CMO - Assignment - 2

Nirbhay sharma
SR: 24806
MTech - AI

October 20, 2024

1 Que-1 - Conjugate gradient Descent

1.1 solution for $Ax = b$

we are given A as $n \times n$ symmetric PD matrix which implies that all eigenvalues of A are strictly positive i.e.

$\lambda_1, \lambda_2 ... \lambda_n$ are the eigenvalues of A st $Av_i = \lambda_i v_i$ $v_i \neq 0$ and $\lambda_i > 0$ $\forall i$, we know that $\det(A) = \Pi_{i=1}^{n} \lambda_i \neq 0$ hence A^{-1} exists, therefore $Ax = b$ has a unique solution $x = A^{-1}b$

the quadratic convex optimization problem for solving $Ax = b$ is as follows

$$\min_{x \in R^n} f(x) = \min_{x \in R^n} \frac{1}{2} x^T A x - b^T x$$

we know that at x^*, $\nabla f(x^*) = 0 \implies Ax^* - b = 0 \implies Ax^* = b$ hence x^* becomes the solution of $Ax = b$

1.2 implementation of conjugate gradient

we implement the conjugate gradient descent algorithm as follows

$$g^k = \nabla f(x^k)$$

$$u^0 = -g^0$$

$$x^{k+1} = x^k + \alpha_k u^k; \alpha_k = \frac{-g^{kT} u^k}{u^{kT} Q u^k}$$

$$u^{k+1} = -g^{k+1} + \beta_k u^k; \beta_k = \frac{g^{(k+1)T} Q u^k}{u^{kT} Q u^k}$$

upon implementing the above algorithm for minimizing the quadratic optimization problem in part1, we get the output as shown in Fig 1 in Que1.2, as expected the conjugate gradient algorithm converges in 5 steps which is the dimension of A

1.3 solution for $Ax = b$ A is $m \times n$ matrix

$Ax = b$ for general $m \times n$ where $m > n$, the solution will exist but need not be unique

to solve the minimization problem $||Ax - b||$, we can see the following equivalent problems

$$x^* = arg \min_{x \in R^n} ||Ax - b|| = arg \min_{x \in R^n} \frac{1}{2} ||Ax - b||^2$$

$$\frac{1}{2}||Ax-b||^2 = \frac{1}{2}(Ax-b)^T(Ax-b) = \frac{1}{2}(x^T A^T - b^T)(Ax-b) = \frac{1}{2}(x^T A^T Ax - 2b^T Ax + b^T b) = \frac{1}{2}x^T A^T Ax - b^T Ax + \frac{1}{2}b^T b$$

$$x^* = arg \min_{x \in R^n} \frac{1}{2} x^T A^T A x - b^T A x + \frac{1}{2} b^T b$$

the solution for the above equation would occur at $\nabla f(x) = 0$

$$\nabla f(x) = A^T A x - A^T b = 0 \implies A^T A x = A^T b$$

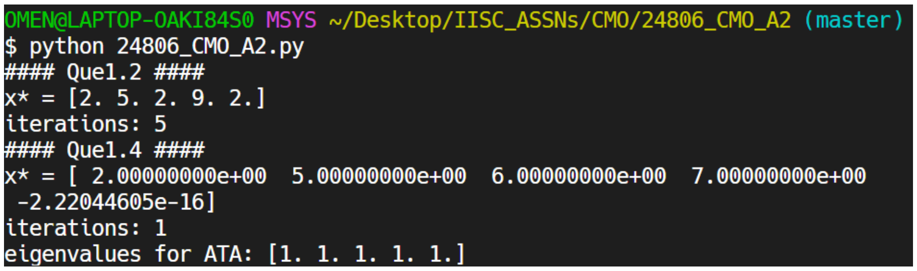
The unique solution for this occurs when $A^T A$ is PD and the unique solution is given by

$$x^* = (A^T A)^{-1} A^T b$$

1.4 part 4

on solving the above quadratic optimization problem using conjugate gradient algorithm, we get optimal x^* and iterations as shown in Fig 1 in Que1.4, the algorithm converges in 1 step because the eigenvalues of $A^T A$ are all positive and they are same, i.e. there is only 1 distinct eigenvalue, hence CG algorithm converges in 1 step.

1.5 Que-1 code output



```
OMEN@LAPTOP-OAKI84S0 MSYS ~/Desktop/IISC_ASSNs/CMO/24806_CMO_A2 (master)
$ python 24806_CMO_A2.py
#### Que1.2 ####
x* = [2. 5. 2. 9. 2.]
iterations: 5
#### Que1.4 ####
x* = [ 2.00000000e+00  5.00000000e+00  6.00000000e+00  7.00000000e+00
 -2.22044605e-16]
iterations: 1
eigenvalues for ATA: [1. 1. 1. 1. 1.]
```

Figure 1: output for part2 and part4

2 Newton Method

2.1 Gradient Descent algorithm

We run Gradient descent algorithm on $f2$ with different values of $\alpha = (10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5})$ and the value of $f2(x)$ is plotted for each α as shown in Fig 2, the convergence is achieved on $\alpha = 0.1, 0.01$, the x^* are reported in Fig 5 in Que-2.1

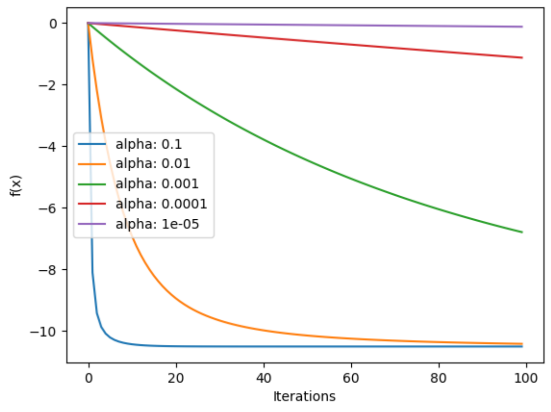


Figure 2: output for part1

2.2 Newton algorithm

We run the newton update for 100 iterations, which is as follows. The function values against iterations are shown in Fig 3

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

comparing Newton and Gradient descent algorithm, we observe that Newton method converge in one step while gradient descent algorithm iteratively converges.

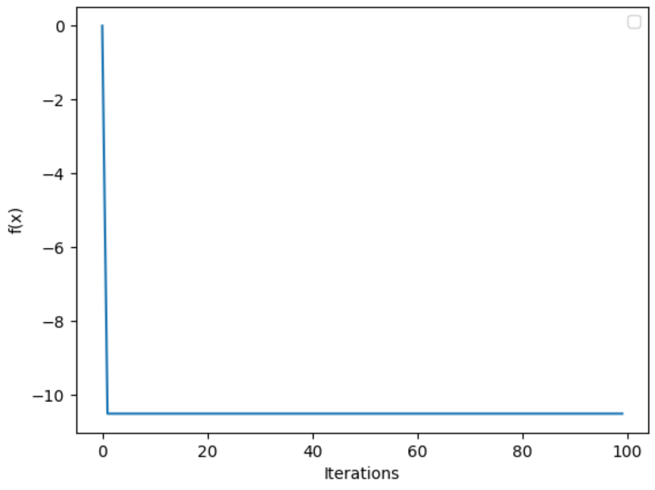


Figure 3: output for newton method

2.3 Newton method for different x^0

We take five different x^0 values and run Newton method, the plot for function values is shown in Fig 4 and the corresponding x^* is reported in Fig 5 in Que-2.3

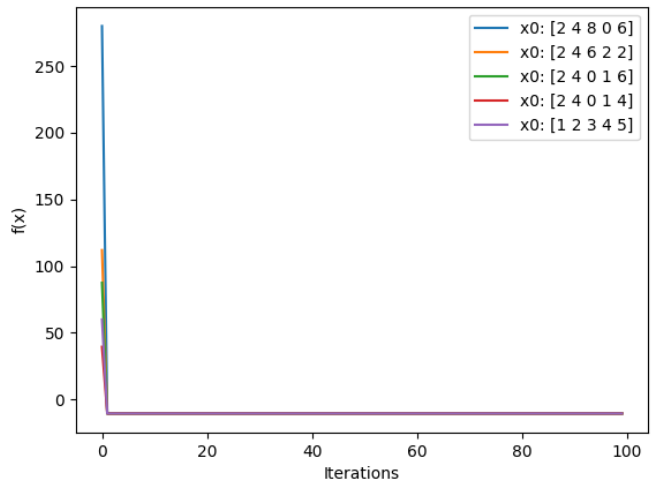


Figure 4: output for newton method for different x^0

We observe the following from the plot, for random 5 values of x^0 the Newton algorithm converges in one step for every x^0, which means the Hessian for the function is PD

2.4 Nature of Oracle $f2$

since the Newton method is converging in one step everytime, we can infer that $f2$ is a quadratic function with hessian PD, the $f2$ must have the form

$$f2(x) = \frac{1}{2}x^T Q x + b^T x + c; Q \succ 0$$

now for this type of function, we show that no matter which initial point x^0 you start from, Newton method will converge in one step

first we will show the optimal x^* for this form, since Q is PD, optimal x^* is given by $\nabla f(x^*) = 0$

$$\nabla f(x) = Qx + b$$

$$\nabla f(x^*) = Qx^* + b = 0 \implies Qx^* = -b \implies x^* = -Q^{-1}b$$

now consider the Newton update, starting from arbitrary x^0

$$x^1 = x^0 - H(x^0)^{-1}\nabla f(x^0)$$

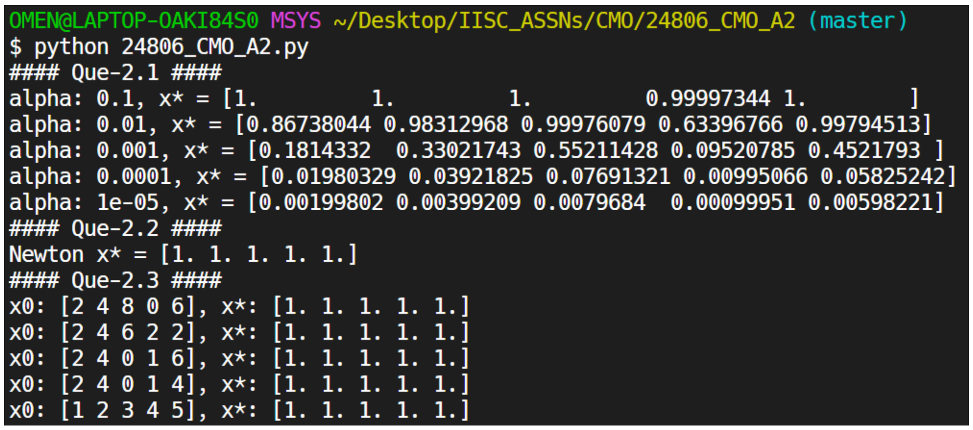
$$\nabla f(x^0) = Qx^0 + b$$

$$H(x^0)^{-1} = Q^{-1}$$

$$x^1 = x^0 - Q^{-1}(Qx^0 + b) = x^0 - Q^{-1}Qx^0 - Q^{-1}b = x^0 - x^0 - Q^{-1}b = -Q^{-1}b = x^*$$

hence it converges in one step only.

2.5 Que-2 code output



```
OMEN@LAPTOP-OAKI84S0 MSYS ~/Desktop/IISC_ASSNs/CMO/24806_CMO_A2 (master)
$ python 24806_CMO_A2.py
#### Que-2.1 ####
alpha: 0.1, x* = [1.         1.         1.         0.99997344 1.        ]
alpha: 0.01, x* = [0.86738044 0.98312968 0.99976079 0.63396766 0.99794513]
alpha: 0.001, x* = [0.1814332  0.33021743 0.55211428 0.09520785 0.4521793 ]
alpha: 0.0001, x* = [0.01980329 0.03921825 0.07691321 0.00995066 0.05825242]
alpha: 1e-05, x* = [0.00199802 0.00399209 0.0079684  0.00099951 0.00598221]
#### Que-2.2 ####
Newton x* = [1. 1. 1. 1. 1.]
#### Que-2.3 ####
x0: [2 4 8 0 6], x*: [1. 1. 1. 1. 1.]
x0: [2 4 6 2 2], x*: [1. 1. 1. 1. 1.]
x0: [2 4 0 1 6], x*: [1. 1. 1. 1. 1.]
x0: [2 4 0 1 4], x*: [1. 1. 1. 1. 1.]
x0: [1 2 3 4 5], x*: [1. 1. 1. 1. 1.]
```

Figure 5: code output for que2

3 Newton Method Continued

3.1 Gradient Descent on f3

we run the gradient descent algorithm on $f3$ for 100 iterations, the $f3(x)$ vs iterations is shown in Fig 6 and the minimum $f3(x)$ value is reported in Fig 14

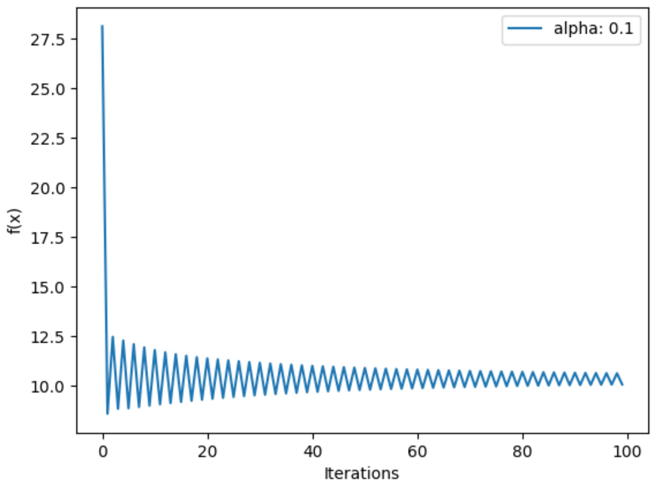


Figure 6: Gradient descent with $\alpha = 0.1$

3.2 Oscillations in Gradient Descent

as shown in Fig 6, there are indeed oscillations in the function values, the possible reason for the oscillations could be high value of α which leads to oscillations in the function value, as it moves away from minima in one step and in another step it comes closer to minima, hence function value oscillates. Infact we tried with more higher values of α, the oscillation is even higher as shown in Fig 7

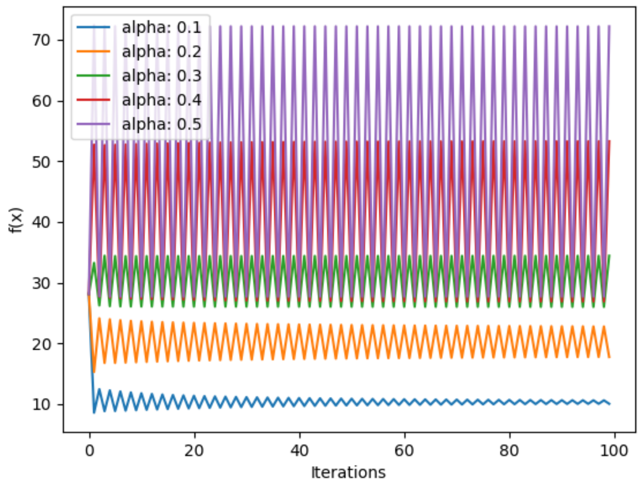


Figure 7: $f3(x)$ for different values of α

One possible way to overcome oscillations, is to use a small value of α. We tried with $\alpha = 0.01, 0.02, 0.009$ and the result is demonstrated in Fig 8 which indeed removes the oscillations.

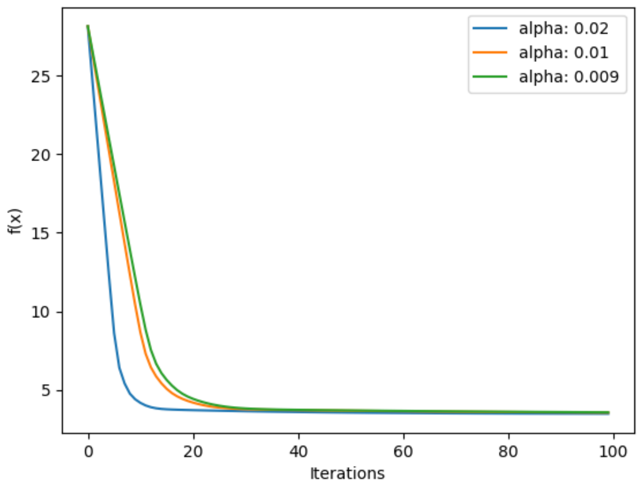


Figure 8: $f3(x)$ for $\alpha = 0.01, 0.02, 0.009$

3.3 Convergence in Newton Method

We run the Newton method for 100 iterations for $f3$, the algorithm did not converge and the function value become *nan* after few iterations as shown in Fig 14. Hence Newton does not converge for this function, the possible reason could be the Hessian is not invertable at some point and hence the function value become *nan*

3.4 Gradient descent + Newton

we experimented with different values of K, α as shown in Fig [9, 10, 11, 12, 13], We tried with different values of $k = [60, 80, 85, 90, 100]$ and different values of $\alpha = [0.01, 0.02, 0.1, 0.2]$, from Fig 14, out of all those K's min $f3(x)$ is achieved at $K = 90, \alpha = 0.01$ with cost 340, also notice that for some values of alpha, the Gradient descent followed by newton does not converge for examples for $alpha = 0.2$ the algorithm did not converge, which is again due to the Hessian not being PD. We show plots for alphas where the algorithm converges and where not.

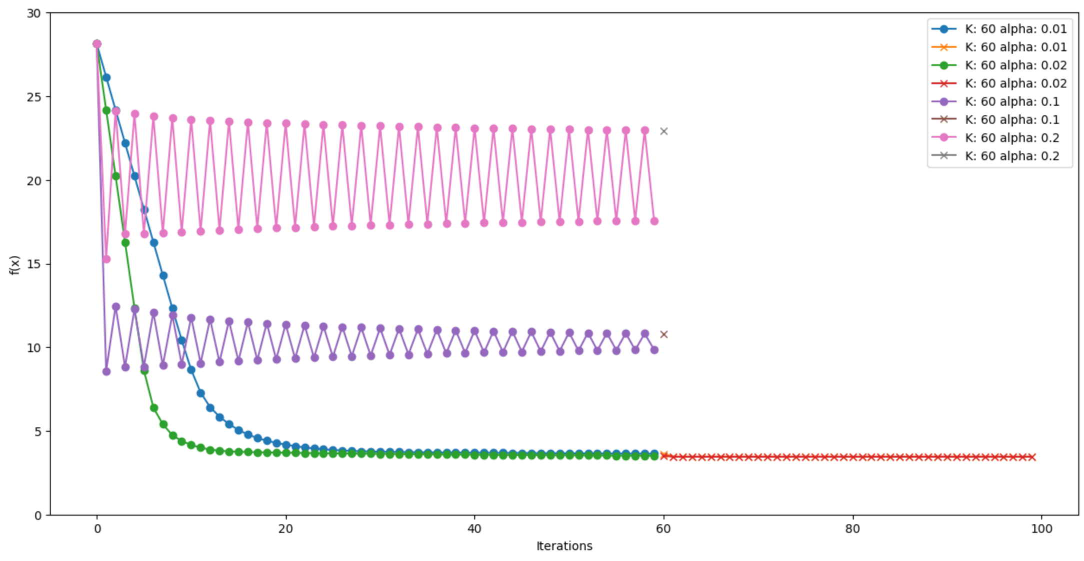


Figure 9: $f3(x)$ for $K = 60$

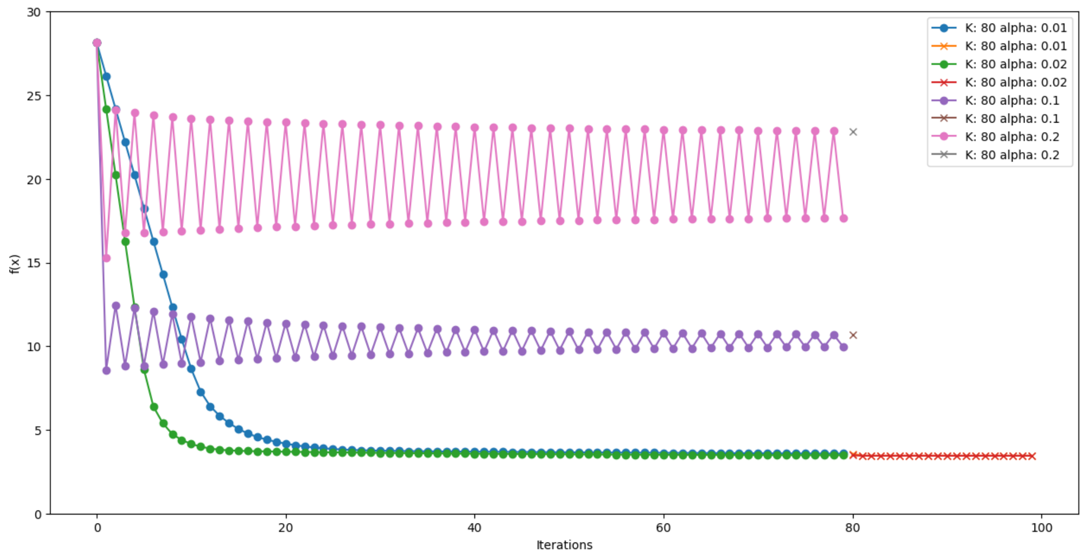


Figure 10: $f3(x)$ for $K = 80$

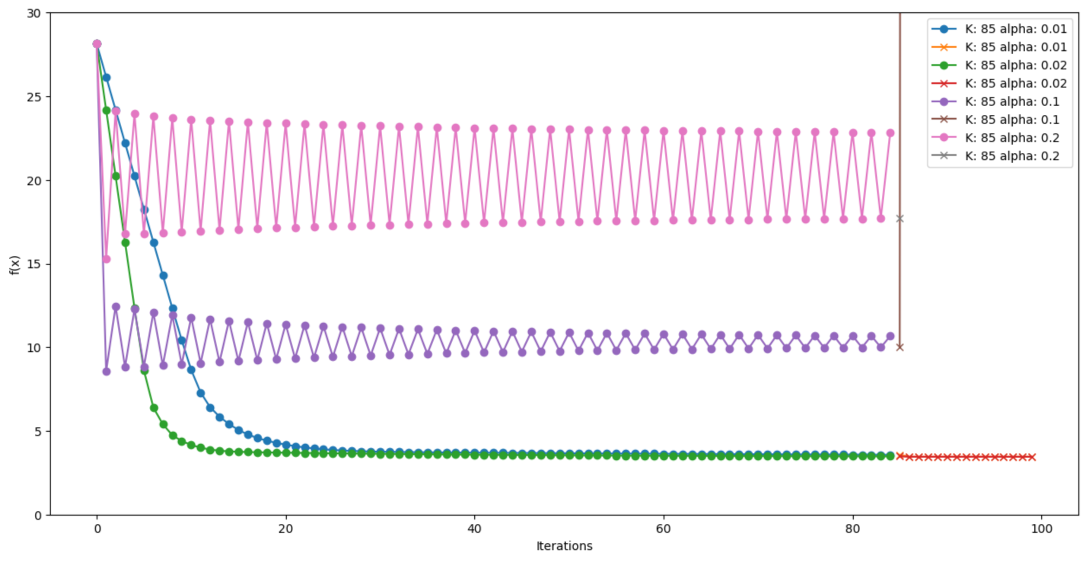


Figure 11: $f3(x)$ for $K = 85$

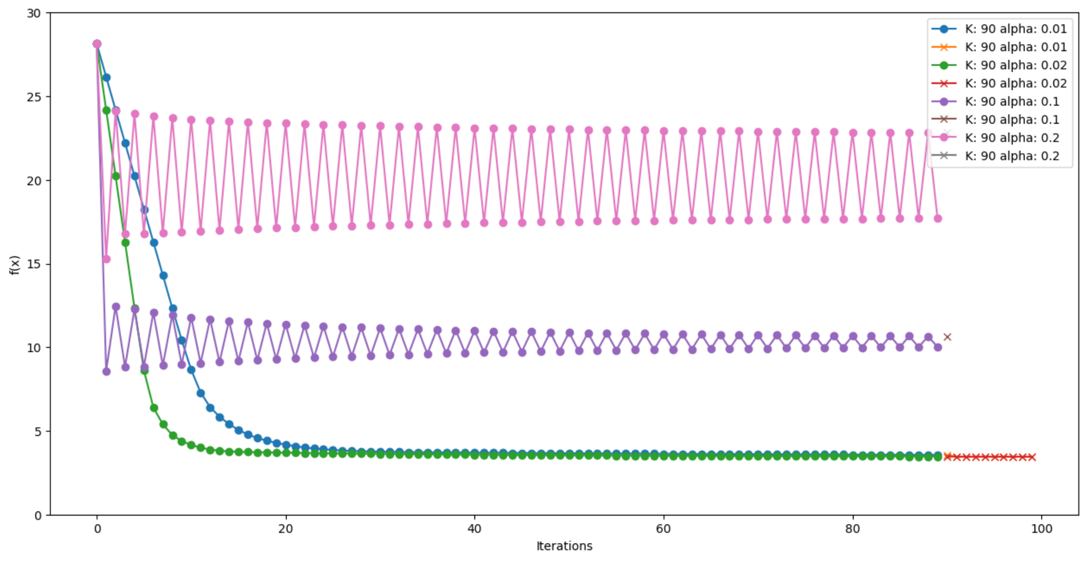


Figure 12: $f3(x)$ for $K = 90$

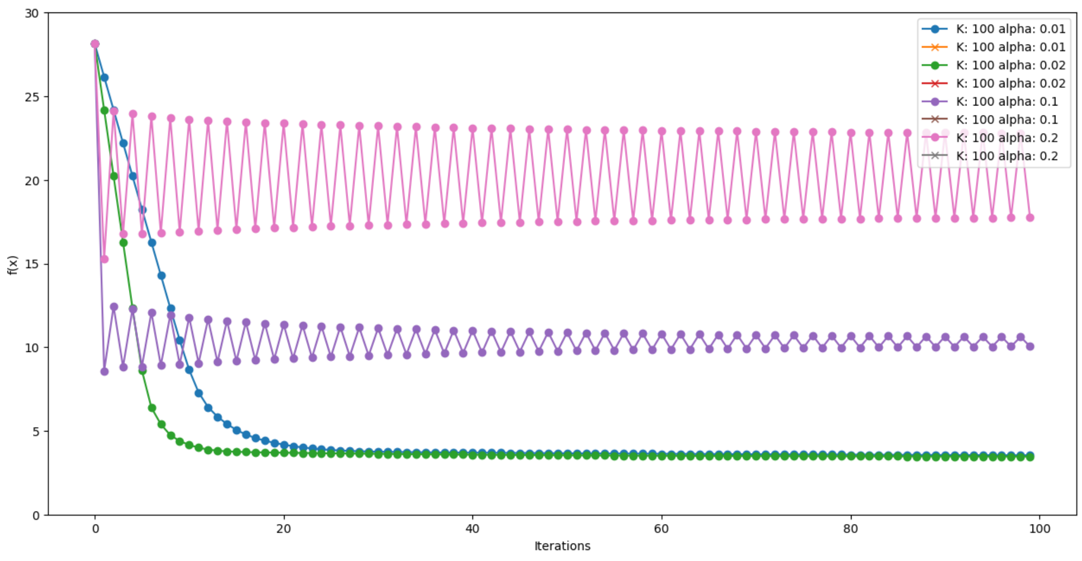
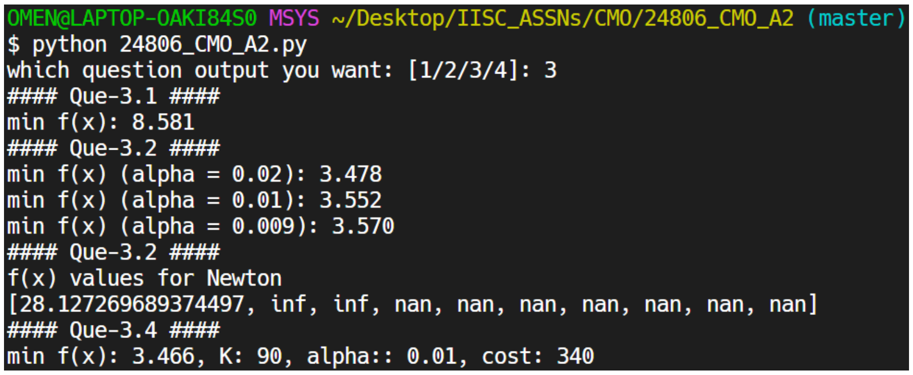


Figure 13: $f3(x)$ for $K = 100$

3.5 Que3 code output



```
OMEN@LAPTOP-OAKI84S0 MSYS ~/Desktop/IISC_ASSNs/CMO/24806_CMO_A2 (master)
$ python 24806_CMO_A2.py
which question output you want: [1/2/3/4]: 3
#### Que-3.1 ####
min f(x): 8.581
#### Que-3.2 ####
min f(x) (alpha = 0.02): 3.478
min f(x) (alpha = 0.01): 3.552
min f(x) (alpha = 0.009): 3.570
#### Que-3.2 ####
f(x) values for Newton
[28.127269689374497, inf, inf, nan, nan, nan, nan, nan, nan, nan]
#### Que-3.4 ####
min f(x): 3.466, K: 90, alpha:: 0.01, cost: 340
```

Figure 14: code output for que3

4 Quasi - Newton method

4.1 Derivation of updated quasi newton method

The notations are as follows

$$g^{(k)} = \nabla f(x^{(k)}) \tag{1}$$
$$\delta^{(k)} = x^{(k+1)} - x^{(k)} \tag{2}$$
$$\gamma^{(k)} = g^{(k+1)} - g^{(k)} \tag{3}$$

we know that

$$H(x^{(k)})\delta^{(k)} = \gamma^{(k)}$$

We want to mimic $H(x^{(k)}) = \lambda_k I$, putting it in the equation we get

$$H(x^{(k)})\delta^{(k)} = \gamma^{(k)} \implies \lambda_k I \delta^{(k)} = \gamma^{(k)} \tag{4}$$

$$\lambda_k \delta^{(k)T} I \delta^{(k)} = \delta^{(k)T}\gamma^{(k)} \implies \lambda_k = \frac{\delta^{(k)T}\gamma^{(k)}}{\delta^{(k)T}\delta^{(k)}} \tag{5}$$

$$H(x^{(k)}) = \frac{\delta^{(k)T}\gamma^{(k)}}{\delta^{(k)T}\delta^{(k)}} I \implies H(x^{(k)})^{-1} = \frac{\delta^{(k)T}\delta^{(k)}}{\delta^{(k)T}\gamma^{(k)}} I \tag{6}$$

another solution for λ_k can be found as follows

$$\lambda_k I \delta^{(k)} = \gamma^{(k)} \implies \lambda_k \gamma^{(k)T} I \delta^{(k)} = \gamma^{(k)T}\gamma^{(k)} \implies \lambda_k = \frac{\gamma^{(k)T}\gamma^{(k)}}{\gamma^{(k)T}\delta^{(k)}} \tag{7}$$

4.2 Quasi - Newton using Rank 1 update

$$u^{(k)} = -G^{(k)} g^{(k)} \tag{8}$$

$$\alpha_k = -\frac{g^{(k)T} u^{(k)}}{u^{(k)T} Q u^{(k)}} \tag{9}$$

$$x^{(k+1)} = x^{(k)} + \alpha_k u^{(k)} \tag{10}$$

$$G^{(k+1)} = G^{(k)} + \frac{(\delta^{(k)} - G^{(k)}\gamma^{(k)})(\delta^{(k)} - G^{(k)}\gamma^{(k)})^T}{(\delta^{(k)} - G^{(k)}\gamma^{(k)})^T \gamma^{(k)}} \tag{11}$$

4.3 Quasi - Newton using $\lambda_k I$ update

$$G^{(0)} = I \tag{12}$$

$$u^{(k)} = -G^{(k)} g^{(k)} \tag{13}$$

$$x^{(k+1)} = x^{(k)} + \alpha_k u^{(k)} \tag{14}$$

$$G^{(k+1)} = \frac{\delta^{(k)T}\delta^{(k)}}{\delta^{(k)T}\gamma^{(k)}} I \tag{15}$$

4.4 finding $u^{(k)T} Q u^{(k)}$ from oracle

$$f2(x) = \frac{1}{2} x^T Q x + b^T x \tag{16}$$

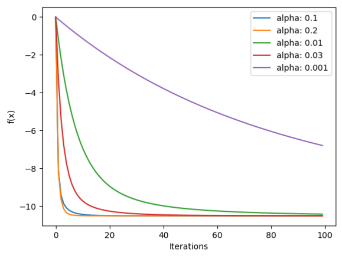
$$\nabla f2(0) = b \tag{17}$$

$$f2(u^{(k)}) = \frac{1}{2} u^{(k)T} Q u^{(k)} + b^T u^{(k)} \tag{18}$$

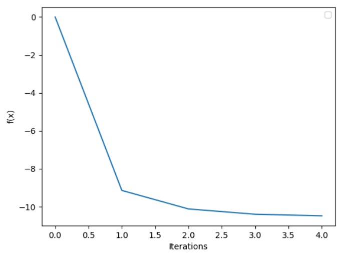
$$u^{(k)T} Q u^{(k)} = 2(f2(u^{(k)}) - b^T u^{(k)}) \tag{19}$$

4.5 Comparison between Gradient Descent, Quasi Newton Rank 1 and Quasi Newton Updated

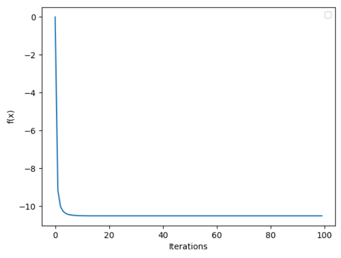
The comparison for different values of α (Gradient descent) is shown in Fig 15, Quasi Newton Rank 1 and Quasi Newton with multiple of Identity uses α_k using exact line search. As expected Quasi Newton Rank 1 update converges in 5 iterations, as shown in Fig 16, Gradient Descent, Quasi Newton Rank1, Quasi Newton Identity all converges to optimal point.



(a) Gradient Descent



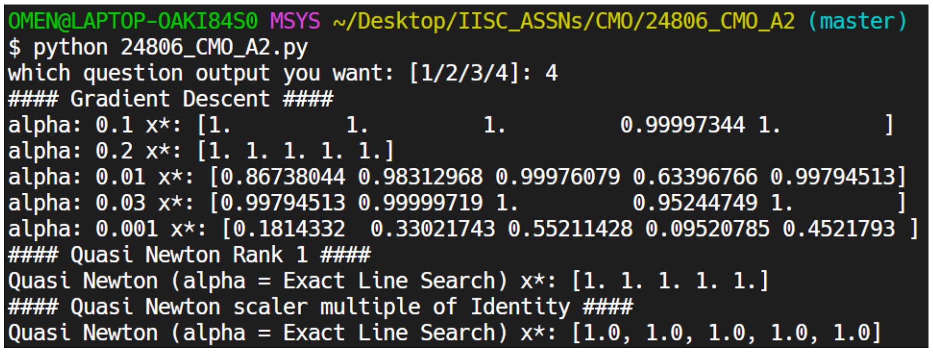
(b) Quasi Newton Rank 1



(C) Quasi Newton Updated

Figure 15: $f2(x)$ vs iterations for different values of α

4.6 Que-4 code output



```
OMEN@LAPTOP-OAKI84S0 MSYS ~/Desktop/IISC_ASSNs/CMO/24806_CMO_A2 (master)
$ python 24806_CMO_A2.py
which question output you want: [1/2/3/4]: 4
#### Gradient Descent ####
alpha: 0.1 x*: [1.         1.         1.         0.99997344 1.        ]
alpha: 0.2 x*: [1. 1. 1. 1. 1.]
alpha: 0.01 x*: [0.86738044 0.98312968 0.99976079 0.63396766 0.99794513]
alpha: 0.03 x*: [0.99794513 0.99999719 1.         0.95244749 1.        ]
alpha: 0.001 x*: [0.1814332  0.33021743 0.55211428 0.09520785 0.4521793 ]
#### Quasi Newton Rank 1 ####
Quasi Newton (alpha = Exact Line Search) x*: [1. 1. 1. 1. 1.]
#### Quasi Newton scaler multiple of Identity ####
Quasi Newton (alpha = Exact Line Search) x*: [1.0, 1.0, 1.0, 1.0, 1.0]
```

Figure 16: code output for que4