Assignment 2: Computational Methods of Optimization

Sacchit
B.Tech Mathematics and Computing
Srn : 22219
Indian Institute of Science

October 2024

1 Conjugate Gradient Descent

1.1 Unique Solution for Symmetric Positive Definite Matrix (2 points)

Suppose A is an $n \times n$ symmetric positive definite matrix. The equation $A\mathbf{x} = \mathbf{b}$ has a unique solution,since A is invertible, To express this as a convex quadratic minimization problem, consider minimizing the following function:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{b}^\top \mathbf{x}$$

The gradient of this function with respect to $\mathbf{x}$ is:

$$\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$$

Setting $\nabla f(\mathbf{x}) = 0$ yields the optimal solution $\mathbf{x}^*$, which satisfies:

$$A\mathbf{x}^* = \mathbf{b}$$

Therefore, the minimization problem has a unique solution.

1.2 Implementation of Conjugate Gradient Descent (5 points)

The conjugate gradient descent algorithm is implemented to solve the system $A\mathbf{x} = \mathbf{b}$. The algorithm starts with an initial guess $\mathbf{x}_0$ and iteratively updates the solution based on the conjugate directions. The key steps are:
1. Compute the residual $r_0 = \mathbf{b} - A\mathbf{x}_0$.
2. Initialize $p_0 = r_0$ (the search direction).
3. For each iteration, compute α_k, update $\mathbf{x}_k$, and adjust the residual and search direction.

The algorithm stops when the residual is sufficiently small or the maximum number of iterations is reached.

Algorithm:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$$
$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

where:

$$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A \mathbf{p}_k}, \quad \beta_k = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$$

Results:

Using the oracle function, we obtained the following results for the optimum solution $\mathbf{x}^*$ and the number of iterations required to reach the optimum:

- $\mathbf{x}^* = [2.0, 3.0, 6.0, 0.0, 5.0]$
- Number of iterations = 5

1.3 Solving $Ax = b$ for $m > n$ (3 points)

If A is an $m \times n$ matrix with $m > n$, the equation $A\mathbf{x} = \mathbf{b}$ does not generally have a unique solution because the system is over-determined. Instead of finding an exact solution, we can solve the least-squares problem:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2$$

This can be formulated as the following convex quadratic minimization problem:

$$\min_{\mathbf{x}} \frac{1}{2}\|A\mathbf{x} - \mathbf{b}\|_2^2 = \frac{1}{2}\left(A\mathbf{x} - \mathbf{b}\right)^\top \left(A\mathbf{x} - \mathbf{b}\right)$$

Taking the gradient with respect to $\mathbf{x}$ and setting it to zero, we get:

$$A^\top A\mathbf{x} = A^\top \mathbf{b}$$

The above equation is the normal equation for the least-squares problem. This equation will have a unique solution if $A^\top A$ is invertible, i.e., if the columns of A are linearly independent.

1.4 Implementation for $m > n$ (5 points)

For this part, we will use the oracle to obtain a matrix A of size $m \times n$ with $m > n$ and a vector $\mathbf{b}$ of size $m \times 1$. We will then find $\mathbf{x}^*$ that minimizes the error $\|A\mathbf{x} - \mathbf{b}\|_2^2$ using the Conjugate Gradient Descent method on the normal equation.

Algorithm:

We solve the normal equation $A^\top A\mathbf{x} = A^\top \mathbf{b}$ using Conjugate Gradient Descent as described earlier.

Results:

Using the oracle function, we obtained the following results for the optimum solution $\mathbf{x}^*$ and the number of iterations required to minimize $\|A\mathbf{x} - \mathbf{b}\|_2^2$:

- $\mathbf{x}^* = [2.0, 3.0, 6.0, 0.0, 5.0]$
- Number of iterations = 1

Problem 2: Newton's Method (10 points)

The Newton-update equation is given by:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - H(\mathbf{x}^{(t)})^{-1}\nabla f(\mathbf{x}^{(t)})$$

where $\nabla f(\mathbf{x})$ and $H(\mathbf{x})$ are the gradient and Hessian of a function $f : \mathbb{R}^d \to \mathbb{R}$ at the point $\mathbf{x}$.

2.1 Gradient Descent for 100 Iterations (2 points)

We run gradient descent for 100 iterations starting from $\mathbf{x}_0 = [0, 0, 0, 0, 0]^\top$, using five different step-sizes. For each step-size, we plot the function value at each iteration and report the value of $\mathbf{x}$ at the final iteration.

Results:

After running gradient descent with five different step sizes, the final values of $\mathbf{x}$ are as follows:

- For step size 0.5: $\mathbf{x} = [[1.0, 1.0, 1.0, 1.0, -2.551552067298684e + 54]]$
- For step size 0.1: $\mathbf{x} = [1.0, 1.0, 1.0, 1.0, 1.0]$
- For step size 0.01: $\mathbf{x} = [0.8674, 0.8674, 0.8674, 0.634, 0.9999]$
- For step size 0.001: $\mathbf{x} = [0.1814, 0.1814, 0.1814, 0.0952, 0.5951]$

We also plot the function values for each step-size across the iterations (see Figure 1).

2.2 Newton's Method for 100 Iterations (2 points)

We execute Newton's method for 100 iterations, starting from $\mathbf{x}_0 = [0, 0, 0, 0, 0]^\top$, and compare the results with gradient descent. The function values are plotted at each iteration.

Results:

After 100 iterations of Newton's method, the final value of $\mathbf{x}$ is:

$$\mathbf{x} = [1.1.1.1.1.]$$

The function values for Newton's method over the 100 iterations are plotted in Figure 2.

The Newton method solutions seems to be more accurate than gradient descent solutions.

2.3 Newton's Method from Different Initial Points (3 points)

We run Newton's method for 100 iterations, starting from five different initial points, and observe the function values at each iteration. The following observations are made:

- The function converged to the same final value..
- For all initial points the function converge in a single step.

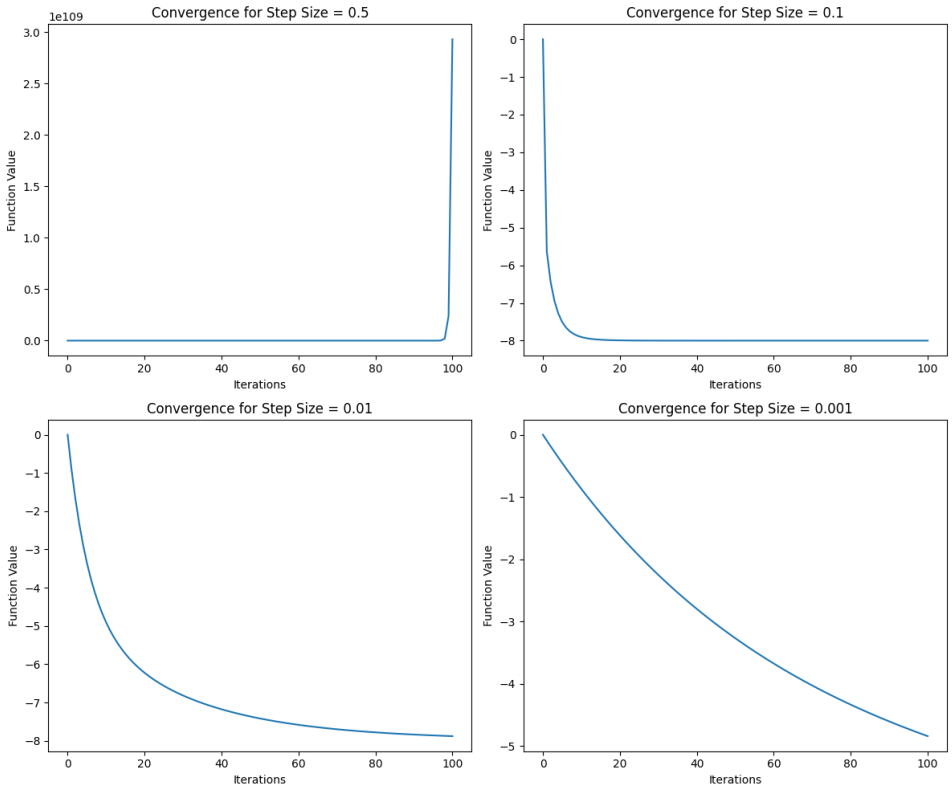

Figure 1: Function values for different step-sizes in gradient descent.

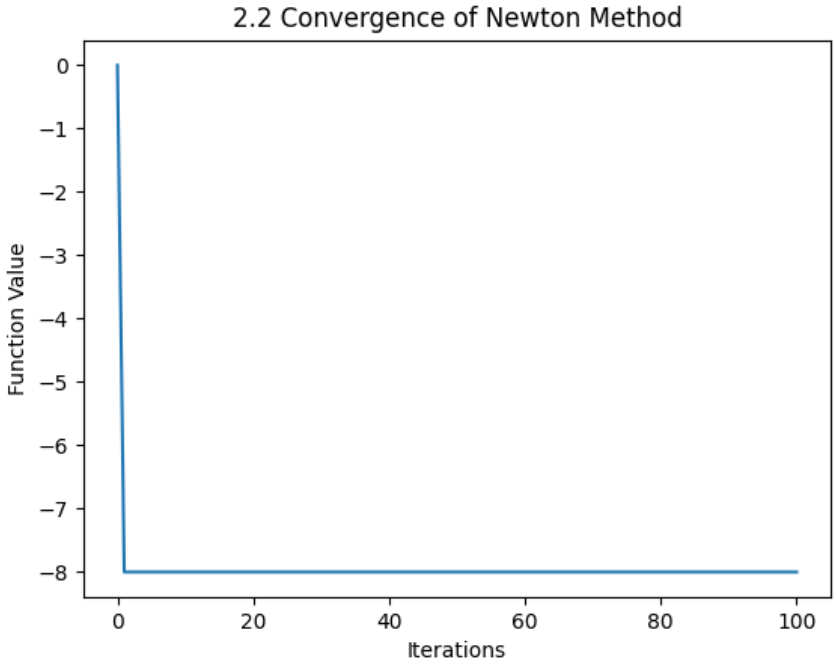

Figure 2: Function values during Newton's Method.

2.4 Guess the Nature of the Function (3 points)

Based on the observations from the previous subpart, we conclude that the function in the oracle is a convex quadratic function. This conclusion is supported by the fact that

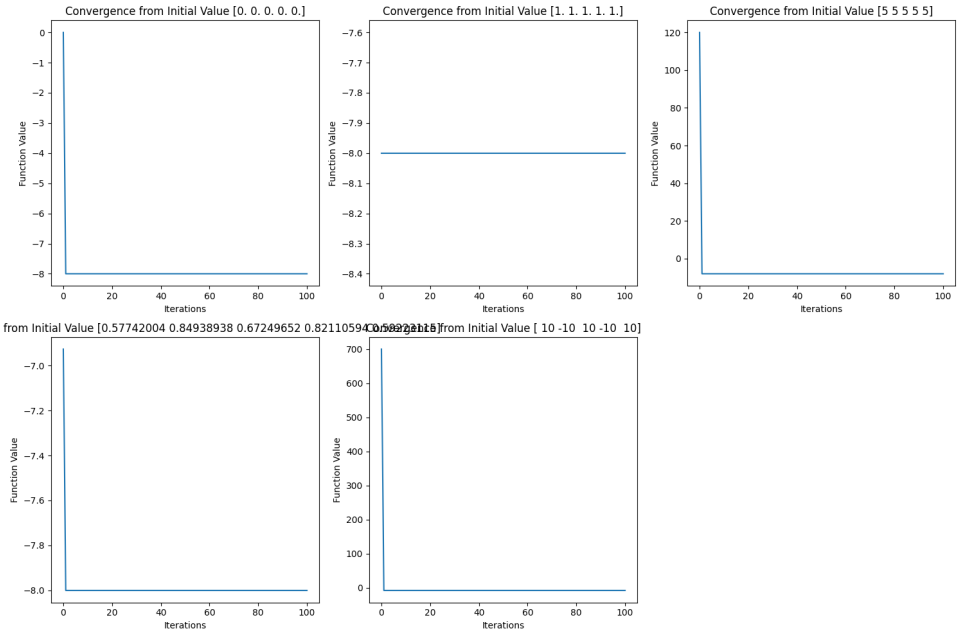

Figure 3: Function values during Newton's Method for different Initial Points.

Newton's method converges in a single iteration from all initial points.

For convex quadratic functions, the Hessian is constant, and Newton's method is able to compute the exact solution in one step. This is because Newton's method minimizes quadratic functions exactly due to the second-order nature of the method. Additionally, the strong convexity of the function ensures rapid convergence, which aligns with the results observed in our experiments.

Problem 3: Newton's Method continued (15 points)

In this problem, we will further explore the performance of Newton's method and compare it with gradient descent. We will use the oracle function f_3, starting with $\mathbf{x}_0 = [1, 1, 1, 1, 1]^\top$.

3.1 Gradient Descent with Step-size 0.1 (2 points)

We run gradient descent with a step-size of 0.1 for 100 iterations and plot the function values at each iteration.

Results: After running gradient descent, the best function value obtained over 100 iterations is:

$$f(\mathbf{x}^*) = 6.4826$$

The plot of the function values over the 100 iterations is shown in Figure 4.

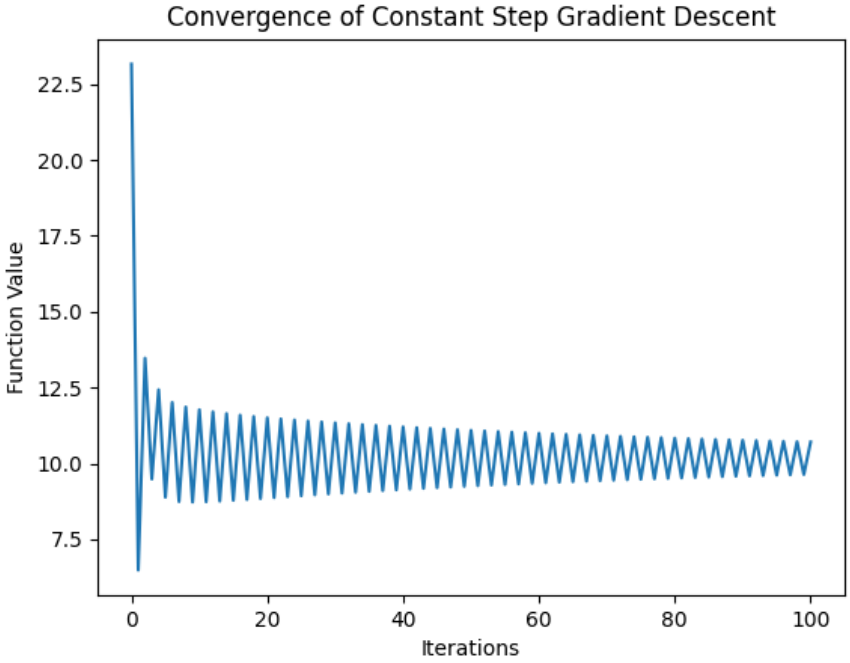

Figure 4: Function values during Gradient Descent with step-size 0.1.

3.2 Observations on Oscillations (2 points)

Oscillations in Gradient Descent

Oscillations in gradient descent can occur due to several reasons. Below, we discuss the possible causes and methods to overcome these oscillations.

Possible Reasons for Oscillations

1. **Large Step Size (Learning Rate):** If the step size is too large, the updates may overshoot the minimum, causing the optimization path to oscillate around the optimal solution.

2. **Highly Conditioned Problem:** When the Hessian of the objective function has a high condition number (i.e., the problem has steep gradients in some directions and shallow gradients in others), gradient descent can oscillate or progress too slowly in certain directions.

3. **Non-convex Objective Function:** Non-convex functions may contain saddle points or local minima where the gradient oscillates between regions, leading to erratic or oscillatory behavior.

Ways to Overcome Oscillations

1. **Reduce Step Size (Learning Rate):** Use a smaller step size to ensure more conservative updates that reduce overshooting.

2. **Use Momentum:** Momentum-based optimization methods (e.g., Momentum Gradient Descent or Nesterov Accelerated Gradient) add a velocity term to smooth out the oscillations by incorporating the history of updates.

3. **Inexact Line Search:** Inexact line search methods such as the *Armijo rule* or *Wolfe conditions* can be used to determine an appropriate step size at each iteration. These methods ensure that each step leads to sufficient descent while controlling oscillations, without requiring the exact minimization of the step size.

3.3 Newton's Method for 100 Iterations (3 points)

We now run Newton's method for 100 iterations, starting from $\mathbf{x}_0 = [1, 1, 1, 1, 1]^\top$, and report the function values for the first 10 iterations.

Results: The function values for the first 10 iterations of Newton's method are as follows:

Iteration	$f(\mathbf{x})$
1	Nan
2	Nan
3	Nan
$\vdots$	$\vdots$
10	Nan

The algorithm did not converge within the first 10 iterations because the oracle is returning `NaN` values for $H(x)^{-1}\nabla f(x)$. This issue likely arises because the Hessian at the current point $\mathbf{x}_0$ is either not invertible or is very close to singular. When the Hessian matrix is singular or poorly conditioned, the inversion becomes unstable, which leads to numerical errors such as division by zero or multiplication with extremely large values, resulting in `NaN` outputs.

Explanation:

- **Non-Invertible Hessian:** At the starting point $\mathbf{x}_0$, the Hessian matrix could be singular or nearly singular. In such cases, the computation of $H(x)^{-1}\nabla f(x)$ becomes problematic, leading to numerical instability.

- **Ill-Conditioned Hessian:** Even if the Hessian is not singular, it might have a high condition number, meaning there is a large disparity between its largest and smallest eigenvalues. This results in inaccurate computation of the inverse, causing numerical errors when multiplied by the gradient.

3.4 Hybrid Approach: Gradient Descent followed by Newton's Method (8 points)

To balance the computational cost of Newton's method with the benefits of its fast convergence, we run a hybrid approach where we start with K iterations of gradient descent followed by Newton's method. Since each Newton-update is approximately 25 times more costly than a gradient-update, we optimize the number of gradient iterations to minimize the total cost while achieving the best function value.

We experiment with different values of K, where K represents the number of gradient iterations before switching to Newton's method. For each experiment, we report the total cost, the function value at the final iteration, and the value of $\mathbf{x}$.

Results: After running the hybrid method with various values of K, we find the following:

K	Total Cost	$f(\mathbf{x})$	$\mathbf{x}$
35	1650	3	$[0, 0, 0, 0, 0]$
40	1540	3	$[0, 0, 0, 0, 0]$
55	1180	3	$[0, 0, 0, 0, 0]$
70	820	3	$[0, 0, 0, 0, 0]$

The best results were achieved for all K ,least cost was 820 units, and the final function value of $f(\mathbf{x}) = 3$.

We also plot the function values over the iterations, with gradient steps denoted by blue dots ($\bullet$) and Newton steps denoted by red crosses ($\times$) in Figure 5.

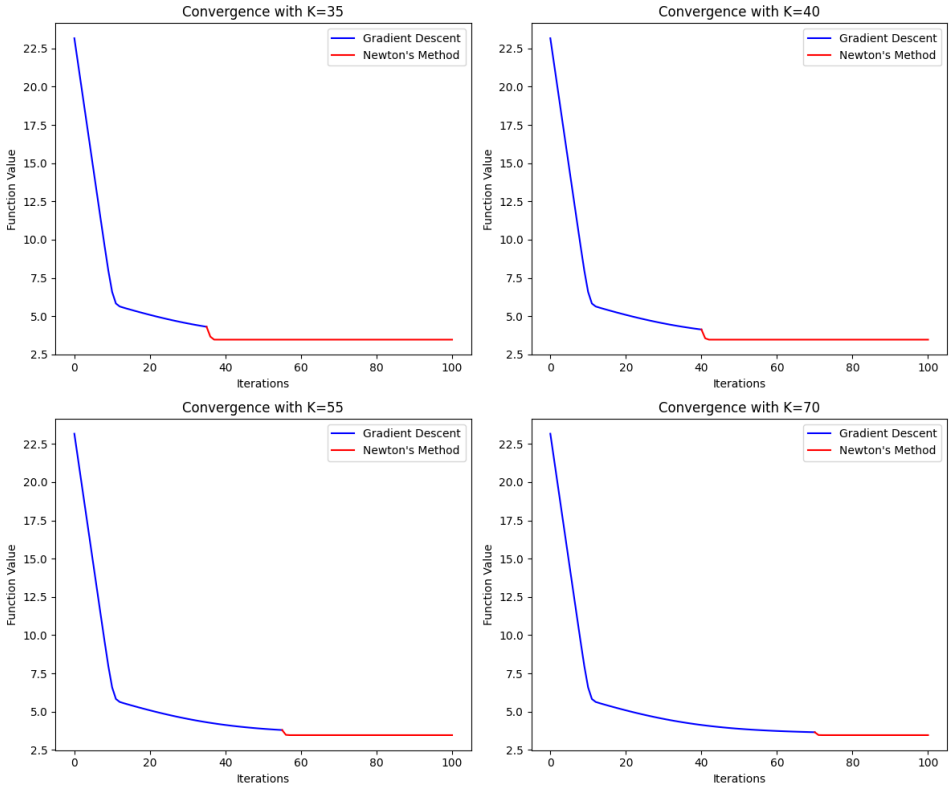

Figure 5: Function values during hybrid approach with gradient and Newton's method.

Problem 4: Quasi-Newton Methods (10 points)

In this problem, we explore Quasi-Newton methods, which are used to approximate the Hessian matrix in optimization problems without explicitly calculating it, often to save computational resources.

4.1 Hessian Approximation Using a Scalar Multiple of the Identity Matrix (5 points)

In cases where memory constraints prevent storing a full Hessian matrix, one strategy is to approximate the Hessian as a scalar multiple of the identity matrix. This approach reduces the memory requirements, as we only need to store a single scalar in addition to the current values of $\mathbf{x}$ and $\nabla f(\mathbf{x})$.

We begin by writing the secant equation for Quasi-Newton methods:

$$H_{k+1}(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

By approximating the Hessian as a scalar multiple of the identity matrix, $H_{k+1} = \lambda I$, where λ is a scalar and I is the identity matrix, we can derive an update rule for λ.

From the secant equation, we have:

$$\lambda(\mathbf{x}_{k+1} - \mathbf{x}_k) = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$$

Solving for λ, we obtain:

$$\lambda = \frac{(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))^\top (\mathbf{x}_{k+1} - \mathbf{x}_k)}{(\mathbf{x}_{k+1} - \mathbf{x}_k)^\top (\mathbf{x}_{k+1} - \mathbf{x}_k)}$$

Thus, we update the approximation of the Hessian at each iteration by adjusting the scalar λ, which maintains the secant equation while minimizing memory usage. This approximation works well when the Hessian matrix is close to a scaled identity matrix. subsection*4.2 Comparison with Gradient Descent and Quasi-Newton Rank 1 Update (5 points)

In this part, we compare the results of using the Hessian approximation from the previous section with standard gradient descent and a more sophisticated Quasi-Newton rank-1 update method.

The Quasi-Newton rank-1 update can be expressed as:

$$H_{k+1} = H_k + \frac{(\mathbf{y}_k - H_k \mathbf{s}_k)(\mathbf{y}_k - H_k \mathbf{s}_k)^\top}{(\mathbf{y}_k - H_k \mathbf{s}_k)^\top \mathbf{s}_k}$$

where:

$$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k), \quad \mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$$

We run each method for 100 iterations and plot the function values at each iteration. The final value of $\mathbf{x}$ obtained by each method is reported, along with the corresponding function values.

Results:

The following table summarizes the results for each method:

Method	$f(\mathbf{x}^*)$	$\mathbf{x}^*$
Gradient Descent	-8	[1.1.1.0.999973441.]
Quasi-Newton (Scalar Approximation)	-8	[1.1.1.1.1.]
Quasi-Newton (Rank 1 Update)	-8	[0.999990.999990.999990.969490.99934673]

Observations:

- The Scalar Approximation with scalar approximation converged faster than gradient descent and full rank one update.
- The Quasi-Newton rank-1 update conerged the slowest.
- Gradient descent required more iterations to reach the same accuracy as Scalar Approximation, but it involved fewer computational resources.
- The Quasi-Newton rank-1 update which is expected to coverge in 6 steps i.e (5+1) diverged this means $(\mathbf{y}_k - H_k\mathbf{s}_k)^\top \mathbf{s}_k = 0$ for some k.
- Hence Armiho-Wolfe conditions were used to overcome the above problem.

Plots:
The function values over the 100 iterations for each method are plotted in Figure 6.

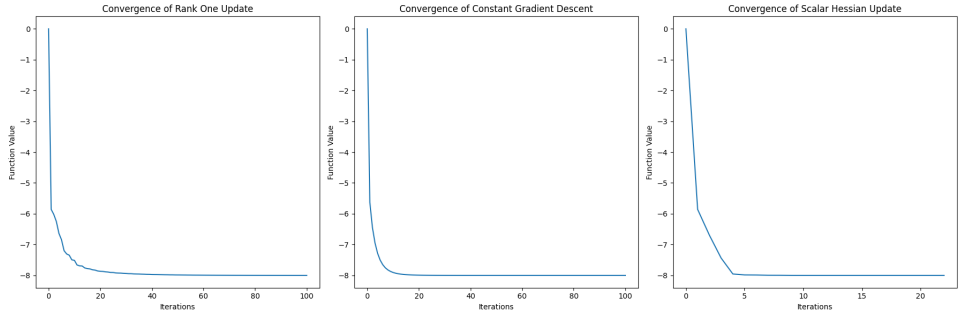

Figure 6: Function values over iterations for Gradient Descent, Quasi-Newton (Scalar Approximation), and Quasi-Newton (Rank 1 Update).