادگیری توزیع‌شده

Course: Distributed Optimization and Learning

(پیاده‌سازی الگوریتم)

(Algorithm Implementation)

1تمرین شماره

نام و نام خانوادگی : گلمهر خسروخاور / Golmehr Khosrokhavar

شماره دانشجویی810198507 :

آبان ماه 1402

TABLE OF CONTENTS

Question 6: Part a

.1

In order to find the optimal step size, I am using the formula obtained by minimizing the quadratic form along the descent direction.

$$\alpha = \frac{g^T g}{g^T Q g}$$

In this question g, the gradient function, is calculated by:

$$f(x) = \frac{1}{2} x^T Q x + q^T x + p => g = Qx - q$$

Hence the optimal step size will be equal = 0.01991525

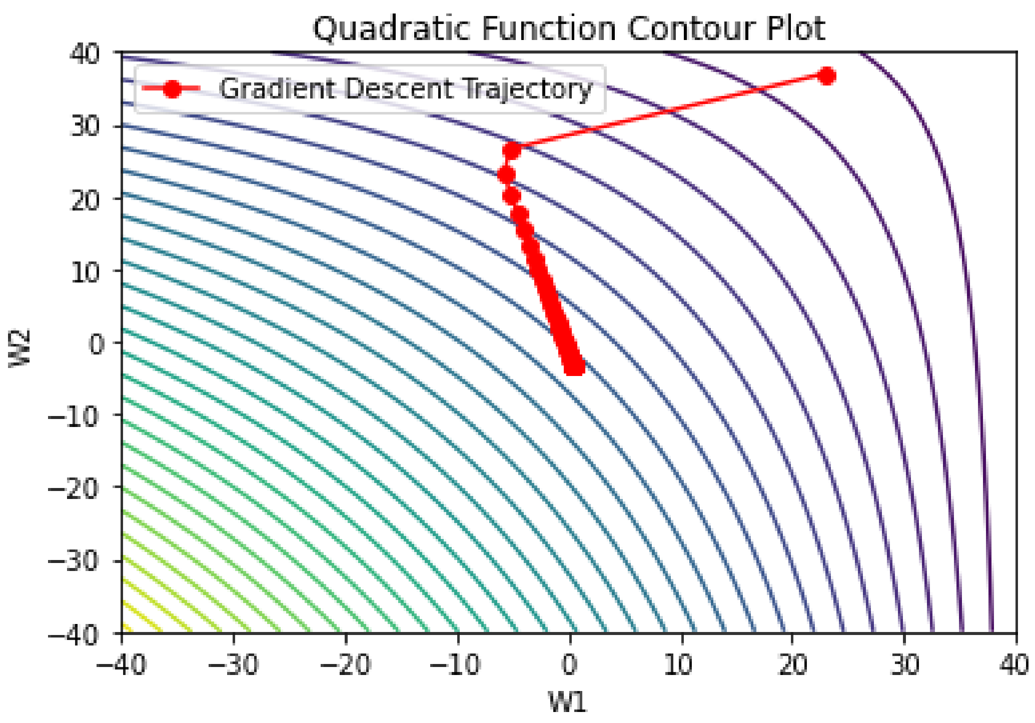


Figure 1

The minimum will occur at the point [0.441, -3.41] and the minimum value of f(x) equals to -32.556

It will converge after 71 iterations with this optimal step size.

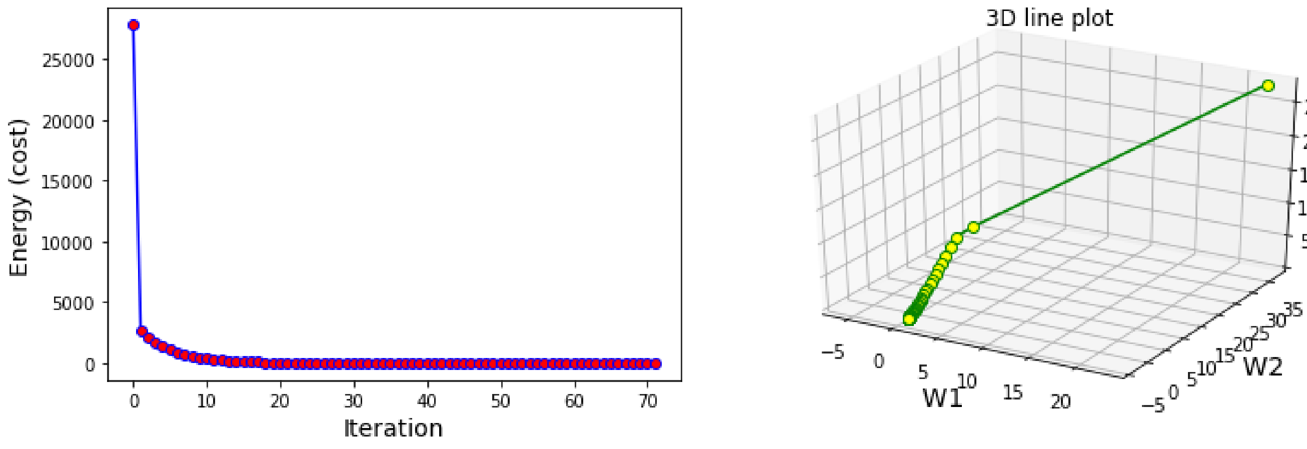


Figure2

.2 CONSTANT STEP SIZE USING GRADIENT DESCENT

- *for* $f_1(x)$:
  - Step size = 0.005 => it converges after 254 iterations to [0.4381695, 3.40631098] and the minimum value of the cost function is [-32.5547266]

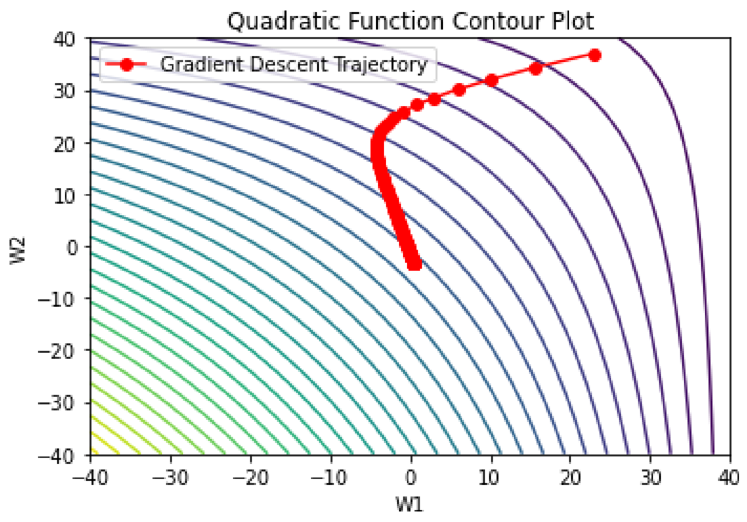


Figure3

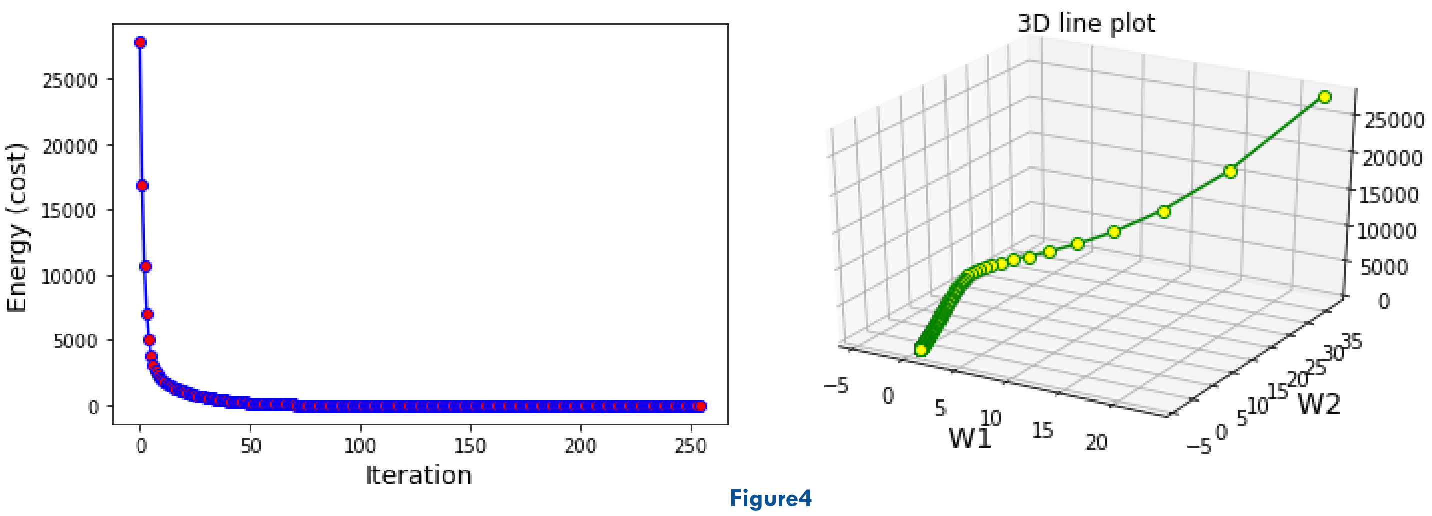


Figure4

- For step size = 0.05 => it doesnot converge. It can be clearly observed that this step size is not small enough to lead to the convergence of this algorithm.

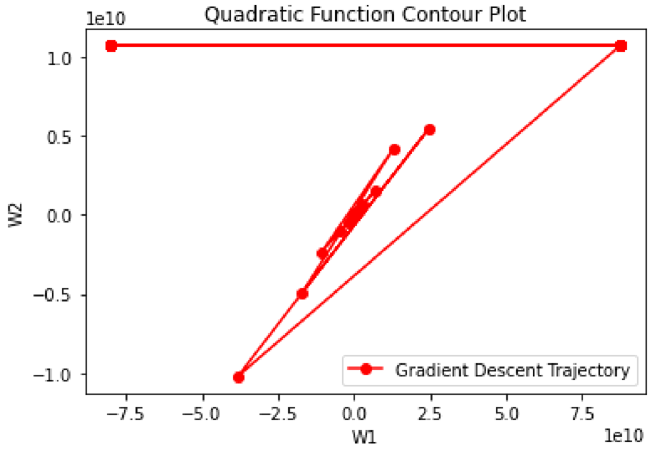


Figure5

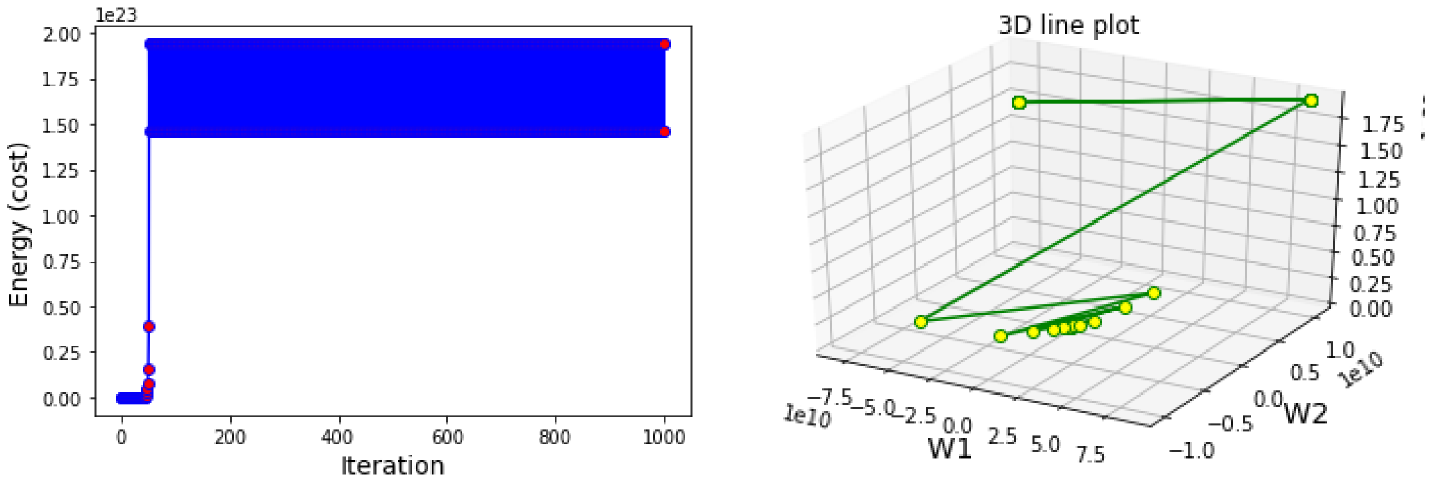


Figure6

- *for* $f_2(x)$:
  - step size = 0.0005 => it converges after 1370 iterations to 0.05636307 - 0.00991656] and the minimum value of the cost function is [3.9861474937]

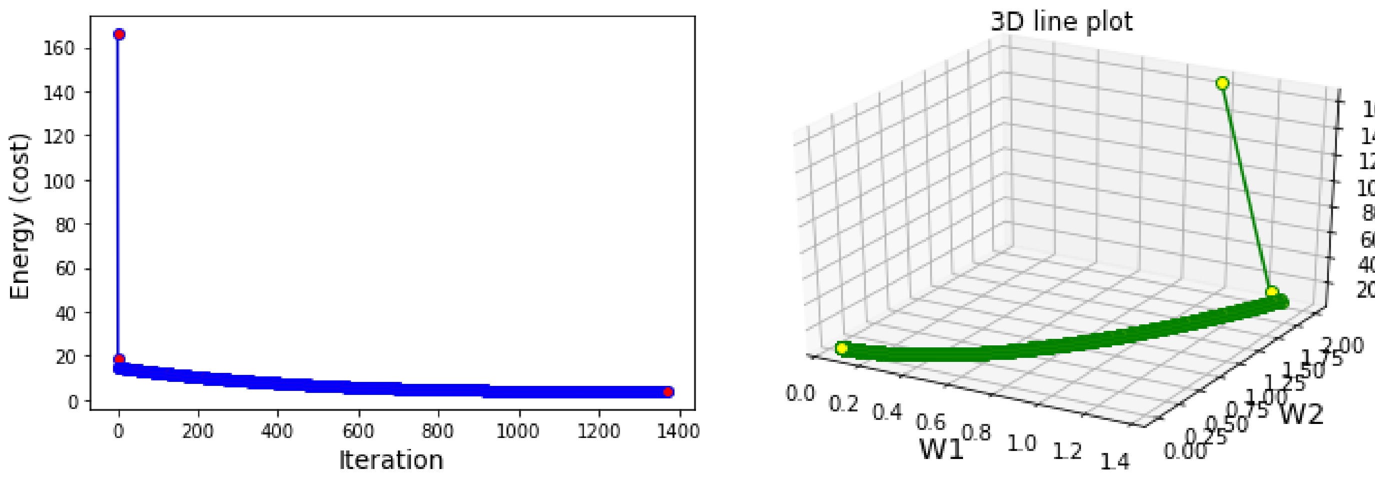


Figure7

- step size = 0.005 => it does not converge.

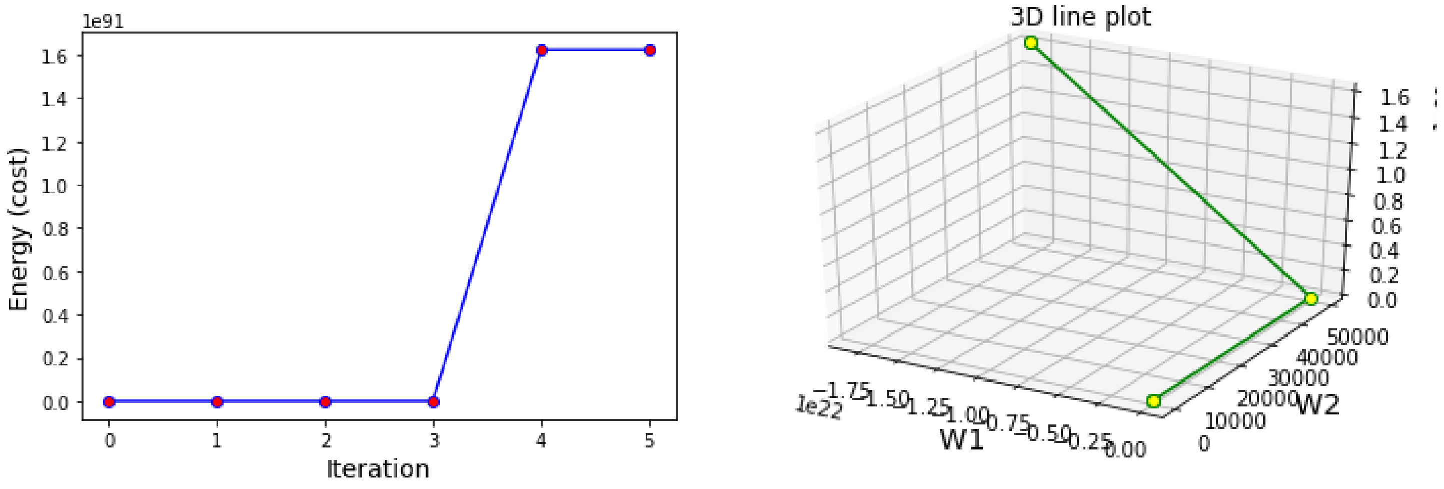


Figure8

- $for\ f_3(x)$:

  - Step size = 0.0009; it converges after 1367 iterations to [0.47296458 0.04887288 0.23220969 0.27882038] and the minimum value of the cost function equals to: [0.05513613971056858]

  - Step size = 0.005; it converges after 402 iterations to [0.33852634 , 0.03452016 , 0.17341403 , 0.19266478] and the minimum value of the the cost function equals to: [0.015937339911921297]

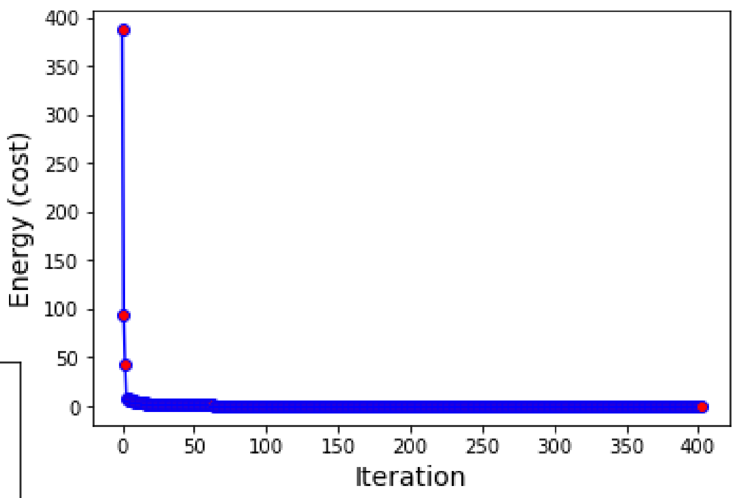


Figure9 : step size = 0.005

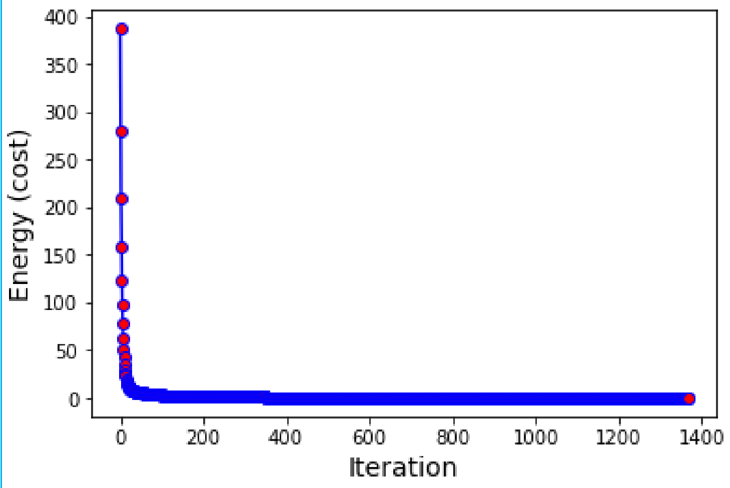


Figure10 : step size = 0.0009

3. BACKTRACKING LINE SEARCH

- $for\ f_1(x)$:
  - It converges after 29 iterations to [0.4428957 , -3.42573101] and the minimum value of the cost function is [-32.55629423]

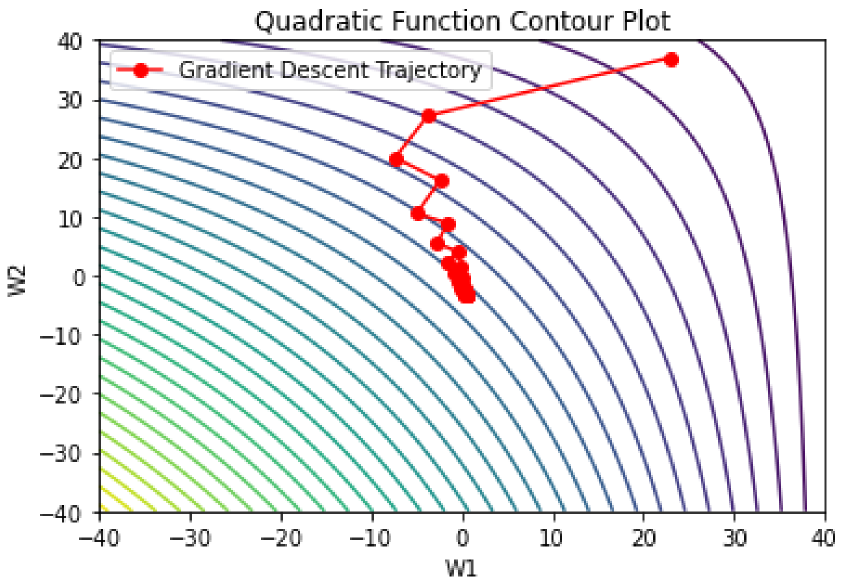


Figure11

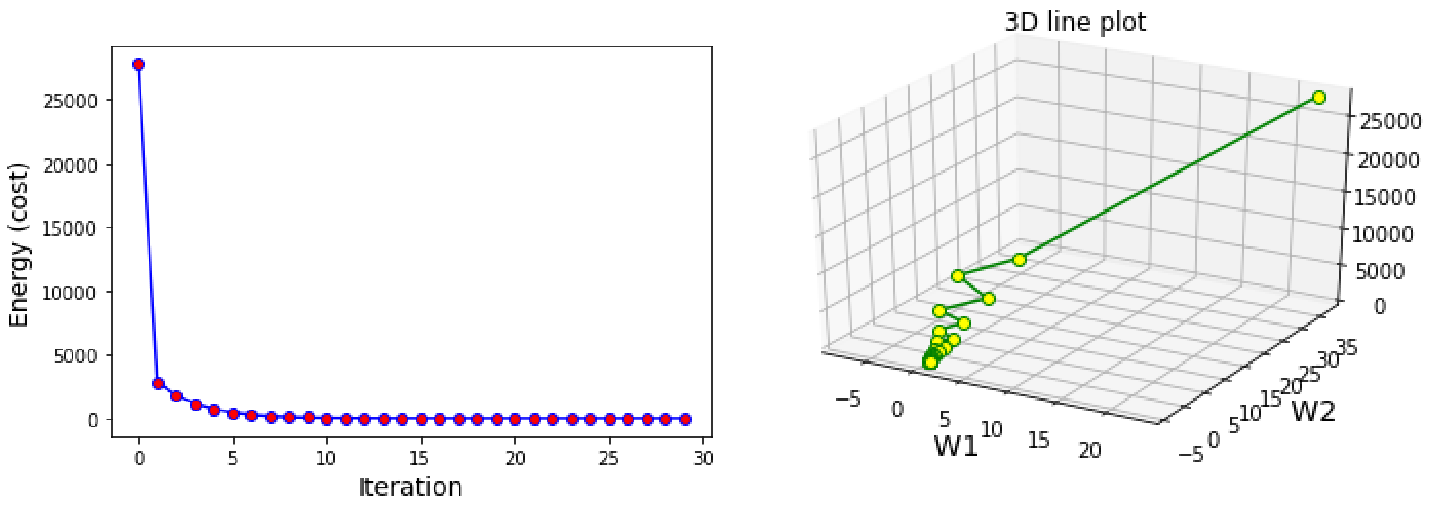
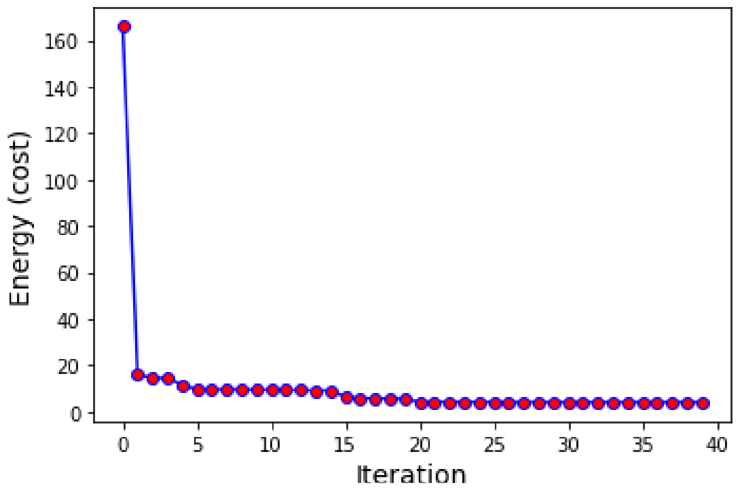


Figure12

It can be seen that this method has the fastest convergence among the two previous methods (which the convergence occurred at 71 and 250 iterations respectively.)

- $for\ f_2(x)$:
  - It converges after 39 iterations to the points: [0.00878322 -0.01322229] and the minimum value of the cost function equals to 3.97381691441792



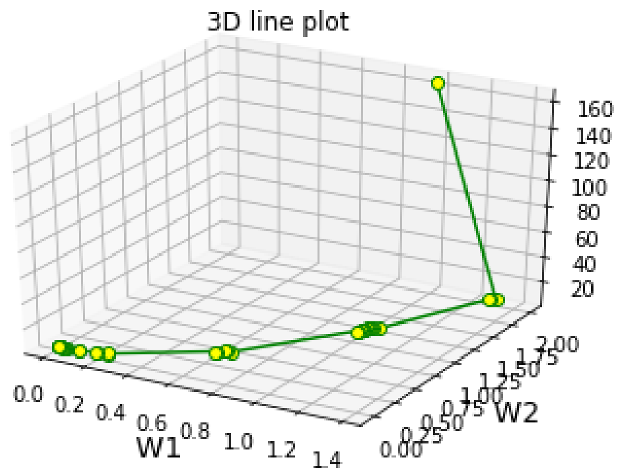


Figure13

- $for\ f_3(x)$: It converges after 58 iterations to the points: [0.27753186, 0.02829602, 0.14455642, 0.15565266] and the minimum value of the cost function equals to 0.007479129916526508 5

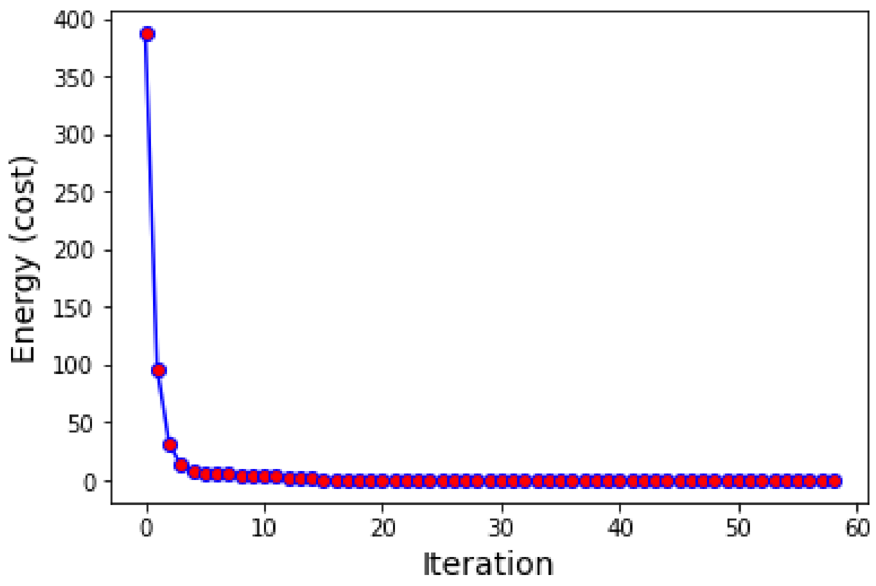


Figure14

4. NEWTON'S METHOD

- $for\ f_1(x)$:
  - The initial point is [23, 37]
  - It converges after 698 iterations to [0.43391609 -3.38829324] and the minimum value of the cost function is equal to [-32.55126442]

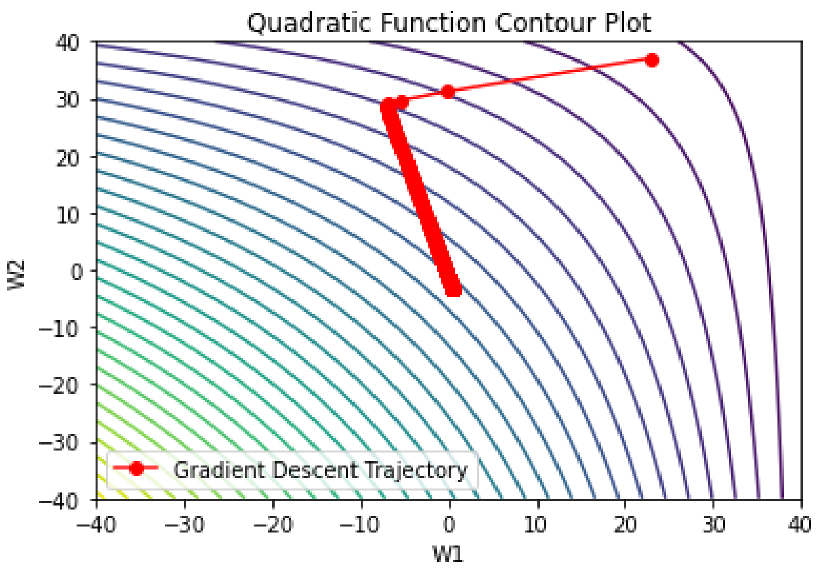


Figure15

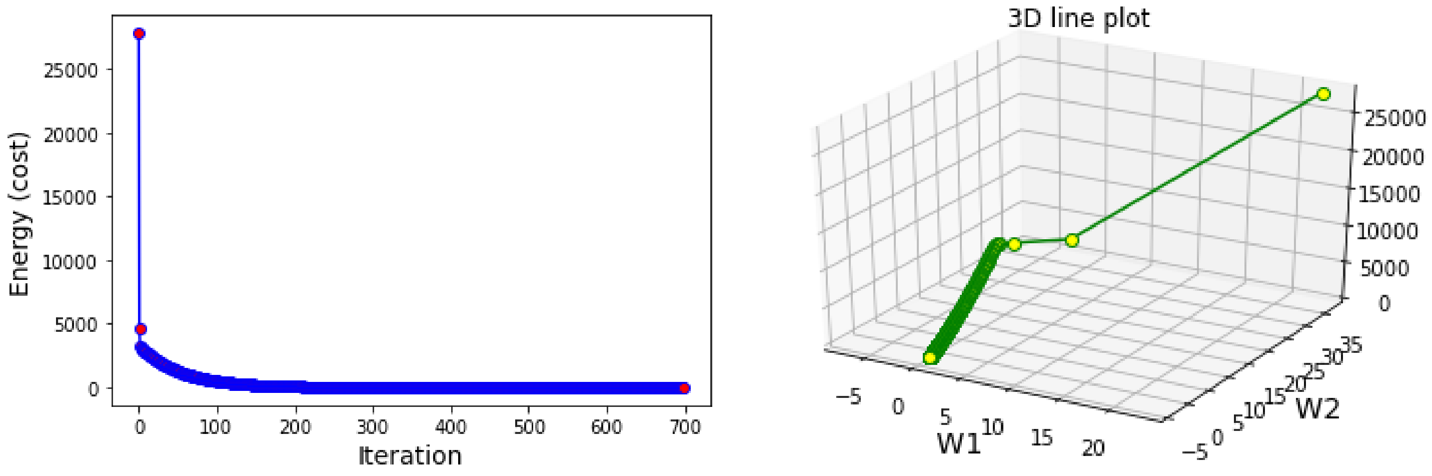


Figure16

  - Again, the initial point is [3, -3]
  - We can observe that the convergence at this initial point is much faster and occurs after 24 iterations. It converges to [0.47455459 -3.56044069] and the minimum value of the cost function is [-32.50535988]

For Newton's method we know that the sequence converges if 3 conditions are held and one of these conditions is about the initial point:
it declares: that x_0 has to be "sufficiently close" to x*.

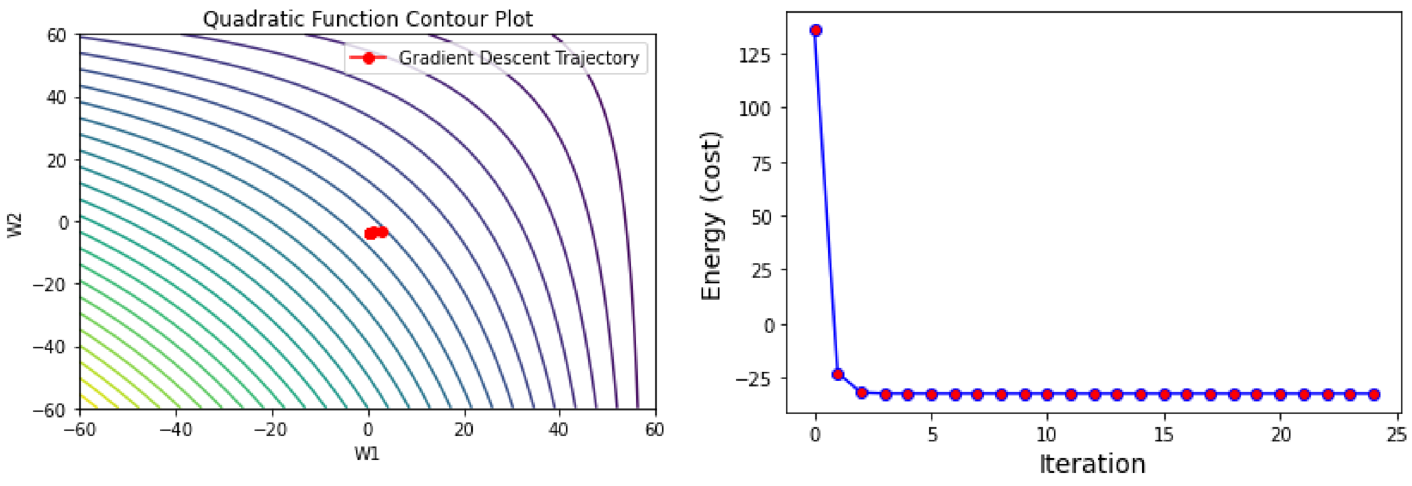


Figure17

- $for\ f_2(x)$:
    - The initial value is [0.8, -0.01]
    - It converges after 1297 iterations to [0.4699534, 0.20992063] and the minimum value of the cost function is 4.90168717487

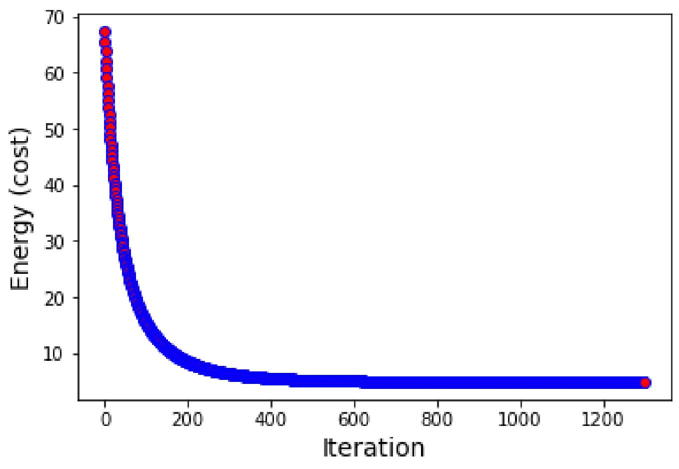


Figure18

- If the initial point is [21, 5], the algorithm will not converge since the condition about the initial value being sufficiently close to the optimal point is not held.

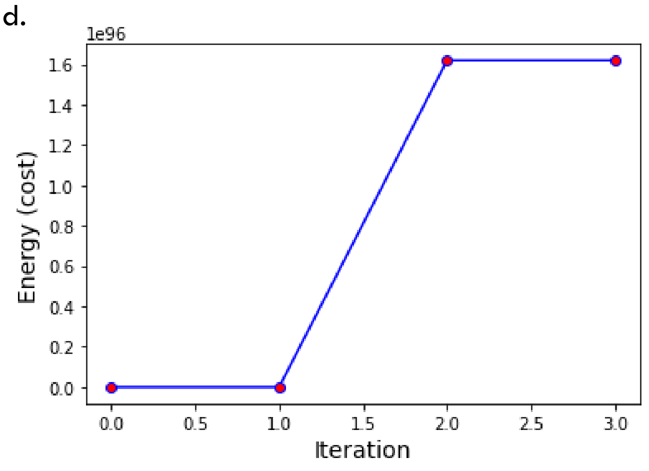


Figure19

- $for\ f_3(x)$:
  - The initial point is [1,0.8,0,1] and it will converge after 6000 iterations to [0.61766993 0.06473243 0.28865128 0.37814786] and the minimum value of the cost function is 0.1428677561309679

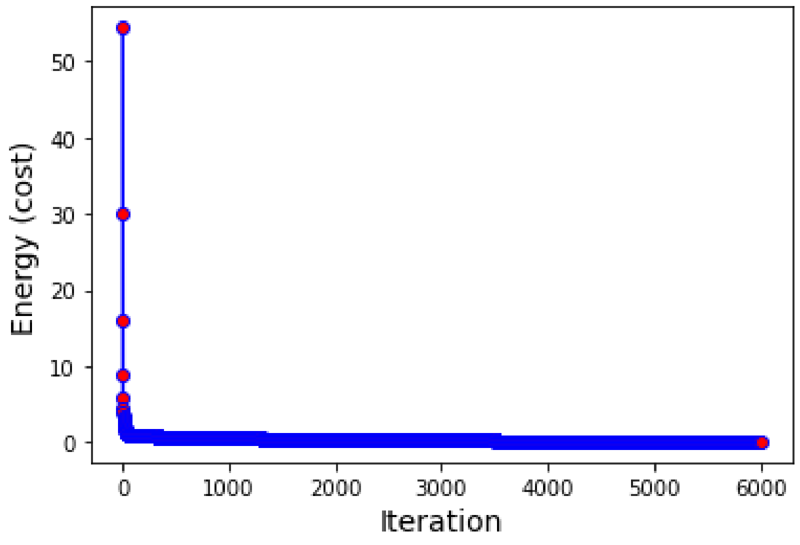


Figure20

  - If the initial point is [2 2 2 2] it will not converge!

5. STOCHASTIC GRADIENT DESCENT

Stochastic gradient descent algorithms are a modification of gradient descent. In stochastic gradient descent, you calculate the gradient using just a random small part of the observations instead of all of them. In some cases, this approach can reduce computation time.

As in the case of ordinary gradient descent, stochastic gradient descent starts with an initial vector of decision variables and updates it through several iterations. The difference between the two is in what happens inside the iterations:

- Stochastic gradient descent randomly divides the set of observations into minibatches.
- For each minibatch, the gradient is computed and the vector is moved.
- Once all mini-batches are used, you say that the iteration, or epoch, is finished and start the next one.

6. EMTIAZI

I have implemented my functions in a way to be adaptable with higher order inputs and they can be generalized.

7. EMTIAZI

We first have to check whether this quadratic function is convex or not.

We know from the lecture notes that the necessary and sufficient condition for a matrix [A] to be positive semi-definite is that the symmetric part of this matrix must be positive semi-definite. But as it can be observed, Q is not symmetric since 12≠2.092.

So, let's calculate the symmetric part of it:

$$Q_{sym} = \begin{pmatrix} 48 & 12 \\ 8 & 2.092 \end{pmatrix}; Q_{sym} = \begin{pmatrix} 48 & 10 \\ 10 & 2.092 \end{pmatrix}$$

$$Check\ if\ Q_{sym}\ is\ positive\ semi-definite =>$$

$$Q_{sym_{11}} = [48] \geq 0\,; Det(Q_{sym}) = 0.416 \geq 0$$

So, Q_sym is Positive Semi-definite.

The eigenvalues are approximately 50.084 and 0.0083. The bigger eigen value is approximately 6000 times bigger than the other one.

$$Finding\ the\ minimum\ point: f(x) = \frac{1}{2}x^TQx + q^Tx + p => \nabla f(x) = Qx + q = 0$$

$$x = -Q^{-1}\,q => Q^{-1} = \frac{1}{\det(Q)}\begin{pmatrix} 2.092 & -12 \\ -8 & 48 \end{pmatrix} =>$$

$$-Q^{-1}\,q = \frac{-1}{4.416}\begin{pmatrix} 2.092 & -12 \\ -8 & 48 \end{pmatrix}\begin{pmatrix} 13 \\ 23 \end{pmatrix} = \frac{-1}{4.416}\begin{pmatrix} -248.804 \\ 1076.804 \end{pmatrix} = \begin{pmatrix} 59.808653846 \\ -263.841615385 \end{pmatrix}$$

$$x^* = \begin{pmatrix} 59.808653846 \\ -263.841615385 \end{pmatrix}$$

Then we have to check if this point is global min or not. Since the function is quadratic and convex, and the Q matrix is PSD, the point x* is a global min indeed.

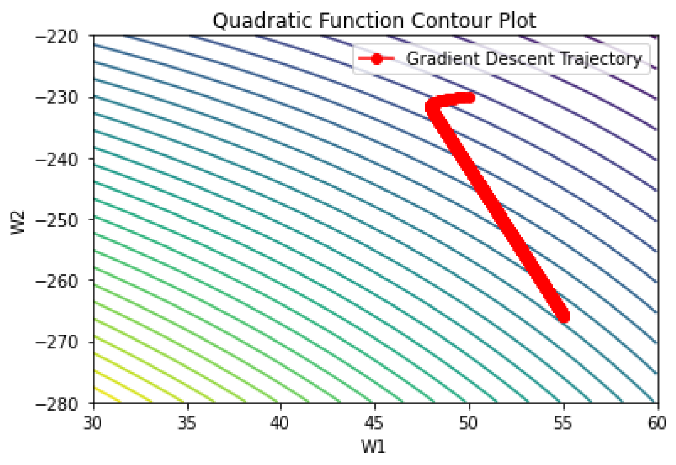
But the gradient descent algorithm won't converge to this point.

- <u>Reasons behind the problem of convergence:</u>
- If one eigenvalue of the matrix Q is 6000 times bigger than the other eigenvalue, it implies that Q is highly anisotropic, meaning it scales the space very differently along different directions. This situation has several implications, especially if Q is used in a quadratic function or an optimization problem:
  - The condition number of Q, which is the ratio of the largest eigenvalue to the smallest eigenvalue, would be very large (in this case, approximately 6000). A large condition number indicates that the matrix is ill-conditioned.

  - Optimization Challenges: In the context of optimization problems, such as those solved by gradient descent, a high condition number can lead to convergence issues. The optimization path would be elongated and narrow, resembling a steep valley or ridge. This makes the optimization algorithm sensitive to the choice of the learning rate and can lead to slow convergence or oscillations.

In order to solve this problem, I took 3 steps:

1. I used the Q_symm instead of the Q itself during my calculations. Using a symmetric matrix can help, especially if it's positive definite. (it is)
2. I used a smaller learning rate since If the learning rate is too high, the algorithm might overshoot the minimum; if it's too low, it might converge very slowly or get stuck.
3. I changed the initial value.



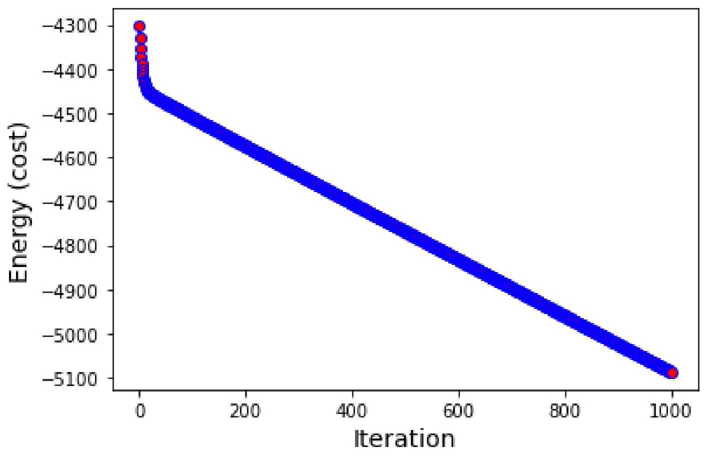


Figure21

It converges to the point [55.05374365 -265.91717153] which is considerably close to the minimum point .

PART B) MINIMIZINGUSING PROJECTED SUBGRADIENT

I followed the following steps to simulate this optimization problem in python:

1. Initialization of parameters
2. Generate X Samples: Create 10000 samples of X matrices in the range from (-500, -500, ..., -500) to (500, 500, ..., 500).
3. Finding X_Feasible: For each sample of X, check if it satisfies the condition
4. Calculating f(X): For each X in X_Feasible, calculate $|a_i^T X + b_i|$ for each i = 1, 2, ..., 500. Find the maximum of these values and associate this value to f(X) at this point. Also, record the index i which gives the maximum value for each X.
5. To solve the minimization problem for the function f(X) using the projected sub-gradient method, we will perform iterative updates on X using the sub-gradient of f(X) and a step size α_k

RESULTS:

The graphs illustrate the progression of the cost function value f(X) and the Euclidean norm of the difference between consecutive iterations $||X_{k+1} - X_k||$ for each step size rule over 1000 iterations.

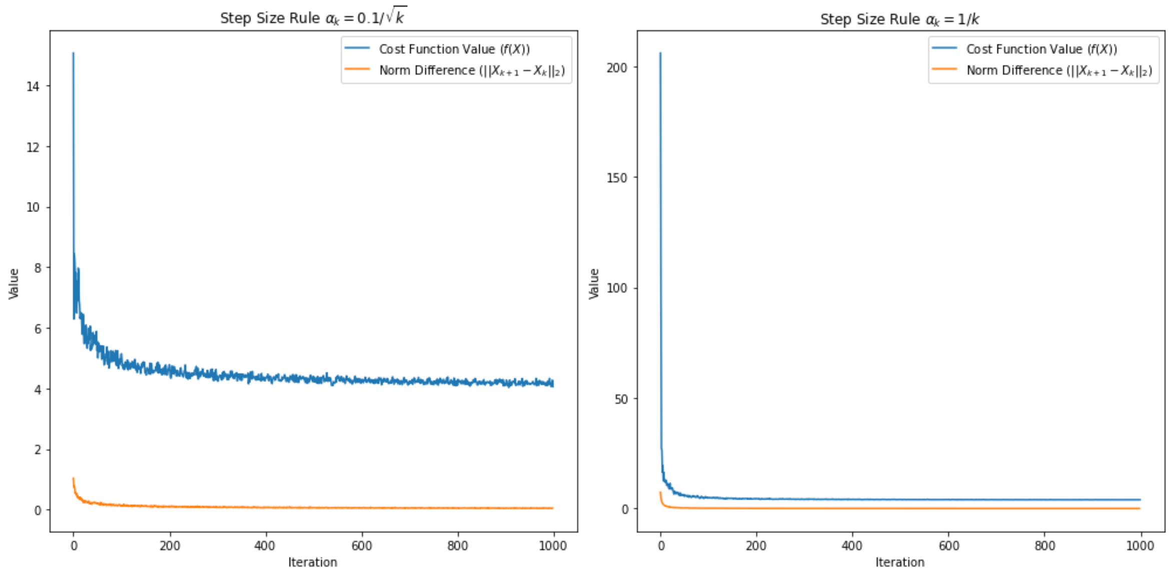


Figure 22

- For the step size rule $\alpha_k = \frac{0.1}{\sqrt{k}}$:

The cost function value decreases sharply initially and then stabilizes as the iterations progress, indicating that the method is making significant progress in reducing the cost and then refining around a minimal value. The norm difference also decreases sharply and flattens out, which suggests that the successive updates to X become smaller, and the algorithm is reaching a stable point.

- For the step size rule $\alpha_k = \frac{1}{k}$

A similar pattern is observed with the cost function value decreasing rapidly before flattening. This rule seems to stabilize slightly quicker than the rule, as indicated by the cost function graph. The norm difference again decreases quickly, showing a rapid reduction in the magnitude of updates to X, and then it levels off, implying convergence to a solution or a stable region.

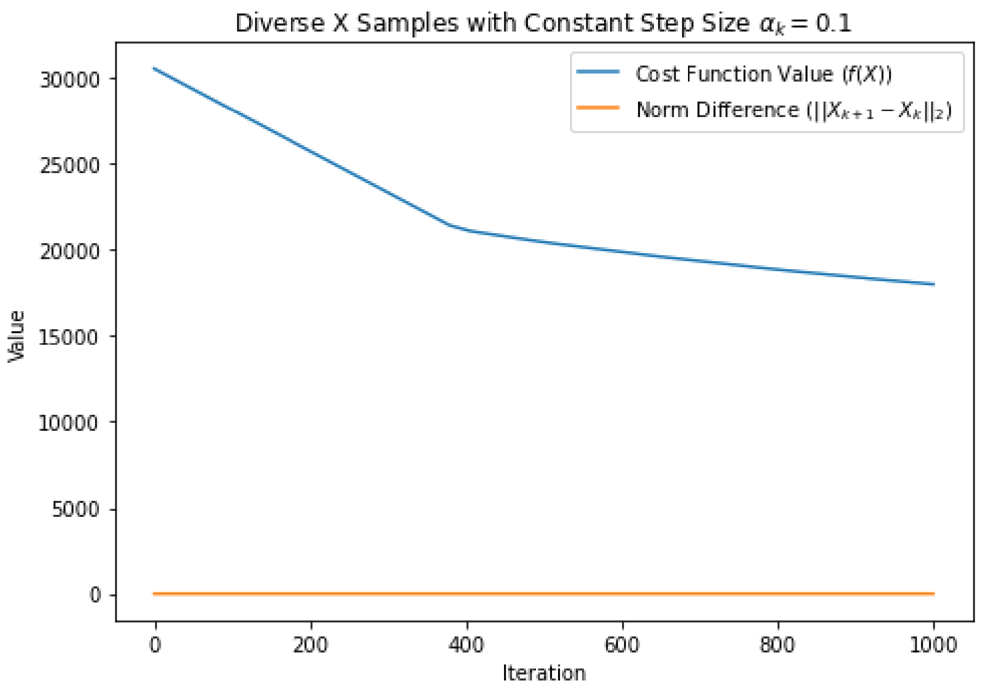


Figure23

- For the constant step size = 0.1

The cost function value decreases as the iterations progress, indicating that the method is optimizing the function f(X). The norm difference also shows a decreasing trend, which suggests the iterative updates to X are getting smaller as the algorithm proceeds, pointing towards convergence.