# Assignment 2

# E0 230

## Computational methods of Optimization

**Professor Chiranjib Bhattacharyya**

**Submitted by:**

Pratyush Kant

## 1 Conjugate Gradient Descent (15 points)

In this question, we will explore different methods to solve a system of equations given by $\mathbf{Ax} = \mathbf{b}$.

1. Suppose $\mathbf{A}$ is an $n \times n$ symmetric, PD matrix. Does this equation $\mathbf{Ax} = \mathbf{b}$ have a unique solution? State a convex quadratic minimization problem whose optimal solution is $\mathbf{x}^*$ such that $\mathbf{Ax}^* = \mathbf{b}$.
2. Query the oracle for a PD matrix $\mathbf{A}$ and a vector $\mathbf{b}$. Implement conjugate gradient descent on the optimization problem in previous part. Report the optimum $\mathbf{x}^*$ and the number of iterations it took to reach the optimum.
3. Suppose $\mathbf{A}$ is an $m \times n$ matrix with $m > n$. Does $\mathbf{Ax} = \mathbf{b}$ have a unique solution? One strategy to solve such equations is to minimize the error term $||\mathbf{Ax} - \mathbf{b}||$. Write down a quadratic minimization problem to minimize this error term. When does it have a unique solution?
4. Query the oracle for an $m \times n$ matrix $\mathbf{A}$ with $m > n$ and a $m \times 1$ vector $\mathbf{b}$. Find an $\mathbf{x}^*$ that minimizes the error term $||\mathbf{Ax} - \mathbf{b}||$. Report the optimal $\mathbf{x}^*$ and the number of iterations it took to reach the optimum.

### Oracle

Import the oracle f1. The arguments to be passed (strictly in the given order) to the function are:
1. srno: the last five digits of your serial number passed as an integer, and,
2. subq: a True or False value passed as a Boolean variable.

The oracle returns a matrix and a vector depending on the value of srno and subq as follows:
1. True: the oracle will return a symmetric, PD matrix $\mathbf{A}$ and an appropriate vector b for subquestion 2.
2. False: the oracle will return a $m \times n$ matrix $\mathbf{A}$ and an $m \times 1$ vector $\mathbf{b}$ for subquestion 4.

## Solution

1. Since $\mathbf{A}$ is an $n \times n$ symmetric, PD matrix, it is invertible. Hence, the equation $\mathbf{Ax} = \mathbf{b}$ has a unique solution, namely $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

   The convex quadratic minimization problem whose optimal solution is $\mathbf{x}^*$ such that $\mathbf{Ax}^* = \mathbf{b}$ is given by:

$$\min_{x} \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{x}$$

   **Lemma 0.1.** *Let $\mathbf{A} \in \mathbb{R}^{n\times n}$. Then $\mathbf{A}$ is invertible if and only if $\mathbf{Ax} = \mathbf{0} \implies \mathbf{x} = 0$.*

*Proof.* If $\mathbf{A}$ is invertible, then $\mathbf{Ax} = \mathbf{0} \implies \mathbf{x} = \mathbf{A}^{-1}\mathbf{0} = \mathbf{0}$.

Suppose there exists a non-zero vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{0}$ and $\mathbf{A}$ is invertible. Then $\mathsf{nullity}(\mathbf{A}) > 0$. From the rank-nullity theorem, we have $\mathsf{rank}(\mathbf{A}) + \mathsf{nullity}(\mathbf{A}) = n$. Since $\mathsf{nullity}(\mathbf{A}) > 0$, we have $\mathsf{rank}(\mathbf{A}) < n$. But $\mathbf{A}$ is invertible, so $\forall\ \mathbf{b} \in \mathbb{R}^n$, $\exists\ \mathbf{x} \in \mathbb{R}^n$ given by $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ such that $\mathbf{Ax} = \mathbf{AA}^{-1}\mathbf{b} = \mathbf{b}$. This implies that $\mathsf{rank}(\mathbf{A}) = n$, which is a contradiction. Hence, $\mathbf{A}$ is not invertible. □

**Lemma 0.2.** *Every PD matrix is invertible.*

*Proof.* Let $\mathbf{A}$ be a PD matrix. Then, for any non-zero vector $\mathbf{x}$, we have $\mathbf{x}^T\mathbf{Ax} > 0$. This implies that $\mathbf{Ax} \neq \mathbf{0}$ for any non-zero vector $\mathbf{x}$. Hence, $\mathbf{A}$ is invertible, for if $\mathbf{A}$ were not invertible, then there would exist a non-zero vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{0}$. □

2. Call the function f1 with sr_no = 22221 and Boolean value 1 to get the PD matrix $\mathbf{A}$ and the vector $\mathbf{b}$.

$$\mathbf{A} = \begin{pmatrix} 2.37626673 & 0.83229223 & 0.6248642 & 0.87410062 & 0.22388762 \\ 0.83229223 & 2.12527297 & 0.62029519 & 0.66906253 & 0.54120011 \\ 0.6248642 & 0.62029519 & 2.69523595 & 0.27647378 & 0.40153497 \\ 0.87410062 & 0.66906253 & 0.27647378 & 2.04197985 & 0.44878946 \\ 0.22388762 & 0.54120011 & 0.40153497 & 0.44878946 & 2.58379722 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 13.43990929 \\ 16.21963782 \\ 22.36924826 \\ 10.59773762 \\ 23.01595542 \end{pmatrix}$$

Recall the conjugate gradient descent algorithm presented in the algorithm 1.

We write a Python function to implement the conjugate gradient descent algorithm named conjugate_gradient_descent. The function takes as input the matrix $\mathbf{A}$, the vector $\mathbf{b}$. Since the convergence is guaranteed in $n$ iterations, we set the initial guess as a random vector of size same as $\mathbf{b}$. The function conjugate_gradient_descent returns the optimal solution $\mathbf{x}^*$ and the number of iterations it took to reach the optimum.

From the output, we have the optimal solution $\mathbf{x}^*$ and the number of iterations it took to reach the optimum are given as:

$$\mathbf{x}^* = (-2, -3, -6, -1, -7)^T$$

Number of iterations = 5

**Algorithm 1** Conjugate gradient method

```
1: Objective:
2: To find the global minimum of $\min_{x\in\mathbb{R}^n} \frac{1}{2}x^TQx + b^Tx$.
3: Input:
4: A first approximation $x_1$ of the solution.
5: The symmetric positive definite matrix $Q \in \mathbb{R}^{n\times n}$.
6: The vector $b \in \mathbb{R}^n$.
7: Output:
8: The solution $x^* \in \mathbb{R}^n$.
9: Initialization:
10: $k := 1$,
11: $d_1 := -Qx_1 - b$.
12: Repeat:
13: $\alpha_k := -\frac{d_k^T(Qx_k+b)}{d_k^TQd_k}$.
14: $x_{k+1} := x_k + \alpha_k d_k$.
15: $\beta_{k+1} := \frac{\nabla f(x_{k+1})^T\nabla f(x_{k+1})}{\nabla f(x_k)^T\nabla f(x_k)} = \frac{(Qx_{k+1}+b)^T(Qx_{k+1}+b)}{(Qx_k+b)^T(Qx_k+b)}$.
16: $d_{k+1} := -Qx_{k+1} - b + \beta_{k+1}d_k$.
17: $k := k+1$.
18: Until: $\|\nabla f(x_k)\| = 0$ or $k = n+1$
19: $x^* = x_k$.
```

3. No, in general the equation $\mathbf{Ax} = \mathbf{b}$ does not have a unique solution when $\mathbf{A}$ is an $m \times n$ matrix with $m > n$. Consider the counterexample where

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Then for all $x \in \mathbb{R}$, it holds that

$$A\begin{pmatrix} 1 \\ x \end{pmatrix} = b$$

In this case, the equation $\mathbf{Ax} = \mathbf{b}$ has infinitely many solutions. The equation $\mathbf{Ax} = \mathbf{b}$ may not have a solution as well as for the same $\mathbf{A}$ above and $\mathbf{b} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^T$, no solution exists. It may have a unique solution as well as illustrated for subquestion 4.

The quadratic minimization problem to minimize the error term $||\mathbf{Ax} - \mathbf{b}||$ is given by:

$$\min_x \frac{1}{2}||\mathbf{Ax} - \mathbf{b}||^2$$

$$\begin{aligned}||\mathbf{Ax}-\mathbf{b}||^2 &= (\mathbf{Ax}-\mathbf{b})^T(\mathbf{Ax}-\mathbf{b})\\ &= (\mathbf{x}^T\mathbf{A}^T-\mathbf{b}^T)(\mathbf{Ax}-\mathbf{b})\\ &= \mathbf{x}^T\mathbf{A}^T\mathbf{Ax}-\mathbf{b}^T\mathbf{Ax}-\mathbf{x}^T\mathbf{A}^T\mathbf{b}+\mathbf{b}^T\mathbf{b}\end{aligned}$$

Since $\mathbf{b}^T\mathbf{Ax}$ is a scalar, we have $\mathbf{b}^T\mathbf{Ax} = (\mathbf{b}^T\mathbf{Ax})^T = \mathbf{x}^T\mathbf{A}^T\mathbf{b}$. Hence, the quadratic minimization problem becomes:

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{A}^T\mathbf{Ax}-\mathbf{b}^T\mathbf{Ax}+\frac{1}{2}\mathbf{b}^T\mathbf{b}$$

Let $\mathbf{x}^*$ be the optimal solution to the above minimization problem. Then it must be a local minimum. The necessary condition for $\mathbf{x}^*$ to be a local minimum is that the gradient of the function at $\mathbf{x}^*$ must be zero. Hence, we have:

$$\begin{aligned}&\nabla f(\mathbf{x}^*) = \mathbf{A}^T\mathbf{Ax}^*-\mathbf{A}^T\mathbf{b}=0\\ \implies &\mathbf{A}^T\mathbf{Ax}^* = \mathbf{A}^T\mathbf{b}\end{aligned}$$

If $\mathbf{A}^T\mathbf{A}$ is invertible, then the above equation has a unique solution $\mathbf{x}^*$, given by $\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$. Hence, the quadratic minimization problem has a unique solution if $\mathbf{A}^T\mathbf{A}$ is invertible.

4. The matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are given by the oracle. The optimal solution can be found by calling the function conjugate_gradient_descent with the matrix $\mathbf{A}^T\mathbf{A}$ and the vector $\mathbf{A}^T\mathbf{b}$. The optimal solution $\mathbf{x}^*$ and the number of iterations it took to reach the optimum are given as:

$$\mathbf{x}^* = \begin{pmatrix} 2.00000000e+00\\ -3.00000000e+00\\ -3.58075015e-16\\ -9.00000000e+00\\ -5.00000000e+00\end{pmatrix}$$

$$\text{Number of iterations} = 5$$

## 2 Newton's Method (10 points)

As discussed in class, the Newton-update 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 $\mathbf{x}$, respectively. Write code to implement Newton's method to minimise an unconstrained objective function. This code-snippet will henceforth be referred to as Newton in the assignment.

Now, import the oracle f2, which encodes a function $f : \mathbb{R}^5 \to \mathbb{R}$. The arguments to be passed (strictly in the given order) to the function are:

1. $\mathbf{x} \in \mathbb{R}^5$: a $1 \times 5$ numpy array,
2. srno: the last five digits of your serial number passed as an integer, and,
3. order $\in \{0, 1, 2\}$.

The oracle returns the function-value at $\mathbf{x}$ if the order is 0, $\nabla f(\mathbf{x})$ if the order is 1, and $\mathbf{H}(\mathbf{x})^{-1}\nabla f(\mathbf{x})$ if the order is 2. Also, let $\mathbf{x}_0 = [0, 0, 0, 0, 0]^T$.

1. Run gradient descent (implemented in Assignment 1) for 100 iterations with five choices of step-sizes with $\mathbf{x}_0$ as the initial point. Plot the function value at each iteration for each choice of step-size. Report $\mathbf{x}$ at the final iteration.
2. Execute Newton for 100 iterations starting from $\mathbf{x}_0$, and plot the function values at each iteration. Compare the results with those from gradient descent. Report $\mathbf{x}$ at the final iteration.
3. Execute Newton for 100 iterations starting from five different initial points, and plot the function values at each iteration. What do you observe? There are two important observations that can be made.
4. Based on the above observation, guess the nature of the function in the oracle, and prove that the observation holds for such functions.

## Solution

Define the following functions in the script:

$$\begin{aligned} \text{f2\_value}(\mathbf{x}) &:= \text{f2}(\mathbf{x}, \text{srno}, 0) \\ \text{f2\_gradient}(\mathbf{x}) &:= \text{f2}(\mathbf{x}, \text{srno}, 1) \\ \text{Newton\_update\_term}(\mathbf{x}) &:= \text{f2}(\mathbf{x}, \text{srno}, 2) \end{aligned}$$

1. We run gradient descent for 100 iterations with five choices of step-sizes with $\mathbf{x}_0 = [0, 0, 0, 0, 0]^T$ as the initial point. The step-sizes are chosen as

   $$\alpha \in \{1e-3, 1e-2, 0.05, 0.1, 0.5\}$$

   The function value at each iteration for each choice of step-size is plotted in figure 1. The final $\mathbf{x}$ at the end of 100 iterations is reported and some other statistics are given in the tables 1 and 2.

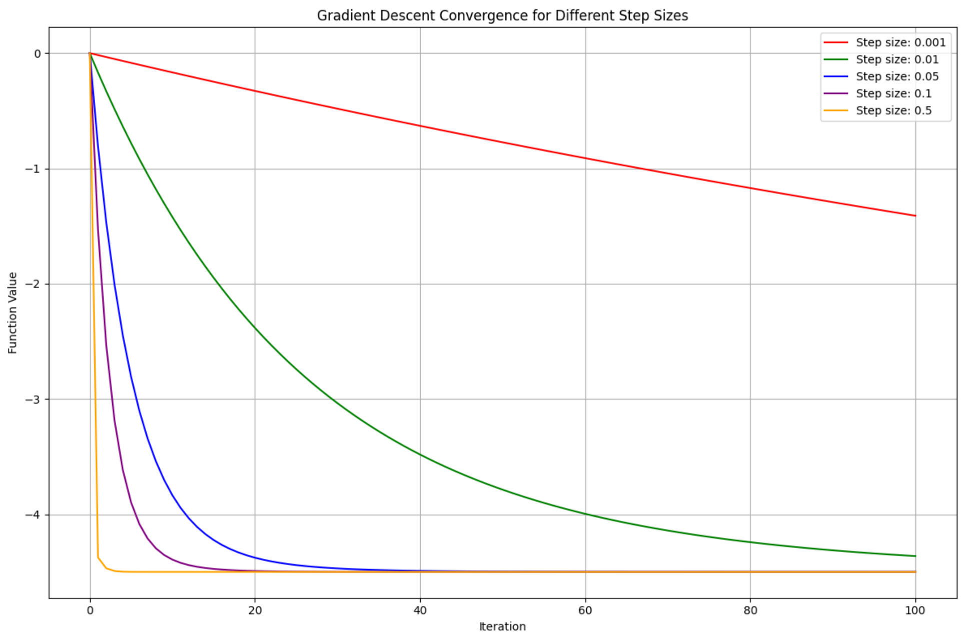


Figure 1: Function value vs iterations for different step-sizes

| **Step Size** | **Final x** |
|---|---|
| $1 \times 10^{-3}$ | [0.1814332, 0.1814332, 0.1814332, 0.1814332, 0.09520785] |
| $1 \times 10^{-2}$ | [0.86738044, 0.86738044, 0.86738044, 0.86738044, 0.63396766] |
| 0.05 | [0.99997344, 0.99997344, 0.99997344, 0.99997344, 0.99407947] |
| 0.1 | [1.0, 1.0, 1.0, 1.0, 0.99997344] |
| 0.5 | [1.0, 1.0, 1.0, 1.0, 1.0] |

Table 1: Results for Final $x$

2. The plot of function values at each iteration for Newton's method is shown in figure 2. The final **x** at the end of 100 iterations is reported in table 3.

   Unlike gradient descent, Newton's method converges to the optimal solution in a single iteration. This is because Newton's method unlike gradient descent uses the second-order information of the function to find the optimal solution. The function value at the final iteration is $-4.5$ and the gradient norm is 0.

3. The plot of function values at each iteration for Newton's method starting from five different initial points is shown in figure 3. The two important observations that can be made are:

   (a) Newton's method converges to the optimal solution in a single iteration irrespective of the initial point.

   (b) The final point is $[1, 1, 1, 1, 1]^T$, the function value at the final iteration is $-4.5$ and the gradient norm is 0 at that point. The point is the local minima of the function as there are points where the function value is greater than $-4.5$. (Refer to the plot in figure 3)

4. The function in the oracle is a quadratic function. The function is given by:

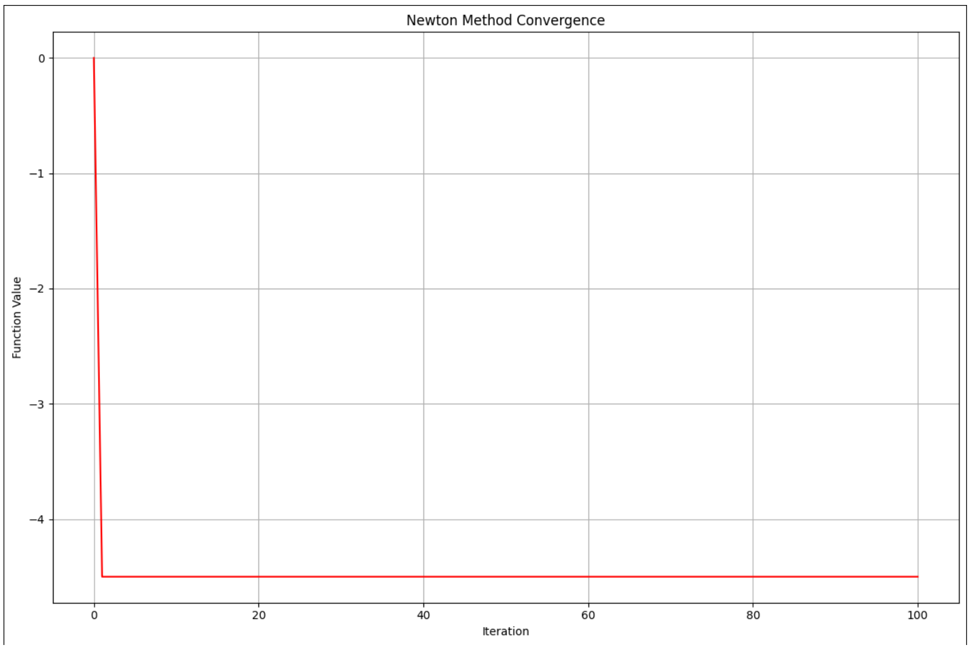


Figure 2: Function value vs iterations for Newton's method

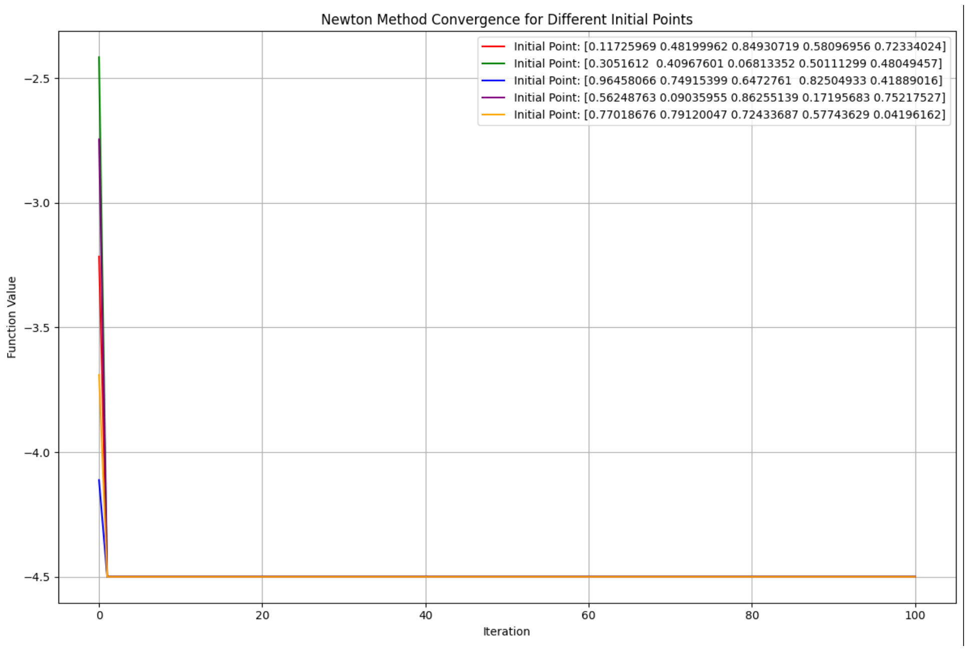


Figure 3: Function value vs iterations for Newton's method with different initial points

| **Step Size** | **Final f(x)** | **Gradient Norm** |
|---|---|---|
| $1 \times 10^{-3}$ | -1.4104691303093915 | 3.39698022503573 |
| $1 \times 10^{-2}$ | -4.362658376148133 | 0.6445050973805453 |
| 0.05 | -4.499982470844842 | 0.005921482447445214 |
| 0.1 | -4.499999999647246 | 2.656139890017905e-05 |
| 0.5 | -4.5 | 0.0 |

Table 2: Results for Final $f(x)$ and Gradient Norm

| **Final x** | **Final f(x)** | **Gradient Norm** |
|---|---|---|
| [1.0, 1.0, 1.0, 1.0, 1.0] | -4.5 | 0 |

Table 3: Results for Final $x$

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{g}^T\mathbf{x} + c$$

Let $\mathbf{x}_0$ be the initial point and $\mathbf{x}^*$ be the optimal solution. Then the gradient and Hessian of the function is given by:

$$\begin{aligned}\nabla f(\mathbf{x}) &= \mathbf{Q}\mathbf{x} - \mathbf{g}\\ \mathbf{H} &= \nabla^2 f(\mathbf{x}) = \mathbf{Q}\end{aligned}$$

At the optimal solution $\mathbf{x}^*$, we have $\nabla f(\mathbf{x}^*) = 0$. Hence, we have:

$$\begin{aligned}\mathbf{Q}\mathbf{x}^* &= \mathbf{g}\\ \implies \mathbf{x}^* &= \mathbf{Q}^{-1}\mathbf{g}\end{aligned}$$

The Newton's method update is given by:

$$\begin{aligned}\mathbf{x}(t+1) &= \mathbf{x}(t) - \mathbf{H}^{-1}\nabla f(\mathbf{x}(t))\\ &= \mathbf{x}(t) - \mathbf{Q}^{-1}(\mathbf{Q}\mathbf{x}(t) - \mathbf{g})\\ &= \mathbf{x}(t) - \mathbf{x}(t) + \mathbf{Q}^{-1}\mathbf{g}\\ &= \mathbf{x}^*\end{aligned}$$

Hence, Newton's method converges to the optimal solution in a single iteration for quadratic functions. This is the reason why Newton's method converges to the optimal solution in a single iteration irrespective of the initial point in subquestion 3.

## 3 Newton's Method continued (15 points)

For this problem, use the oracle f3, which takes the same arguments as f2 and returns values in a similar format. Also, for all of the following subparts, start with $\mathbf{x}^{(0)} = [1, 1, 1, 1, 1]6T$ as the initial point.

1. Run gradient descent with a step-size of 0.1 for a 100 iterations and plot the function-values. Report the best function value obtained over all the iterations.

2. Did you observe oscillations in the above plot? Try to guess possible reasons for this phenomenon, and suggest ways to overcome them.

3. Execute Newton for a 100 iterations, and report the function-values for only the first 10 iterations. Did the algorithm converge? If not, why?

4. Now, let's play a game! Note that the cost of calculating the Hessian is often close to $\mathcal{O}(d^2)$ for a function in $\mathbb{R}^d$. And, to make matters worse, Newton's method necessitates the inversion of the Hessian, which is in $\mathcal{O}(d^3)$. Since $d = 5$ for this problem, the cost of each Newton-update is about $25\times$ that of a gradient-update! To circumvent the high cost of Newton's method, and to avoid issues with its convergence, a common practice is to run gradient-descent for $K$ iterations, followed by the use of Newton's method to take advantage of the quadratic convergence once the iterates are in the vicinity of a (local) minimum.

   Replicate the aforesaid strategy by running a total of 100 iterations, of which $K \leq 100$ iterations are gradient-based, and the rest are Newton-based. Let each gradient-update have a unit-cost, and consequently, the cost of each Newton-update is 25 units. Experiment with different values of step-sizes and $K$, and track the total cost of the optimisation procedure. Report the least observed cost, the corresponding function-value and $\mathbf{x}$ at the last iteration for that value of $K$. Also, plot the function-values at each iteration, denoting each gradient-step with a blue dot (●), and each Newton-step with a red-coloured cross (×). One may even interleave the gradient and Newton updates, if that leads to lower costs.

## Solution

1. Again define the functions f3_value, f3_gradient and Newton_update_term in the script. Run gradient descent for 100 iterations with a step-size of 0.1 and plot the function values at each iteration. The plot is shown in figure 4.

   The best function value obtained over all the iterations is

   $$\min \text{f3\_values} = 6.1331442462800325$$

   The other statistics are given in the tables 4 and 5.

2. Yes, oscillations are observed in the plot. The possible reasons for this phenomenon are:

   (a) The initial point is too far from the optimal solution.

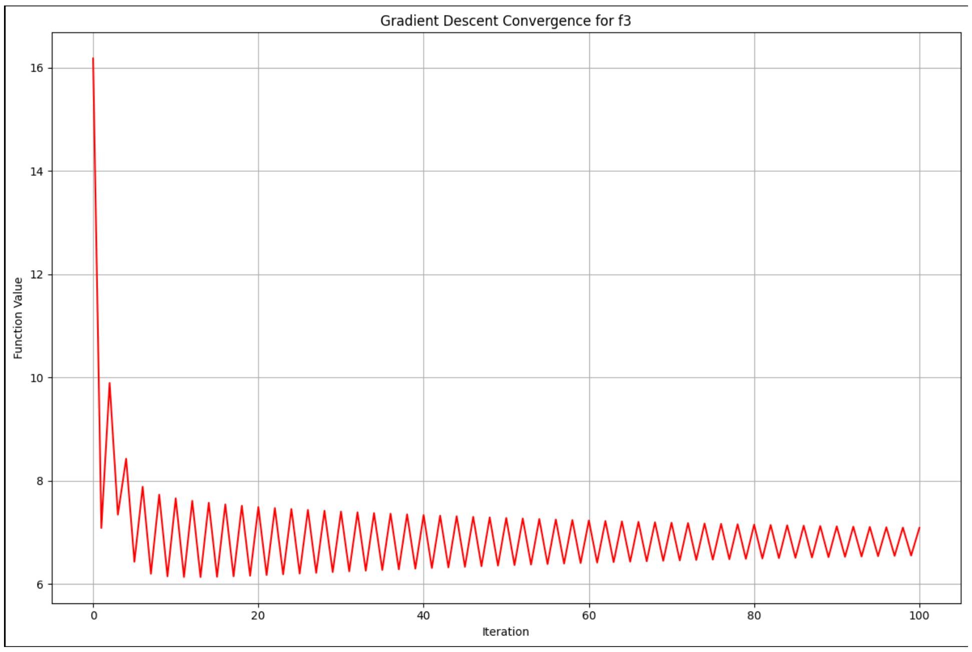


Figure 4: Function value vs iterations for gradient descent

| **Final x** |
|---|
| [-4.79441025e-01, 2.17408789e-17, 2.17408789e-17, 2.17408789e-17, 3.19757550e-05] |

Table 4: Results for Final $x$ and $f(x)$

| **Final f(x)** | **Gradient Norm** |
|---|---|
| 7.08773661132298 | 8.996784413067298 |

Table 5: Results for Final $f(x)$ and Gradient Norm

| **Function Values** |
|---|
| np.float64(16.18137781002638) |
| np.float64(inf) |
| np.float64(inf) |
| np.float64(nan) |
| np.float64(nan) |
| np.float64(nan) |
| np.float64(nan) |
| np.float64(nan) |
| np.float64(nan) |
| np.float64(nan) |

Table 6: Function values for the first 10 iterations

(b) The step-size is too large for the gradient descent to converge to the optimal solution.

(c) The number of iterations is too small for the gradient descent to converge to the optimal solution.

The gradient descent overshoots the optimal solution and oscillates around it. To overcome this, we can try the following:

(a) Choose a smaller step-size.

(b) Choose a better initial point.

(c) Increase the number of iterations.

3. The function values for the first 10 iterations are presented in table 6. The algorithm does not converge.

   One of the possible reasons for the algorithm not converging is that the initial point is to far from the optimal solution. The algorithm diverges and the function values become $\infty$ and NaN. Another possible reason can be that the function is too much non-linear and the Newton's method is not able to converge to the optimal solution.

   We observe that starting at another initial point, $(0.1, 0.1, 0.1, 0.1, 0.1)^T$ the Newton's method converges at a point where $\nabla f(\mathbf{x}) = 0$ and the function value is 3.4657359027997265. This is shown in the figure 5.

   By changing the initial point, one observes that at the origin the Gradient is already 0 and hence origin is the optimal solution. The function value at the origin is 3.4657359027997265. This is shown in the figure 6.

   The reason why Newton's method did not converge at the initial point $(1, 1, 1, 1, 1)^T$ is maybe because the point is too far from the optimal solution. While the latter initial point, $(0.1, 0.1, 0.1, 0.1, 0.1)^T$ is closer to the optimal solution and hence the Newton's method converges to the optimal solution.

4. The plot for step size 0.01 is shown in the figure 7. Similarly the plots for step size 0.02, 0.1 and 0.15 are shown in the figures 8, 9 and 10 respectively.

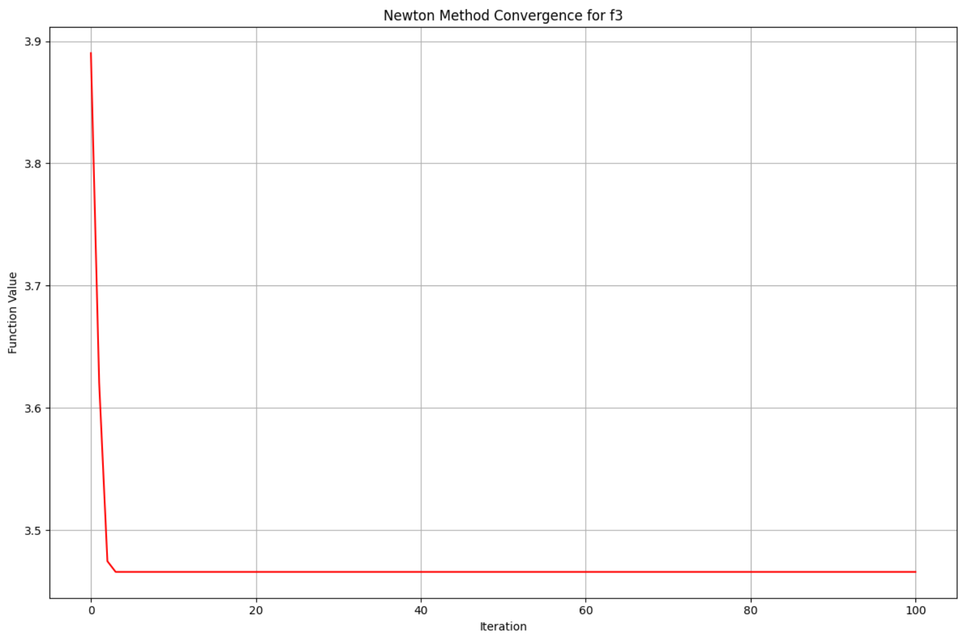


Figure 5: Function value vs iterations for Newton's method

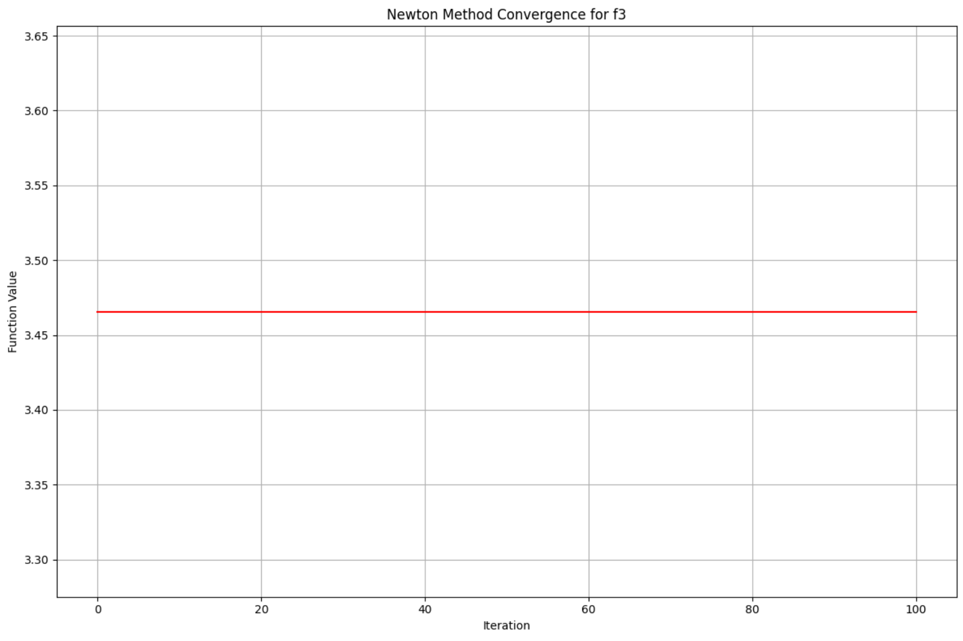
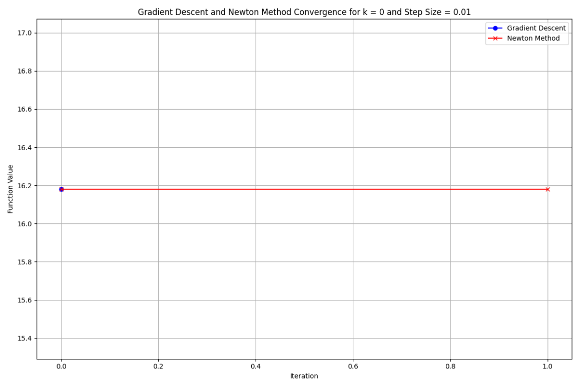
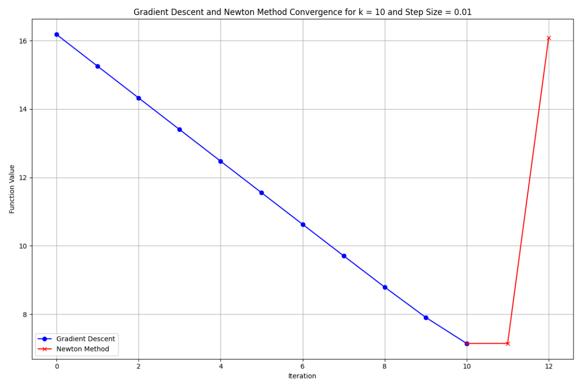


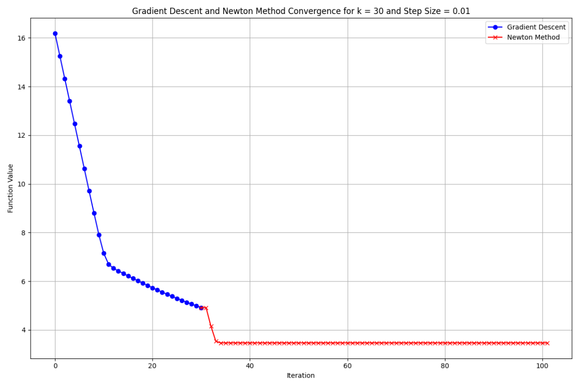
Figure 6: Function value vs iterations for Newton's method



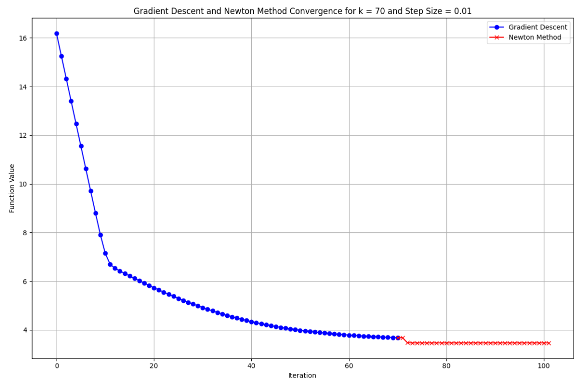
$K = 0$



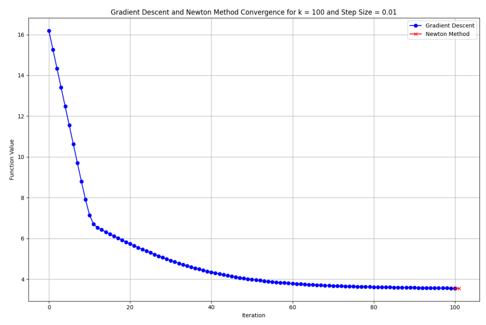
$K = 10$



$K = 30$

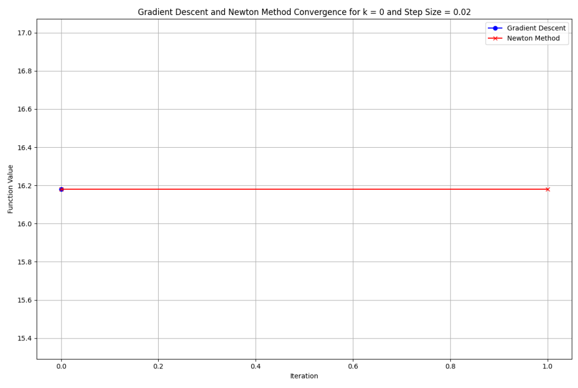


$K = 70$

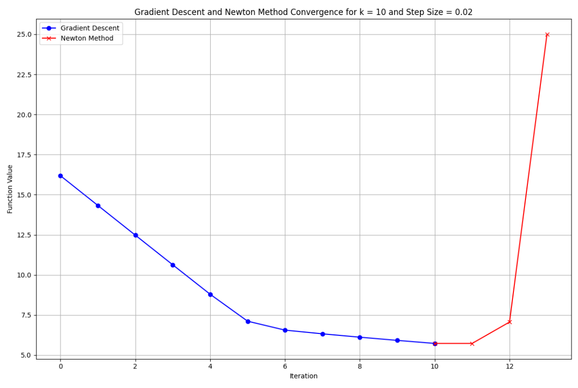


$K = 100$

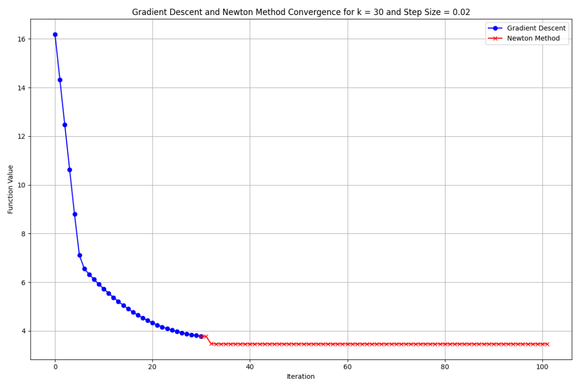
Figure 7: Step size = 0.01



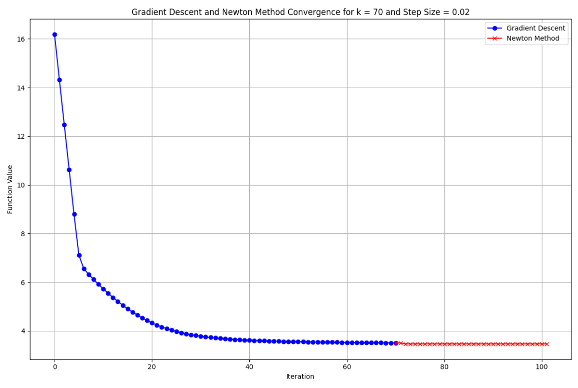
$K = 0$



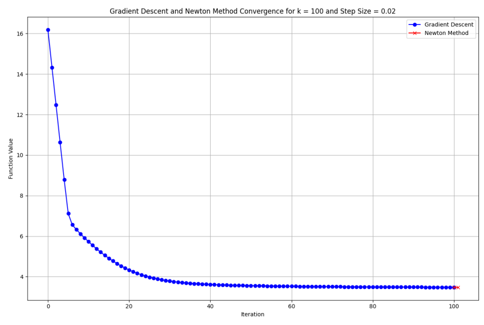
$K = 10$



$K = 30$

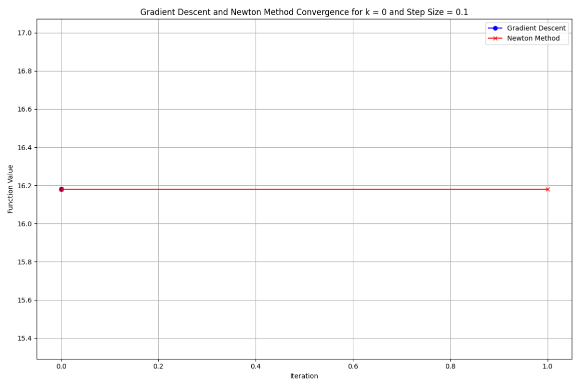


$K = 70$

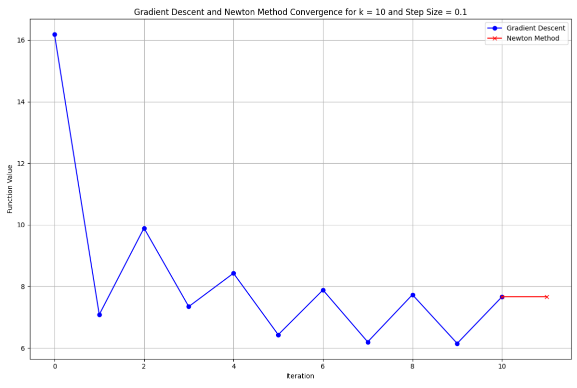


$K = 100$

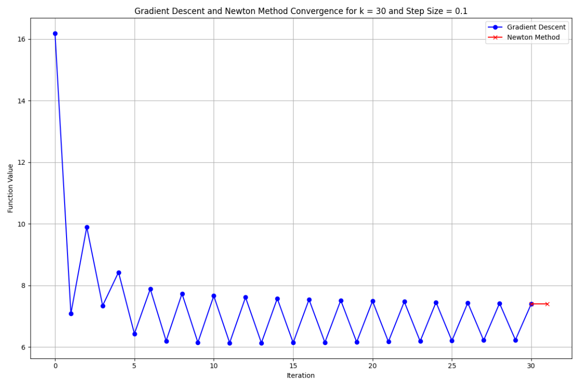
Figure 8: Step size = 0.02



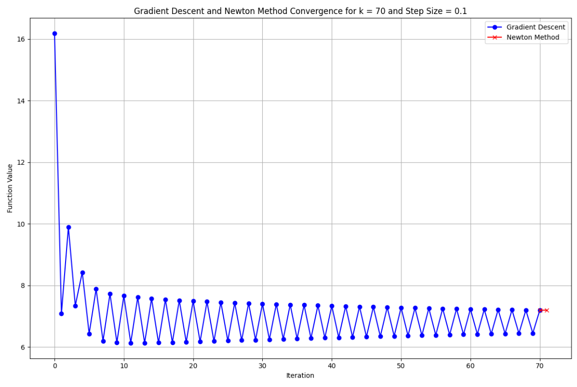
$K = 0$



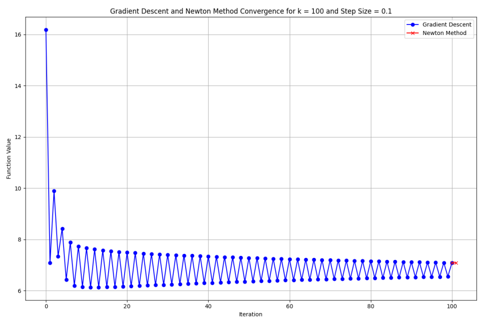
$K = 10$



$K = 30$

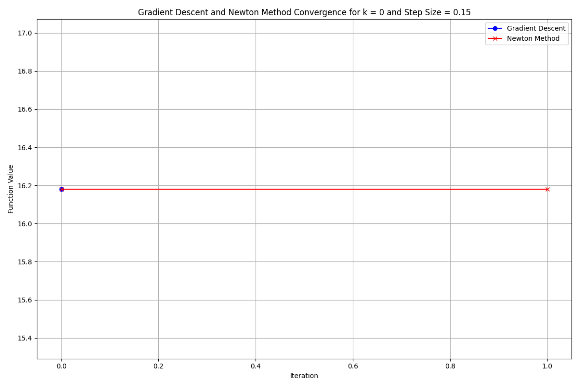


$K = 70$

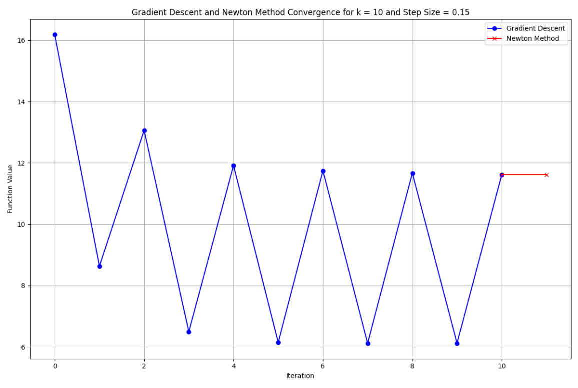


$K = 100$

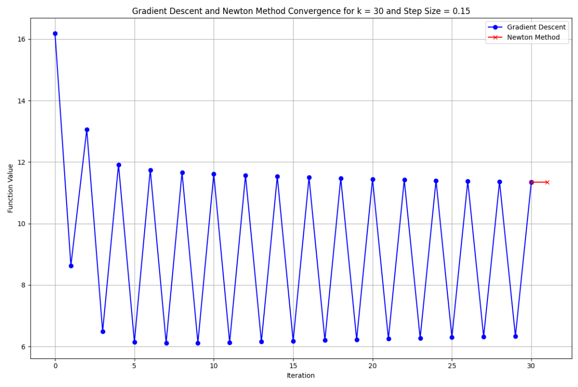
Figure 9: Step size = 0.1



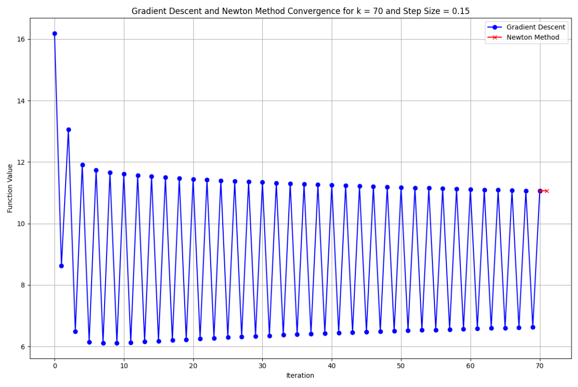
$K = 0$



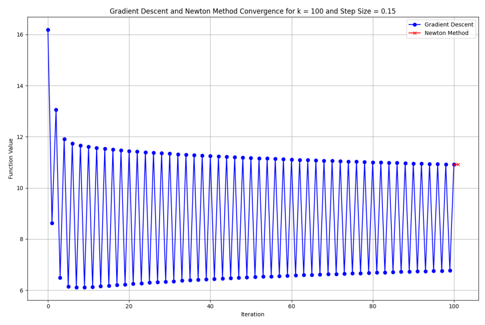
$K = 10$



$K = 30$



$K = 70$



$K = 100$

Figure 10: Step size $= 0.15$

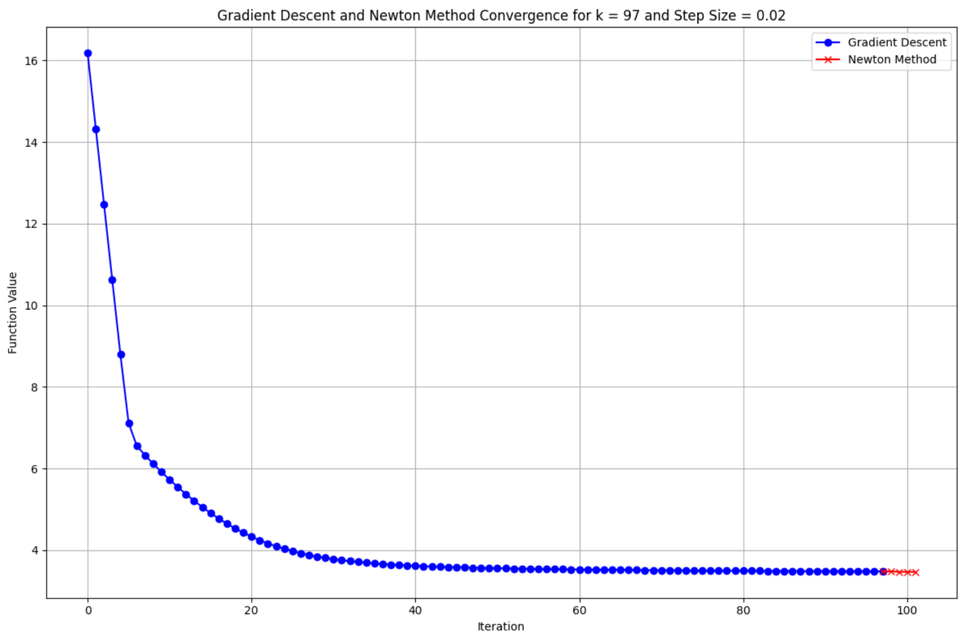


Figure 11: Function value vs iterations for step size = 0.02 and K = 97

The algorithm converges only for (step size = 0.01, K = 30), (step size = 0.01, K = 70), (step size = 0.02, K = 30), (step size = 0.01, K = 70).[1] The cost for a general $K$ is

$$\text{Cost} = 1 \times K + (100 - K) \times 25 = 2500 - 24K$$

Therefore, the least observed cost is for $K = 70$ at $2500 - 24 \times 70 = 2500 - 1680 = 820$ corresponding to step size 0.01 and $K = 70$ the final $\mathbf{x}$ is

$$\mathbf{x} = \begin{pmatrix} -1.12433801e-18 \\ 7.55614269e-18 \\ 7.55614269e-18 \\ 7.55614269e-18 \\ 2.95094984e-17 \end{pmatrix}$$

and the function value is 3.4657359027997265. In case of step size = 0.02 and $K = 70$, the final $\mathbf{x}$ is

$$\mathbf{x} = \begin{pmatrix} -2.11787047e-18 \\ 1.16592497e-17 \\ 1.16592497e-17 \\ 1.16592497e-17 \\ -3.68628739e-18 \end{pmatrix}$$

and the function value is 3.4657359027997265.

The method also converges for step size 0.02 and $K = 97$ as shown in the figure 11. The cost in this case is $2500 - 24 \times 97 = 2500 - 2328 = 172$ and the final $\mathbf{x}$ is

[1] Checked by seeing that the norm of the Gradient at the final $\mathbf{x}$ is 0.

$$\mathbf{x} = \begin{pmatrix} 4.54346768e-18 \\ 7.86195416e-18 \\ 7.86195416e-18 \\ 7.86195416e-18 \\ -3.51378383e-18 \end{pmatrix}$$

while the function value is 3.4657359027997265.

Comparing these results with the results of the Newton's method in subquestion 3, we observe that the cost is significantly reduced by using the hybrid method. The hybrid method is able to converge to the optimal solution in a lesser number of iterations and hence the cost is reduced. The hybrid method is able to take advantage of the quadratic convergence of Newton's method once the iterates are in the vicinity of the local minimum. The lowest cost is observed for step size 0.02 and $K = 97$ as 172.

## 4 Quasi-Newton Methods (10 points)

In this question, we will explore Quasi-Newton methods. Any approximation of the Hessian in the secant approximation can be considered a Quasi-Newton method.

1. There may be cases where you are severely constrained by memory and you can only store one extra variable apart from two values of $\mathbf{x}$ and two values of $\nabla f(\mathbf{x})$. A good idea here is to approximate the Hessian by a scalar multiple of the identity matrix. Derive an update expression for this approach. You may find two solutions.
2. Use the oracle from question 2. Compare your quasi-Newton solutions from the previous part with gradient descent (try a few different step sizes) and the quasi-Newton rank 1 update as discussed in class. Plot the values of $f$ for 100 iterations for these methods. Report $\mathbf{x}^*$ obtained by each method.

## Solution

1. Let $\mathbf{H} \approx \gamma_k \mathbf{I}$ be the approximation of the Hessian. The Quasi-Newton condition states then:

$$\begin{aligned} \mathbf{H}_{k+1}\mathbf{s}_\mathrm{k} &= \nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}) \\ \implies \gamma_{k+1}\mathbf{I}\mathbf{s}_\mathrm{k} &= \nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}) \\ \implies \gamma_{k+1}\mathbf{s}_\mathrm{k} &= \nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}) \end{aligned}$$

Multiplying both sides by $\mathbf{s}_\mathrm{k}{}^T$ on the left, we get:

$$\begin{aligned} \gamma_{k+1}\mathbf{s}_\mathrm{k}\mathbf{s}_\mathrm{k}{}^T &= \mathbf{s}_\mathrm{k}{}^T(\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1})) \\ \implies \gamma_{k+1} &= \frac{\mathbf{s}_\mathrm{k}{}^T(\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))}{\mathbf{s}_\mathrm{k}{}^T\mathbf{s}_\mathrm{k}} \end{aligned}$$

Hence the update rule for the approximation of the Hessian is

$$\mathbf{H}_{k+1} = \frac{\mathbf{s}_\mathrm{k}{}^T(\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))}{\mathbf{s}_\mathrm{k}{}^T\mathbf{s}_\mathrm{k}}\mathbf{I}$$

The update rule then becomes

$$\begin{aligned} \mathbf{x}_{\mathrm{k}+1} &= \mathbf{x}_\mathrm{k} - \gamma_{k+1}^{-1}\mathbf{I}\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) \\ &= \mathbf{x}_\mathrm{k} - \frac{\mathbf{s}_\mathrm{k}{}^T\mathbf{s}_\mathrm{k}}{\mathbf{s}_\mathrm{k}{}^T(\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla\mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))}\nabla\mathbf{f}(\mathbf{x}_\mathrm{k}) \end{aligned}$$

Suppose instead of $H$ we want to approximate $H^{-1}$ by a scalar multiple of the identity matrix, $\mathbf{H}_k^{-1} = \gamma_k\mathbf{I}$ Then,

$$\begin{aligned}
&\mathbf{H}_{k+1}^{-1}(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1})) = \mathbf{s}_\mathrm{k} \\
\implies &\gamma_{k+1}^{-1}\mathbf{I}(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1})) = \mathbf{s}_\mathrm{k} \\
\implies &\gamma_{k+1}^{-1}(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1})) = \mathbf{s}_\mathrm{k}
\end{aligned}$$

Multiplying both sides by $(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T$ on the left, we get:

$$\implies \gamma_{k+1}^{-1} = \frac{\mathbf{s}_\mathrm{k}(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T}{(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T}$$

Hence the update rule for the approximation of the inverse of the Hessian is

$$\mathbf{H}_{k+1}^{-1} = \frac{\mathbf{s}_\mathrm{k}(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T}{(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T}\mathbf{I}$$

The update rule then becomes

$$\begin{aligned}
\mathbf{x}_{\mathrm{k}+1} &= \mathbf{x}_\mathrm{k} - \gamma_{k+1}\mathbf{I}\mathbf{s}_\mathrm{k} \\
&= \mathbf{x}_\mathrm{k} - \frac{(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T\mathbf{s}_\mathrm{k}}{(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))(\nabla \mathbf{f}(\mathbf{x}_\mathrm{k}) - \nabla \mathbf{f}(\mathbf{x}_{\mathrm{k}-1}))^T}\mathbf{s}_\mathrm{k}
\end{aligned}$$

2. Based on the above two methods, two Quasi Newton methods named as custom_Newton_1 and custom_Newton_2 are implemented in the script. The function values at each iteration for these methods are plotted in the figures 12 and 13 respectively. The Rank-1 update method is also implemented in the script as Quasi_Newton_Rank_1 and the function values at each iteration for this method is plotted in the figure 14.

   All three methods, custom_Newton_1, custom_Newton_2, and the Quasi_Newton_Rank_1, converged to the optimal solution $\mathbf{x}^* = [1, 1, 1, 1, 1]^T$ with a function value of $-4.5$ in just 3 iterations. These methods effectively took advantage of second-order information, allowing for faster convergence. In contrast, gradient descent required 100 iterations and a higher step-size of 0.5 (refer to 1 and 1) to reach the same solution. This highlights the efficiency of Quasi-Newton methods compared to gradient descent, which took more time and required more iterations.

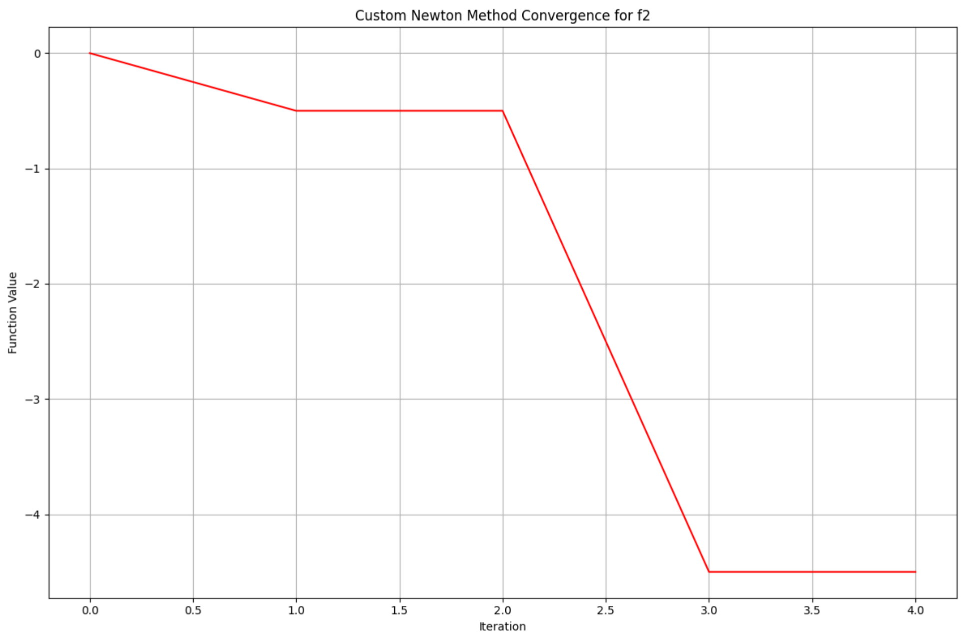


Figure 12: Function value vs iterations for custom Newton 1

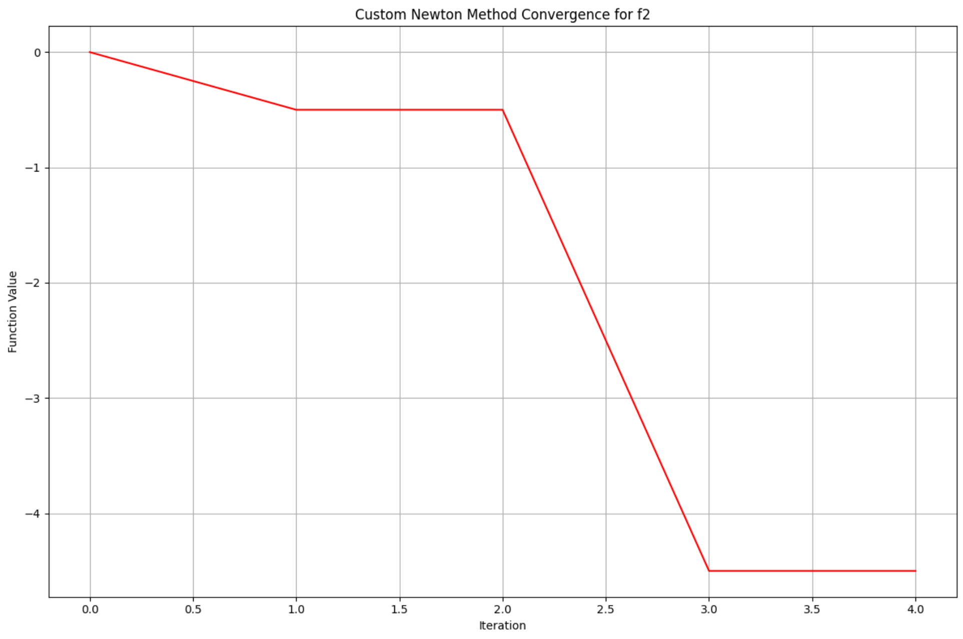


Figure 13: Function value vs iterations for custom Newton 2

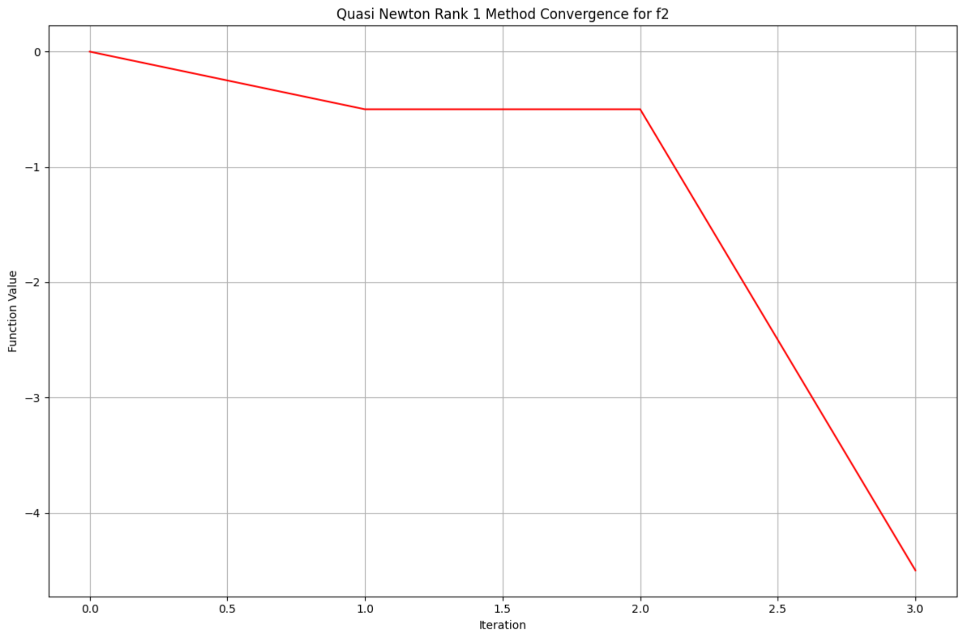


Figure 14: Function value vs iterations for Rank-1 update