CMO Assignment 2

SWAPNIL MITESHKUMAR JOSHI SWAPNILJOSHI@IISC.AC.IN
SR number: 25846

Question 1

[1.1]:

Let Q be a symmetric positive definite matrix and $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\mathsf{T}Q\mathbf{x} - \mathbf{b}^\mathsf{T}\mathbf{x}$. Suppose we generate directions $\{\mathbf{u}_k\}$ using the Gram-Schmidt recursion with the Q-inner product:

$$\mathbf{d}_{k+1} = \mathbf{p}_{k+1} - \sum_{i=0}^{k} \frac{\mathbf{p}_{k+1}^\mathsf{T} Q\mathbf{d}_i}{\mathbf{d}_i^\mathsf{T} Q\mathbf{d}_i}\mathbf{d}_i,$$

where $\{\mathbf{p}_k\}$ are the raw residuals or search directions.

[1.1.A]:

Are the vectors $\{\mathbf{d}_k\}$ produced by this recursion Q-conjugate? Justify briefly.

Yes, the vectors $\{\mathbf{d}_k\}$ produced by this recursion are **Q-conjugate** (**Q**-orthogonal).

Explanation: This recursion essentially applies the **Gram-Schmidt process** with respect to the **Q**-inner product, defined as $\langle \mathbf{x}, \mathbf{y}\rangle_\mathbf{Q} = \mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{y}$. At each step, the projection of $\mathbf{p}_{k+1}$ onto the span of the previous vectors $\mathbf{d}_0, \ldots, \mathbf{d}k$ is subtracted, which guarantees that the new vector $\mathbf{d}k + 1$ is orthogonal to all previous directions under this inner product. Since we know that **Q** is symmetric and positive definite, **Q**-orthogonality directly implies **Q**-conjugacy. Hence, we can say that the set $\mathbf{d}_k$ is indeed **Q**-conjugate.

[1.1.B]:

What happens when $\mathbf{Q} = \mathbf{I}$? Comment on what $\{\mathbf{d}_k\}$ become in this special case.

When the matrix **Q** is replaced by the Identity matrix **I**, the **Q**-inner product simplifies to the **standard Euclidean inner product** (or we can say the dot product):

$$\langle \mathbf{x}, \mathbf{y}\rangle_\mathbf{I} = \mathbf{x}^\mathsf{T}\mathbf{I}\mathbf{y} = \mathbf{x}^\mathsf{T}\mathbf{y}$$

In this situation, the Gram–Schmidt procedure simplifies to the standard, unweighted version. As a result, the vectors $\mathbf{d}_k$ produced by the recursion are simply **orthogonal in the usual sense**. That is, each direction is perpendicular to all the previous ones, so $\mathbf{d}_i^\mathsf{T}\mathbf{d}_j = 0$ whenever $i \neq j$. Essentially, we just get a set of standard orthogonal vectors.

[1.1.C]:

Query the oracle `f2(srno, True)` to obtain A (this plays the role of Q) and b. Implement the Conjugate Directions method (name it `CD_SOLVE`) starting from $\mathbf{x}_0 = \mathbf{0}$. Use the eigenvectors of A as the conjugate directions. At each iteration, compute and print the following quantities for the first 7 steps:

1. the step size α_k,

2. the value $-\nabla f(\mathbf{x}_k)^\mathsf{T}\mathbf{u}_k$,

3. the corresponding eigenvalue λ_k.

CONSOLE OUTPUT:

Step (k)	**Step Size** (α_k)	**Value** ($-\nabla f(\mathbf{x}_k)^\top \mathbf{u}_k$)	**Eigenvalue** (λ_k)
0	0.019884605772667617	0.39778901400941546	20.004873043859693
1	-0.0661228443722912	-1.3374320690293602	20.226475157348357
2	0.06255486948323666	1.2893400826645727	20.61134637983837
3	0.017549598510726377	0.3763493106359347	21.44489575678375
4	0.01591417628827923	0.35440365968076487	22.269682907922945
5	-0.06538331735118803	-1.5683046872264663	23.98631257577159
6	0.05357084733135389	1.3624447725667843	25.432578360009867

Table 1: Results from the Conjugate Directions method for the first 7 steps.

[1.2]:

Query the oracle `f2(srno, True)` to obtain a symmetric positive definite matrix A (this plays the role of Q) and a vector b. Implement the Conjugate Gradient method (name the function `CG_SOLVE`), starting from $\mathbf{x}_0 = \mathbf{0}$ to solve $A\mathbf{x} = \mathbf{b}$.

Number of Directions Computed (**m**)

The algorithm converged in 13 iterations.
The number of **CG** search directions computed (**m**) is:

$$\mathbf{m = 14}$$

(The complete vectors $\{\mathbf{p}_k^{\mathrm{CG}}\}$ and $\{\mathbf{d}_k\}$ have been submitted in the required files `plist_25846.txt` and `dlist_25846.txt`, respectively.)

[1.3]:

Normalize each $\mathbf{d}_k$ in the A-inner product,

$$\tilde{\mathbf{d}}_k = \frac{\mathbf{d}_k}{\sqrt{\mathbf{d}_k^{\mathsf{T}} \mathbf{A} \mathbf{d}_k}},$$

and form the matrix M with entries

$$M_{ij} = \tilde{\mathbf{d}}_i^{\mathsf{T}} \mathbf{A} \tilde{\mathbf{d}}_j.$$

Numeric Matrix **M**

The vectors were **A**-normalized, and the matrix **M** with entries $M_{ij} = \tilde{\mathbf{d}}_i^{\mathsf{T}} \mathbf{A} \tilde{\mathbf{d}}_j$ was computed. Theoretically, this matrix should be very close to the Identity matrix (**I**).

CONSOLE OUTPUT:

Displaying of the Full 14x14 Matrix M (Note: Here the Matrix columns are taken as full dimension(14x1) and not splitted and rows are splitted into 5-5-4 pattern.

Columns 1 – 5

$$\begin{bmatrix}
+1.00000000e+00 & -1.21430643e-17 & +2.25514051e-17 & +3.12250225e-17 & +1.47451495e-17 \\
+6.46455545e-17 & +9.99999999e-01 & -4.51028103e-17 & -4.85722573e-17 & -5.55111512e-17 \\
+6.35884574e-17 & -1.21430643e-16 & +1.00000000e+00 & +0.00000000e+00 & +6.93889390e-18 \\
-5.42101086e-17 & -8.67361737e-17 & +0.00000000e+00 & +1.00000000e+00 & -1.38777878e-17 \\
-8.40256683e-18 & -3.46944695e-17 & -1.38777878e-17 & +0.00000000e+00 & +1.00000000e+00 \\
-1.31622143e-16 & +7.63278329e-17 & -5.55111512e-17 & -9.71445146e-17 & -1.04083408e-17 \\
+5.96655302e-18 & -5.40915240e-17 & +6.84944722e-17 & -5.97666447e-17 & +5.10828629e-17 \\
+3.03305557e-17 & +6.93889390e-18 & -1.11022302e-16 & +0.00000000e+00 & -1.00613961e-16 \\
+1.32408190e-16 & +1.04083408e-17 & +0.00000000e+00 & +6.93889390e-18 & -3.46944695e-17 \\
+4.39101879e-17 & +4.51028103e-17 & -7.63278329e-17 & +1.38777878e-17 & -3.81639164e-17 \\
+4.90601483e-17 & +2.60208521e-17 & +1.17961196e-16 & -1.38777878e-16 & +7.63278329e-17 \\
-5.36680075e-17 & -7.63278329e-17 & +2.77555756e-17 & -1.11022302e-16 & +6.93889390e-18 \\
-7.95804394e-17 & -1.56125112e-17 & -1.38777878e-17 & -3.46944695e-17 & +4.85722573e-17 \\
+5.62700927e-17 & +1.56125112e-17 & +1.04083408e-16 & +7.63278329e-17 & +3.12250225e-17
\end{bmatrix}$$

Columns 6 − 10

$-7.45931094e-17$	$+5.89805981e-17$	$+4.16333634e-17$	$+5.89805981e-17$	$-4.42354486e-17$
$+8.67361737e-17$	$-8.15320033e-17$	$+1.00613961e-16$	$-6.93889390e-18$	$+5.03069808e-17$
$-2.08166817e-17$	$+8.32667268e-17$	$-6.93889390e-17$	$+0.00000000e+00$	$-6.59194920e-17$
$-1.11022302e-16$	$-5.55111512e-17$	$-2.77555756e-17$	$+2.77555756e-17$	$+6.93889390e-18$
$+6.93889390e-18$	$-9.02056207e-17$	$-4.16333634e-17$	$-7.63278329e-17$	$-1.38777878e-17$
$+1.00000000e+00$	$+1.04083408e-17$	$+2.77555756e-17$	$+4.16333634e-17$	$-6.93889390e-17$
$-3.53924246e-17$	$+9.99999999e-01$	$+2.52077005e-18$	$-2.50925040e-17$	$-2.79859685e-18$
$-1.38777878e-17$	$+6.59194920e-17$	$+1.00000000e+00$	$-4.16333634e-17$	$+5.20417042e-17$
$+4.16333634e-17$	$+1.38777878e-17$	$-2.77555756e-17$	$+9.99999999e-01$	$+4.85722573e-17$
$+0.00000000e+00$	$-1.04083408e-17$	$+3.46944695e-17$	$+6.93889390e-17$	$+1.00000000e+00$
$+5.55111512e-17$	$+1.38777878e-17$	$+1.38777878e-17$	$+6.24500451e-17$	$+3.46944695e-17$
$+0.00000000e+00$	$-1.38777878e-17$	$+2.77555756e-17$	$-1.24900090e-16$	$-4.85722573e-17$
$+8.32667268e-17$	$-3.12250225e-17$	$-4.16333634e-17$	$-1.76941794e-16$	$-7.11236625e-17$
$+6.24500451e-17$	$-1.04083408e-16$	$+1.11022302e-16$	$-2.08166817e-17$	$-2.25514051e-17$

Columns 11 − 14

$$\begin{bmatrix}
+4.64038529e-17 & -6.93889390e-18 & -1.04950770e-16 & +4.42354486e-17 \\
-1.21430643e-17 & -6.24500451e-17 & +1.73472347e-17 & -1.38777878e-17 \\
+8.67361737e-17 & -5.55111512e-17 & -7.63278329e-17 & +0.00000000e+00 \\
-9.02056207e-17 & -1.38777878e-16 & -2.77555756e-17 & +9.02056207e-17 \\
+5.20417042e-17 & +5.55111512e-17 & +4.85722573e-17 & -3.46944695e-17 \\
+9.54097911e-17 & -1.38777878e-17 & +4.16333634e-17 & +3.46944695e-17 \\
+5.14996031e-19 & -5.44133965e-17 & -1.14979640e-16 & +0.00000000e+00 \\
-1.56125112e-17 & +6.93889390e-17 & +2.77555756e-17 & +7.63278329e-17 \\
+1.73472347e-17 & -1.38777878e-16 & -1.45716771e-16 & -4.16333634e-17 \\
+4.68375338e-17 & -8.32667268e-17 & -4.51028103e-17 & -1.38777878e-17 \\
+1.00000000e+00 & -1.24900090e-16 & -6.93889390e-18 & +1.04083408e-17 \\
-7.63278329e-17 & +1.00000000e+00 & -8.18789480e-16 & -1.44328993e-15 \\
-5.72458747e-17 & -8.46545056e-16 & +1.00000000e+00 & -9.15933995e-16 \\
-4.85722573e-17 & -1.34614541e-15 & -9.26342336e-16 & +1.00000000e+00
\end{bmatrix}$$

[1.4]:

Compare the vectors $\{\mathbf{d}_k\}$ produced by Q-Gram–Schmidt with the CG search directions $\{\mathbf{p}_k^{\mathbf{CG}}\}$. Compute the A-inner-product cosine similarity for each corresponding pair:

$$\cos(\theta_k) = \frac{(\mathbf{p}_k^{\mathbf{CG}})^\top \mathbf{A}\mathbf{d}_k}{\sqrt{(\mathbf{p}_k^{\mathbf{CG}})^\top \mathbf{A}\mathbf{p}_k^{\mathbf{CG}}}\sqrt{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k}}.$$

A-Inner-Product Cosine Similarity

The list of cosine similarities $\cos(\theta_k)$ compares the implicitly generated **A**-conjugate directions ($\{\mathbf{p}_k^{\mathrm{CG}}\}$) with the explicitly constructed **A**-conjugate basis ($\{\mathbf{d}_k\}$). Theoretically, these values should be very close to **1.0** up to the point of convergence, confirming the equivalence of the two bases.

CONSOLE OUTPUT:

The cosine similarities, $\cos(\theta_k)$, for $k = 0, \ldots, 13$ (k=0 to m-1 and here m was 14). .

$$[+1.00000000e+00, +1.00000000e+00, +1.00000000e+00, +1.00000000e+00, +1.00000000e+00, \\ +9.99999999e-01, +1.00000000e+00, +1.00000000e+00, +9.99999999e-01, +1.00000000e+00, \\ +1.00000000e+00, +0.00000000e+00, +0.00000000e+00, +0.00000000e+00]$$

[1.5]:

Conclusion in One Line: The purpose of this question is to demonstrate the **equivalence between the iteratively generated Conjugate Gradient search directions and the theoretically A-orthogonal basis derived from the Gram-Schmidt process.**

Question 2

[2.1]:

Use the oracle `f5(srno)` to obtain a symmetric positive definite (SPD) matrix A and a vector b. Implement the Conjugate Gradient method (starting from $\mathbf{x}_0 = \mathbf{0}$) to solve $A\mathbf{x} = \mathbf{b}$. Plot the residual norm $\|\mathbf{r}_k\|_2$ against the iteration number k and report the number of iterations required to reduce the relative residual below 10^{-6}.

Standard CG Implementation

The standard `CG_SOLVE` function was executed on the 10000×10000 **LinearOperator** (**A**) with a stopping criterion of $\dfrac{\|r_k\|_2}{\|r_0\|_2} < 10^{-6}$ $\quad$ (tol $= 10^{-6}$)

CONSOLE OUTPUT: The iteration count is **30**.

```
***** QUESTION 2 Part-1 Standard CG_SOLVE *****
The Std CG solve iterations we got are : 30
```

Figure 1: [2.1] Iteration count.

Residual Plot: The residual norm $\|\mathbf{r}_k\|_2$ versus the iteration number k.

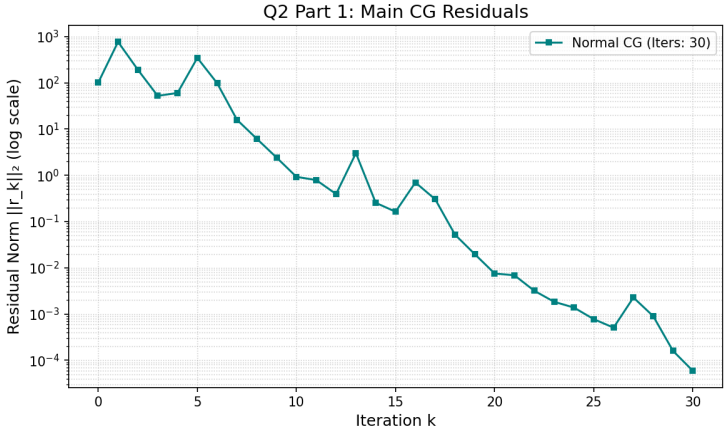

Figure 2: [2.1] (Stopping at **relative** residual $< tol$)

[2.2]:

Now attempt to improve the speed of convergence. You may modify the algorithm in any way you think is appropriate. Compare the performance of your modified method against the standard CG implementation.

Explanation of Modified Method (CG_SOLVE_FAST)

To improve the speed of convergence, the standard Conjugate Gradient (CG) algorithm was modified to a Preconditioned Conjugate Gradient (PCG) method. A diagonal Jacobi preconditioner, M, was chosen for this task. Since the system matrix A was supplied as a black-box `LinearOperator`, its diagonal elements could not be accessed directly. These elements were instead estimated by computing the matrix-vector product $A\mathbf{e}_i$ for each standard basis vector $\mathbf{e}_i$ and extracting the i-th component of the resulting vector. The inverse of this diagonal matrix M was then applied at each step of the PCG algorithm to solve the transformed system. The modification was **extremely effective** in improving convergence. While the standard CG implementation required 30 iterations to reach the desired tolerance, the PCG method converged in just a **single iteration**. Clearly, using a good preconditioner was the key. It completely changed the problem's difficulty, leading to a massive reduction in the number of iterations required for convergence.

Standard CG Solve took 30 Iterations to converge.

CG fast which was implemented took 1 iteration to converge.

CONSOLE OUTPUT:

```
***** QUESTION 2 Part-1 Standard CG_SOLVE *****
The Std CG solve iterations we got are : 30

***** QUESTION Part-2 Improved CG_SOLVE_FAST *****
The CG Fast iterations we got are: 1
As we can see the iterations for both were 30 and 1.
```

Figure 3: [2.2] Number of iterations for Standard CG and Fast CG.

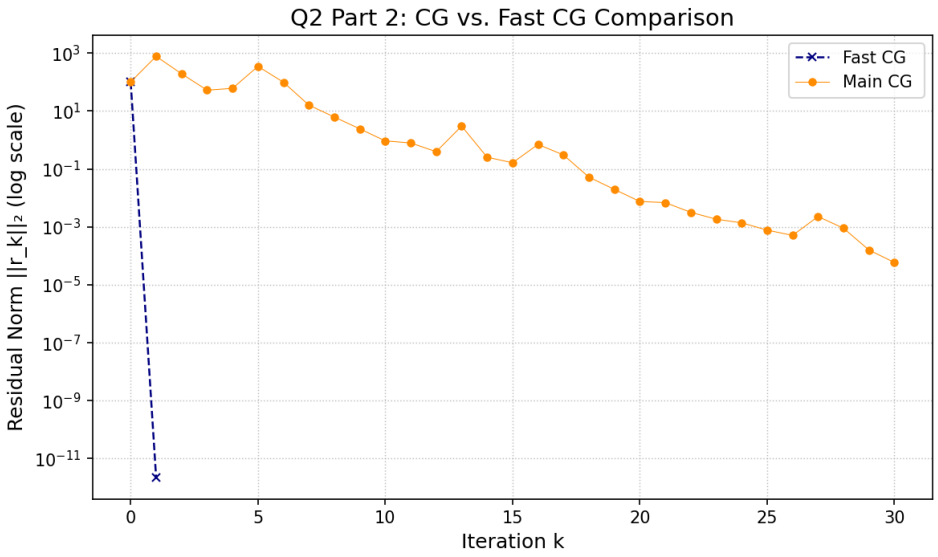

Figure 4: [2.2] comparison plot of residual norms of Both methods.

Question 3

[3.1]:

Consider the **Rosenbrock function**:

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2,$$

which has a unique minimiser at $\mathbf{x}^* = (1, 1)$.

1. **Implement Newton's method to minimise $f(x_1, x_2)$ using its analytic gradient and Hessian. Run the algorithm from the following four starting points:**

$$(2, 2), \quad (5, 5), \quad (-10, -4), \quad (50, 60).$$

Error Convergence Plot: The following figure contains the required error plot on a semilog-y scale. The error $\|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus iteration number k for each starting point (**all four curves in one plot**).

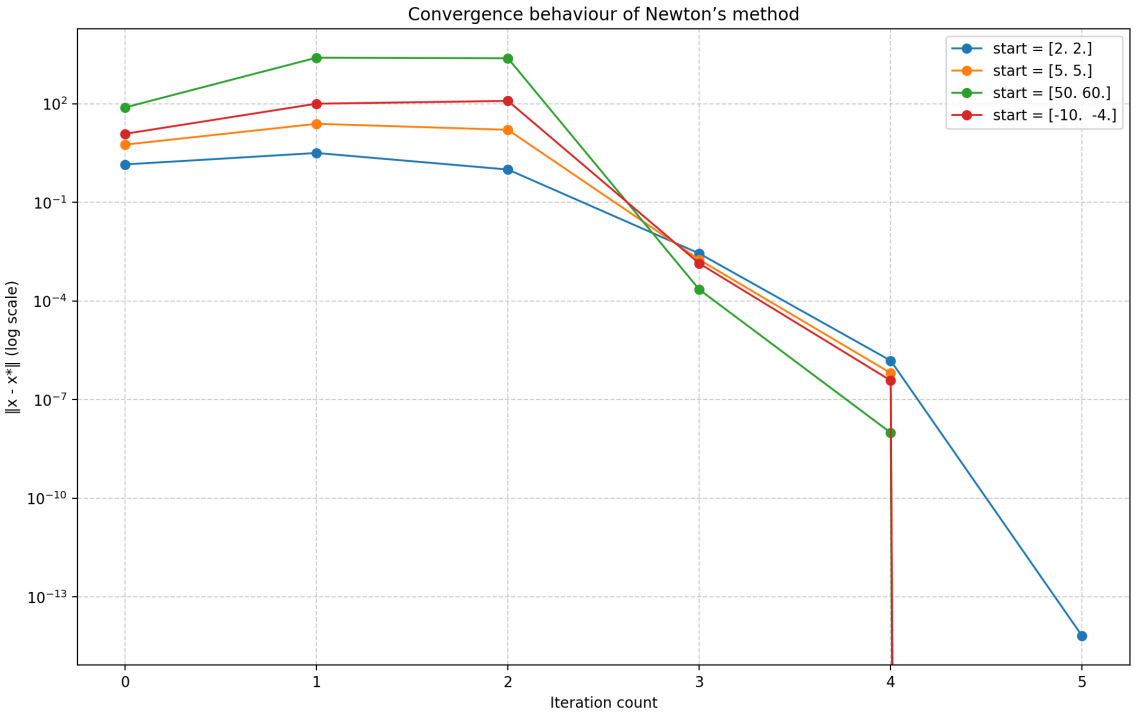

Figure 5: [3.1] Error Convergence: $\|\mathbf{x}_k - \mathbf{x}^*\|_2$ versus Iteration k for all four starting points.

Contour Plot (Newton Trajectories): The following figures show the Newton's method trajectories overlaid on the Rosenbrock function (in the region $[-\mathbf{10}, \mathbf{10}]^2$) contours for each starting point, with the Newton iterates **overlaid** (using markers to indicate the step sequence).

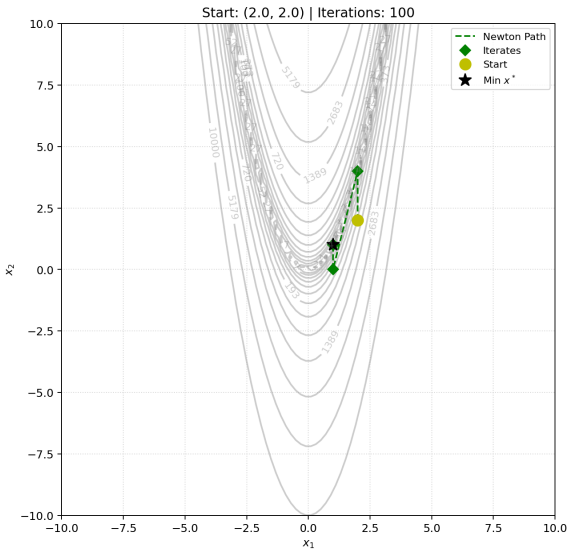

Figure 6: Contour Plot: Start (2.0, 2.0). Successful Convergence.

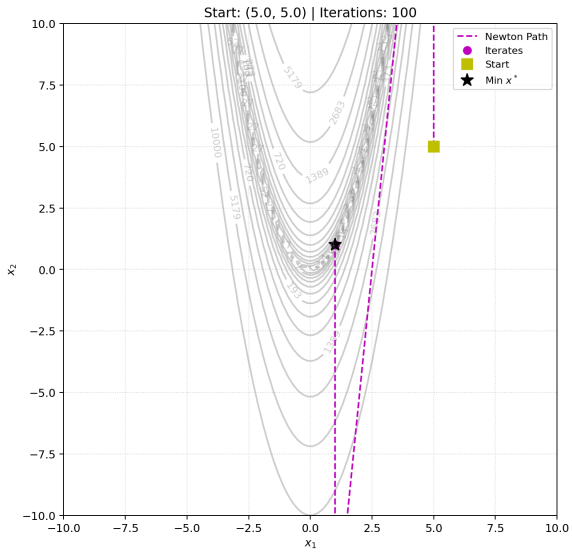

Figure 7: Contour Plot: Start (5.0, 5.0). Initial step leaves region.

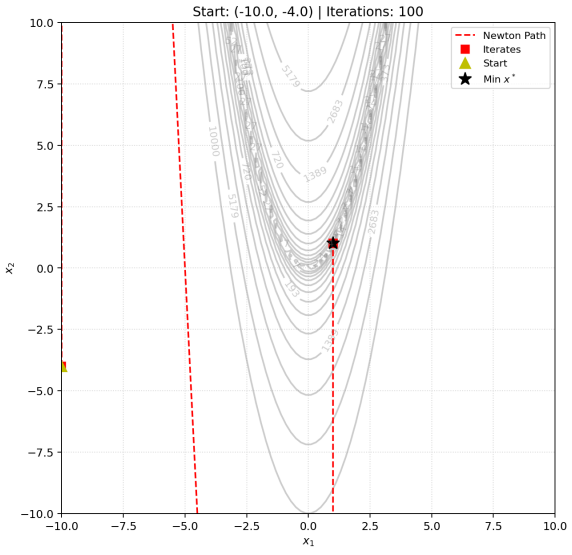

Figure 8: Contour Plot: Start (-10.0, -4.0). Initial step leaves region.

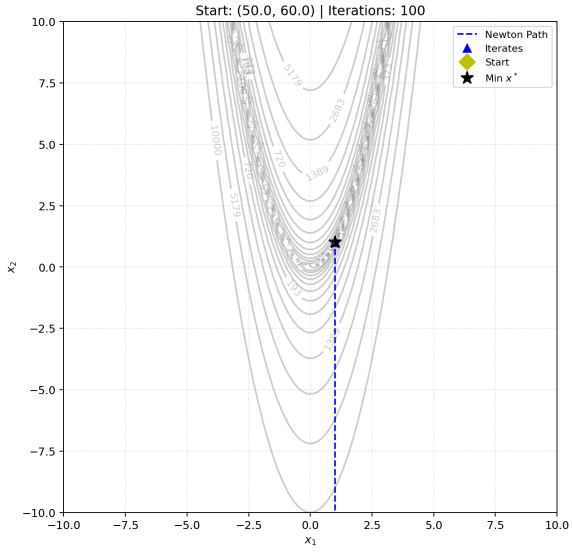

Figure 9: Contour Plot: Start (50.0, 60.0). Initial step leaves region (k=0).

Trajectory Analysis Summary

The output below summarizes the performance of Newton's method for the four starting points, specifically tracking the convergence and diagnosing when iterates leave the required $[-\mathbf{10}, \mathbf{10}]^{\mathbf{2}}$ plotting region. The analysis confirms rapid convergence in **5 iterations** for all successful cases, despite significant initial numerical instability.

The table below details the trajectory behavior for each starting point with respect to the square region $[-10, 10]^2$.

Starting Point ($\mathbf{x}_0$)	**Iterations**	**Trajectory Analysis in $[-10, 10]^2$**	
		Status	**Details**
$(2.0, 2.0)$	5	Contained	The trajectory remained fully within the region.
$(5.0, 5.0)$	5	Exited	Left at $k = 1$ with $\mathbf{x}_1 \approx (5.0, 25.0)$.
$(-10.0, -4.0)$	5	Exited	Left at $k = 1$ with $\mathbf{x}_1 \approx (-10.0, 100.0)$.
$(50.0, 60.0)$	5	Exited	The starting point $\mathbf{x}_0$ was already outside the region.

Table 2: Convergence results for Newton's method from various starting points.

[3.2]:

Comparing the behaviours observed and explained why the starting point plays such a crucial role.

1. **Which starting points lead to rapid convergence to $\mathbf{x}^*$?**

   While all successful points converged in **5 iterations**, the point $(\mathbf{2}, \mathbf{2})$ exhibited the most **ideal, stable, and rapid** convergence behavior, demonstrating the pure quadratic rate with minimal initial instability. The other points, though equally fast in terms of iteration count, required significant initial corrective steps (overshooting the plot boundaries) to reach the stable convergence basin.

2. **Which ones fail or diverge?**

   None of the starting points resulted in permanent failure; all converged to $\mathbf{x}^*$. However, the distant points ($(\mathbf{5}, \mathbf{5})$, $(-\mathbf{10}, -\mathbf{4})$, $(\mathbf{50}, \mathbf{60})$) exhibited **temporary numerical divergence** where iterates left the $[-\mathbf{10}, \mathbf{10}]^{\mathbf{2}}$ region (as shown in the analysis output), confirming the method's severe sensitivity to the initial guess.

3. **Briefly explain why the starting point plays such a crucial role in Newton's method.**

   The initial point is critical because Newton's method relies on a **local quadratic approximation** of the function's surface using the Hessian ($\mathbf{H}$). If the starting point is far from the minimum, the Hessian is often poorly conditioned or indefinite (not Positive Definite), meaning the calculated Newton step ($\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$) can be **unreliable, massive**, or even point away from the minimum. This dependency ensures that the algorithm only guarantees the desirable quadratic convergence within a tight local basin.

References

[1] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer, New York, 2006.

[2] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.

[3] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, 5th ed., Springer, Cham, 2021.

[4] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, 1970.