YZV202E Optimization for Data Science
Homework II

Student: İbrahim BANCAR, 150220313, bancar22@itu.edu.tr

Problem 1: Rotated Ellipse Fitting via Nonlinear Least Squares

(a) Jacobian derivation and first–order linearization. For a datum (x_i, y_i) and parameter vector $\mathbf{p} = \big[h,\, k,\, \alpha,\, \beta,\, \theta\big]^\top$, introduce the rotated coordinates

$$u_i = (x_i - h)\cos\theta + (y_i - k)\sin\theta, \tag{1}$$
$$v_i = -(x_i - h)\sin\theta + (y_i - k)\cos\theta, \tag{2}$$

and define the residual

$$r_i(\mathbf{p}) = \left(\frac{u_i}{\alpha}\right)^2 + \left(\frac{v_i}{\beta}\right)^2 - 1.$$

Partial derivatives. With $d_{x_i} = x_i - h$ and $d_{y_i} = y_i - k$, the intermediate derivatives are

$$\frac{\partial u_i}{\partial h} = -\cos\theta, \qquad \frac{\partial v_i}{\partial h} = \sin\theta,$$
$$\frac{\partial u_i}{\partial k} = -\sin\theta, \qquad \frac{\partial v_i}{\partial k} = -\cos\theta,$$
$$\frac{\partial u_i}{\partial \theta} = -d_{x_i}\sin\theta + d_{y_i}\cos\theta, \qquad \frac{\partial v_i}{\partial \theta} = -d_{x_i}\cos\theta - d_{y_i}\sin\theta.$$

Applying the chain rule yields

$$\boxed{\begin{aligned}
\frac{\partial r_i}{\partial h} &= 2\left(\frac{u_i}{\alpha^2}(-\cos\theta) + \frac{v_i}{\beta^2}\sin\theta\right),\\
\frac{\partial r_i}{\partial k} &= 2\left(\frac{u_i}{\alpha^2}(-\sin\theta) + \frac{v_i}{\beta^2}(-\cos\theta)\right),\\
\frac{\partial r_i}{\partial \alpha} &= -\frac{2u_i^2}{\alpha^3},\\
\frac{\partial r_i}{\partial \beta} &= -\frac{2v_i^2}{\beta^3},\\
\frac{\partial r_i}{\partial \theta} &= 2\left(\frac{u_i}{\alpha^2}\big(-d_{x_i}\sin\theta + d_{y_i}\cos\theta\big) + \frac{v_i}{\beta^2}\big(-d_{x_i}\cos\theta - d_{y_i}\sin\theta\big)\right).
\end{aligned}}$$

Jacobian matrix. Collecting the five partial derivatives for each observation produces

$$\mathbf{J}(\mathbf{p}) = \begin{bmatrix}\mathbf{J}_1\\ \vdots \\ \mathbf{J}_N\end{bmatrix} \in \mathbb{R}^{N\times 5},$$

where the i-th row is

$$\mathbf{J}_i = \begin{bmatrix} 2\left(\frac{u_i}{\alpha^2}(-\cos\theta) + \frac{v_i}{\beta^2}\sin\theta\right),\ 2\left(\frac{u_i}{\alpha^2}(-\sin\theta) + \frac{v_i}{\beta^2}(-\cos\theta)\right), \\ -\frac{2u_i^2}{\alpha^3},\ -\frac{2v_i^2}{\beta^3},\ 2\left(\frac{u_i}{\alpha^2}\big(-d_{x_i}\sin\theta + d_{y_i}\cos\theta\big) + \frac{v_i}{\beta^2}\big(-d_{x_i}\cos\theta - d_{y_i}\sin\theta\big)\right)\end{bmatrix}$$

First-order (Gauss–Newton) linearisation. For the stacked residual vector $\mathbf{r}(\mathbf{p}) = [r_1, \ldots, r_N]^\top$,

$$\mathbf{r}(\mathbf{p} + \Delta\mathbf{p}) \;\approx\; \mathbf{r}(\mathbf{p}) + \mathbf{J}(\mathbf{p})\,\Delta\mathbf{p},$$

which leads to the normal equations

$$\big(\mathbf{J}^\top\mathbf{J}\big)\Delta\mathbf{p} = -\,\mathbf{J}^\top\mathbf{r}, \qquad \big(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{I}\big)\Delta\mathbf{p} = -\,\mathbf{J}^\top\mathbf{r}.$$

(b) Implementation and comparison of nonlinear solvers. Two algorithms were employed for the least-squares minimisation:

- Gauss–Newton,
- Levenberg–Marquardt.

The following performance indicators are reported for each algorithm:

- Iteration count (or Jacobian evaluations) until convergence,
- Final sum-of-squares cost,
- Condition number of the Jacobian at convergence,
- Parameter vector at convergence $[h,\ k,\ \alpha,\ \beta,\ \theta]$.

Gauss–Newton

Iterations	17
Sum-of-squares cost	3.424281×10^1
Condition number of J	4.977251
Parameters at convergence	$[4.999315,\ 8.017751,\ 0.273092,\ 0.969594,\ -0.009107]$

Levenberg–Marquardt

Jacobian evaluations	9
Sum-of-squares cost	3.424281×10^1
Condition number of J	4.977251
Parameters at convergence	$[4.999315,\ 8.017751,\ 0.273092,\ 0.969594,\ -0.009107]$

Gauss–Newton and Levenberg–Marquardt both converged to the same solution with identical final cost and Jacobian condition number. However, Gauss–Newton required 17 iterations, while Levenberg–Marquardt completed in only 9 Jacobian evaluations. This difference likely stems from step control: Gauss–Newton uses undamped updates and relies on line search, which may reject overly aggressive steps when the curvature is high or anisotropic. In contrast, Levenberg–Marquardt applies an internal damping factor that naturally shortens steps in uncertain regions, improving robustness during early iterations. Despite similar per-iteration costs, Levenberg–Marquardt demonstrated higher efficiency in this case due to fewer rejected steps.

(c) Sensitivity to parameter initialisation. In order to assess the sensitivity of the nonlinear solver to initial conditions, three distinct starting points were tested. All cases were evaluated using the Levenberg–Marquardt algorithm with analytic Jacobian, identical termination tolerances, and a maximum of 2000 function evaluations. The outcomes are summarised below:

Initial guess: Reasonable (BBox-centre)

- Convergence status: `‘ftol‘ termination condition is satisfied.`
- Jacobian evaluations: 9
- Final loss value: 3.424281×10^{1}
- Condition number of Jacobian: 4.977×10^{0}
- Estimated parameters: $[4.999315,\ 8.017751,\ 0.273092,\ 0.969594,\ -0.009107]$

Initial guess: Too large + rotated

- Convergence status: The maximum number of function evaluations is exceeded.
- Jacobian evaluations: 397
- Final loss value: 7.580080×10^{-15}
- Condition number of Jacobian: 3.276×10^{16}
- Estimated parameters: $[5.71 \times 10^{7},\ -7.34 \times 10^{5},\ 6.34 \times 10^{5},\ 5.71 \times 10^{7},\ 1.558]$

Initial guess: Underscaled

- Convergence status: The maximum number of function evaluations is exceeded.
- Jacobian evaluations: 384
- Final loss value: 9.109905×10^{-15}
- Condition number of Jacobian: 1.871×10^{16}
- Estimated parameters: $[-5.14 \times 10^{7},\ 1.29 \times 10^{6},\ 5.14 \times 10^{7},\ 1.73 \times 10^{4},\ -939.36]$

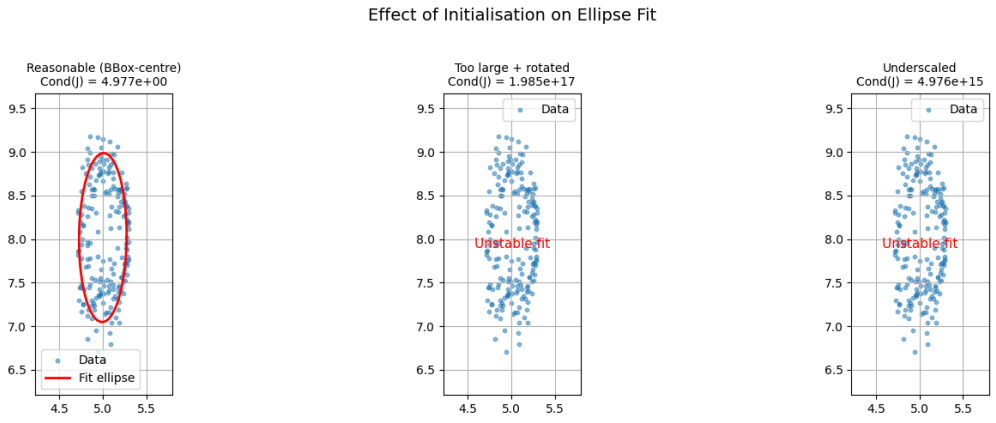

Figure 1: The effect of different initial parameter values on the ellipse fitting process. Only the well-scaled initialisation results in a geometrically accurate and numerically stable fit. Oversized or underscaled axes lead to extreme parameter estimates and ill-conditioning.

Comparision Only the reasonable initial guess led to successful and well-conditioned convergence. The other two initializations, despite achieving near-zero residual norms, resulted in parameter estimates of extremely large magnitude and were terminated due to the function evaluation limit. Their final Jacobian matrices exhibited severe ill-conditioning (condition numbers exceeding 10^{16}), indicating numerical instability. This suggests that while the residual may be minimized, the estimated parameters can become unreliable when starting far from the true solution, highlighting the importance of informed initialization.

(d) Impact of parameter initialization. Among the five parameters (h, k, α, β, θ), the experiments indicate that the initialization of the scale parameters α and β has the most significant impact on convergence.

In particular, the test cases "Too large + rotated" and "Underscaled" both introduced large distortions to α and β relative to the true values. Despite achieving extremely low residuals (loss values $< 10^{-14}$), both runs failed to converge properly and terminated due to exceeding the maximum number of function evaluations. More critically, the resulting parameter vectors contained values on the order of 10^7, and the condition numbers of the Jacobian matrices exceeded 10^{16}, indicating numerical instability.

By contrast, variations in the center coordinates (h, k) and rotation angle θ had relatively minor influence. The "Reasonable" initial guess—which had slightly inaccurate h and k, and a neutral $\theta = 0$—converged in 9 Jacobian evaluations, with a final loss and condition number essentially identical to the best-fit solution.

These results suggest that the nonlinear system is particularly sensitive to the initial scale of the ellipse axes. Since the residuals involve division by α^2 and β^2, poor initial values can distort the gradient and lead to highly ill-conditioned Jacobians. Properly scaling α and β based on the data spread is therefore essential to ensure stable convergence.

Problem 2: Dolphin Function Optimization Study

(a) Function Visualization The multimodal objective function :

$$f(x, y) = -\left|\sin(x-3)\cos(y-1)\right| - 4\exp\left(-\left[0.5(x-3)^2 + 0.5(y-1)^2\right]\right)$$

The function is visualized over the domain $x, y \in [-5, 10]$ using both a 3D surface plot and a 2D contour plot to illustrate the underlying structure of the objective landscape.

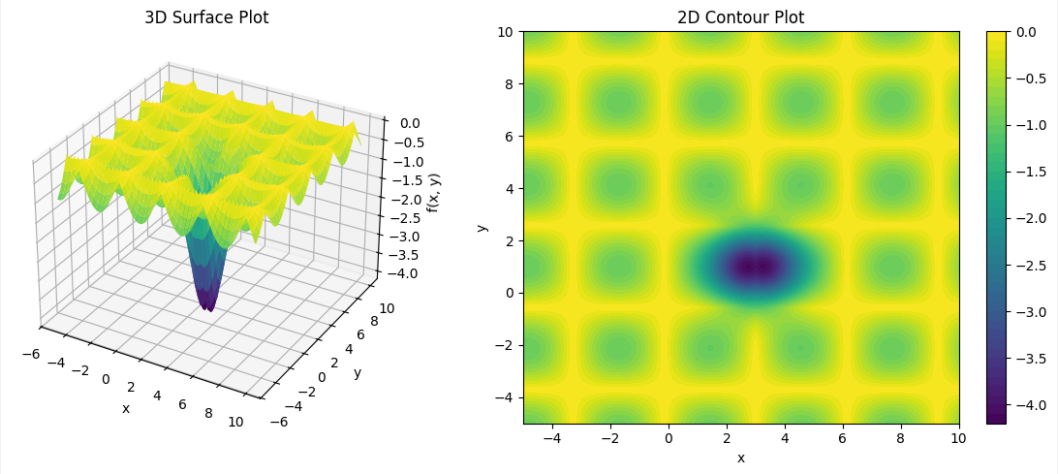

Figure 2: Combined 3D surface and 2D contour plot of the Dolphin function

(b) Performance Evaluation over Multiple Trials The Particle Swarm Optimization (PSO) method was executed over 25 independent trials to assess the stability of the optimization algorithm . In each trial, different random seeds were used while optimizing the Dolphin function defined as:

$$f(x, y) = -\left|\sin(x-3)\cos(y-1)\right| - 4\exp\left(-\left[0.5(x-3)^2 + 0.5(y-1)^2\right]\right)$$

The optimization was conducted within the domain $x, y \in [-5, 10]$. Each PSO run used 30 particles and was allowed to iterate for 100 steps (corresponding to 3000 function evaluations per trial). After each run, the best fitness value obtained was recorded below.
The following table summarizes the statistical results across the 25 runs:

Metric	Value
Mean fitness value	-4.12433691
Standard deviation	8.88×10^{-16}
Success rate ($f(x, y) \leq -3.999$)	100%

Table 1: Performance of PSO over 25 independent trials on the Dolphin function

The results demonstrate that the algorithm consistently converged to the global minimum across all runs. The near-zero standard deviation indicates extremely stable behavior regardless of initialization. The high success rate confirms the effectiveness of PSO in locating the optimal solution in the absence of noise or domain constraints.

(c) Sensitivity to Parameter Initialization 25 independent trials again were conducted for each of three different initial conditions in order to analyze the sensitivity of the Particle Swarm Optimization (PSO) algorithm to parameter initialization . In each trial, the search domain was restricted to a randomly sampled rectangular subregion of the full domain $x, y \in [-10, 10]$.
The following statistical results summarize the mean fitness, standard deviation, and success rate (defined as reaching $f(x, y) \leq -3.999$) across 25 trials under constrained subregions:

Metric	Value (restricted domain)
Mean fitness value	-1.62898564
Standard deviation	1.22
Success rate	16%

Table 2: Performance of PSO over constrained subregions of the domain

Compared to the unconstrained case where the success rate was 100%, restricting the domain introduces a higher chance of convergence to suboptimal local minima. The variance in final fitness values also increased significantly, indicating that the algorithm's performance deteriorates under limited exploration space.

(d) Robustness under Gaussian Noise To evaluate the robustness of the PSO algorithm under noisy conditions, Gaussian noise was added to the objective function evaluations. The function range was estimated from the global minimum and the observed maxima within the domain $x, y \in [-5, 10]$.
The algorithm was then re-executed for 25 independent trials under these noisy conditions. The following table reports the statistical performance:

Metric	Value (with noise)
Mean fitness value	-4.26286867
Standard deviation	0.0175
Success rate	100%

Table 3: Performance of PSO under Gaussian noise $\mathcal{N}(0, \sigma^2)$

Despite the perturbation introduced by Gaussian noise, the algorithm maintained a 100% success rate, consistently converging to a solution close to the true optimum. The standard deviation increased compared to the noiseless case, as expected, but the mean fitness remained very close to the global minimum. These results indicate that the PSO algorithm is robust against moderate levels of stochastic noise in the objective function evaluations.

(e) Performance under Budget Constraints The PSO algorithm was executed under three different budget constraints: 750, 1000, and 1500 function evaluations per run. These correspond to approximately 25, 34, and 50 iterations, respectively, using 30 particles per iteration.
The following table summarizes the performance metrics for each case, including the mean fitness value, standard deviation, and success rate over 25 trials:

Budget	Mean fitness	Std. dev.	Success rate
750 evals (25 iters)	-4.12433146	1.22×10^{-5}	100%
1000 evals (34 iters)	-4.12433687	8.15×10^{-8}	100%
1500 evals (50 iters)	-4.12433691	6.73×10^{-12}	100%

Table 4: Performance of PSO under varying function evaluation budgets

The results show that PSO consistently converges to the global optimum even under a restricted evaluation budget. However, increasing the budget slightly improves the precision of the final solution, as seen in the decreasing standard deviation.

(f) Which algorithm was more successful under stochastic and noisy conditions? Under noisy conditions, the Particle Swarm Optimization (PSO) algorithm demonstrated consistent and robust performance. Despite the addition of Gaussian noise sampled from $\mathcal{N}(0, \sigma^2)$, with $\sigma = 0.01 \times \text{range}(f)$, PSO achieved a 100% success rate over 25 trials and maintained a low standard deviation (≈ 0.0175). This indicates that PSO is resilient to moderate levels of stochasticity in the objective function. The inherent population-based and exploratory nature of PSO allows it to average out noise effects over particles, preventing premature convergence due to local perturbations.

(g) Which function posed greater difficulty for the optimization process, and why? Among the tested objective functions, the Dolphin function posed greater difficulty for the optimization process. Although the global minimum is sharp and well-defined, the presence of misleading local basins increases the likelihood of premature convergence when initialization is suboptimal or domain constraints are imposed. Experimental results in part (c) demonstrated that restricting the domain led to a significant drop in success rate (from 100% to 16%) and increased variance, which highlights the sensitivity of the algorithm to the function's landscape complexity.

Problem 3: Optimization on Beale Function

The non-convex Beale function considered in this section is defined as:

$$f(x_1, x_2) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

Two optimization algorithms—Fletcher–Reeves Conjugate Gradient (CG) and Newton's Method—were implemented to minimize this function. The algorithms were evaluated using three distinct initial points:

- A **near** point close to the global minimum,
- A **far** point located far from the optimum,
- A **difficult** point chosen to induce challenging convergence behavior.

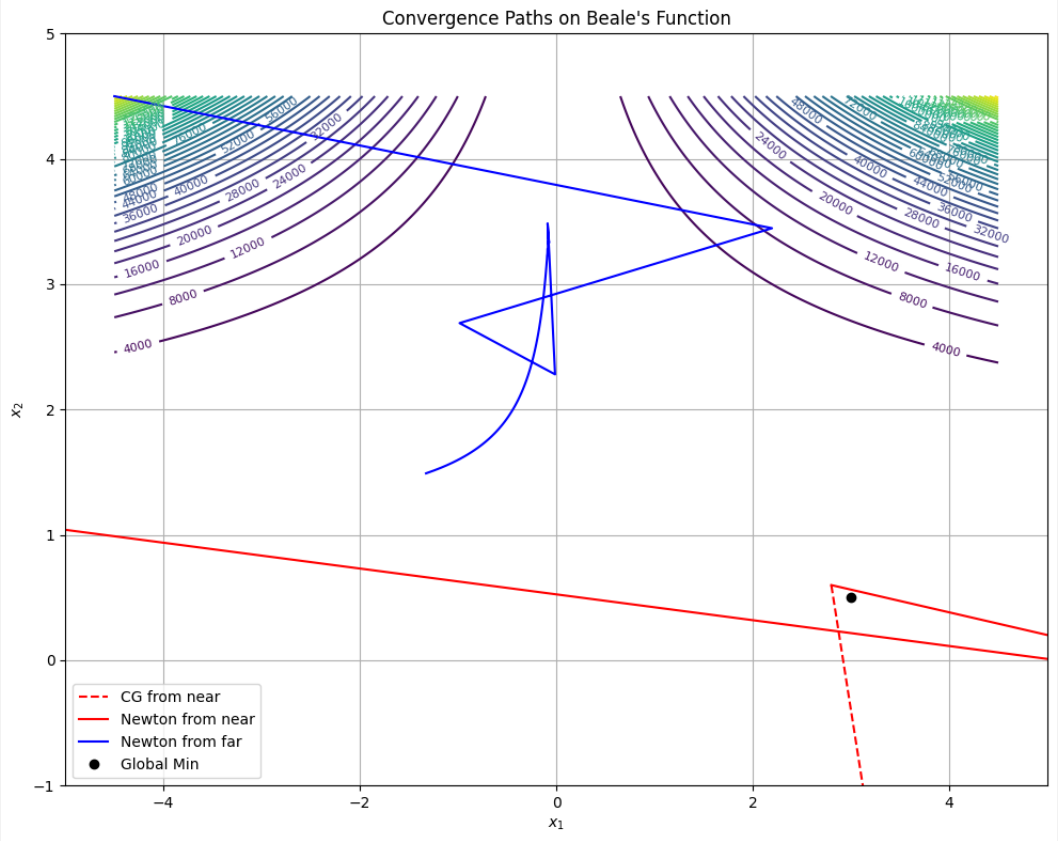

Figure 3: Convergence trajectories of Newton's Method and Conjugate Gradient on the Beale function from various initializations.

As observed in the visualization, convergence from the *near* initial point is relatively stable for both methods, with Newton's Method reaching the global minimum efficiently. In contrast, trajectories from the *far* and *difficult* points frequently diverged or exhibited unstable behavior, especially for the Conjugate Gradient method. This highlights the sensitivity of both algorithms to initialization when optimizing non-convex functions. Certain diverging paths were omitted from the plot to preserve visual clarity. To understand the behavior of difficult initializations, additional experiments were conducted by randomly sampling a dense grid of initial points across the domain $x_1 \in [-4.5, 4.5]$, $x_2 \in [-1, 4.5]$. Each point was used to initialize both Newton's Method and Conjugate Gradient, and their convergence outcomes were recorded. These results may come from non-convex structure of the Beale function
The analysis revealed that initial points located within regions exhibiting high curvature or steep nonlinearity—particularly where gradient magnitudes vary rapidly—were more likely to cause divergence or oscillations. Specifically, points near saddle-like areas or flat regions with misleading gradient directions resulted in unstable behavior for the Conjugate Gradient method, which lacks second-order curvature correction.
One heuristic for identifying challenging initializations is to examine areas where the gradient norm is small, but the Hessian has ill-conditioned eigenvalues or large condition numbers. Such regions tend to delay convergence or push the trajectory toward suboptimal paths.
Therefore, challenging initial points can often be predicted by analyzing:

- The local gradient magnitude and its alignment with the objective contour,
- The eigenvalues of the Hessian (when available) to assess curvature conditions,
- Historical divergence rates from nearby grid samples in exploratory sweeps.

These insights demonstrate the critical role of curvature information and initialization strategy, particularly in non-convex landscapes like the Beale function.

(b) Local Curvature Analysis Firstly, computation and visualization of the condition number of the Hessian matrix, $\kappa(H_f(x))$, along the Newton method trajectories from the three initial points defined in part (a).

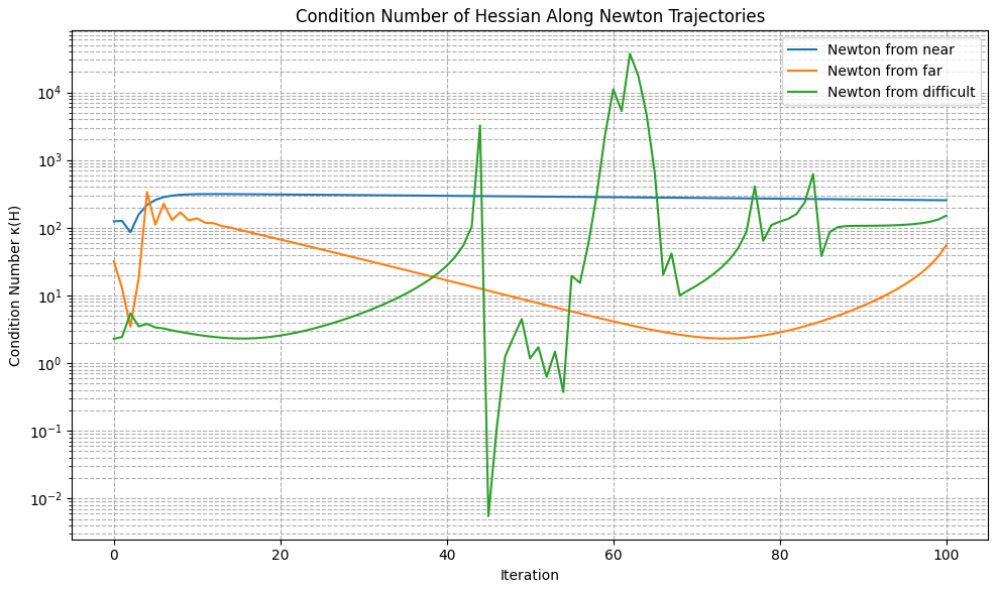

Figure 4: Condition number $\kappa(H_f(x))$ of the Hessian matrix along Newton trajectories for each initialization.

As illustrated in the figure, the condition number remains relatively stable when the algorithm is initialized from the *near* point, indicating well-conditioned curvature and contributing to the reliable convergence of Newton's method.
For the *far* initialization, the condition number starts relatively high but decreases steadily as the algorithm progresses toward better-behaved regions. This suggests that despite an initially poor curvature profile, the algorithm eventually reaches a zone with more favorable geometry.
The *difficult* point shows erratic fluctuations and extreme spikes in the condition number, exceeding 10^5 at times. These observations indicate ill-conditioned or near-singular Hessians, which lead to unreliable Newton steps and account for the algorithm's divergence or unstable oscillations in this scenario.
These findings demonstrate that Newton's method is highly sensitive to local curvature and its effectiveness depends on the numerical conditioning of the Hessian. Monitoring condition numbers offers a useful diagnostic for assessing convergence reliability.

(c) Method Recommendation for Non-Convex Optimization Based on both theoretical insights and experimental results, Newton's Method is recommended for optimizing non-convex functions like the Beale function *only* when a reliable initialization is available and the local curvature is sufficiently well-conditioned.

The experimental findings show that Newton's Method converged robustly from the *near* point, leveraging curvature information effectively due to a well-behaved Hessian. However, from the *far* and especially the *difficult* initializations, the performance deteriorated sharply due to high and unstable Hessian condition numbers. In those cases, the method either diverged or produced oscillatory behavior.

On the other hand, the Conjugate Gradient method exhibited greater robustness to initialization, although it also struggled to converge in some non-convex regions. Its reliance on first-order information only makes it more stable in poorly conditioned areas, though often at the cost of slower convergence.

Conclusion: Newton's Method should be preferred when local curvature is favorable (e.g., near a local/global optimum), whereas Conjugate Gradient is more suitable when the landscape is irregular or when Hessians are ill-conditioned. In practice, a hybrid approach or a global method (e.g., trust-region or line-search with curvature checks) may offer a better trade-off for complex non-convex landscapes, but one should consider Newton's Method is more expensive than Conjugate Gradient due to its Hessian computation.