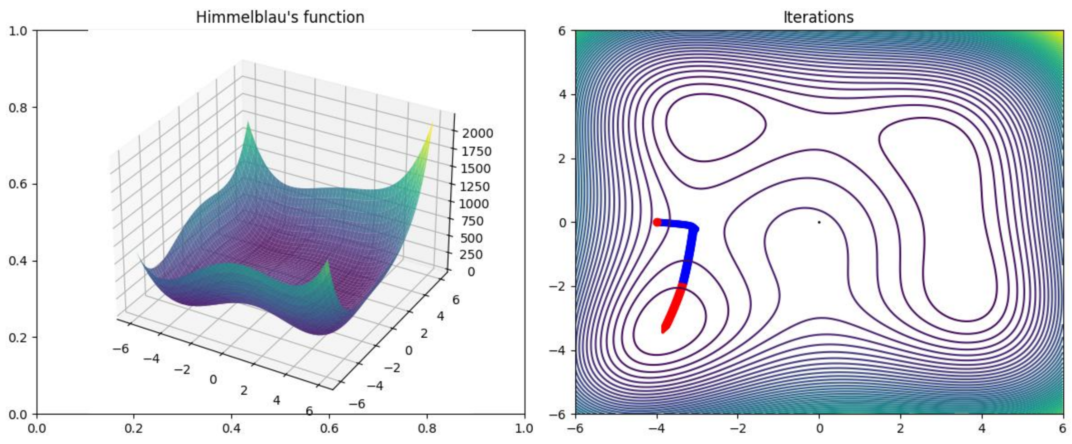
Numerical Optimization – Homework 4

Problem 1: Create a hybrid code where the steepest-descent direction is used a point where the Hessian is not positive definite and Newton direction is used at a point where Hessian is positive definite. Test your code with a function of your own choice.

Answer:

The purpose of this problem is to write a hybrid optimization algorithm that combines the Steepest Descent method and the Newton method. The idea is to use the Newton direction when the Hessian matrix is positive definite, and to use the Steepest Descent direction otherwise. This approach allows the algorithm to take advantage of Newton's fast convergence near minima but still remain stable in regions where the Hessian might lead the search in the wrong direction, such as near saddle points or flat surfaces.

1. Function, gradient, and hessian: I used Himmelblau's function, which has four minima and one saddle point.
2. Checking the Hessian: Before each step, the code checks if the Hessian matrix is positive definite by looking at its eigenvalues:
    - If all eigenvalues are positive: use Newton's direction.
    - Otherwise: use Steepest Descent direction.
3. Choosing the search direction based on the chosen method
4. Updating the position: the algorithm moves to the new point using the step length $\alpha = 0.01$



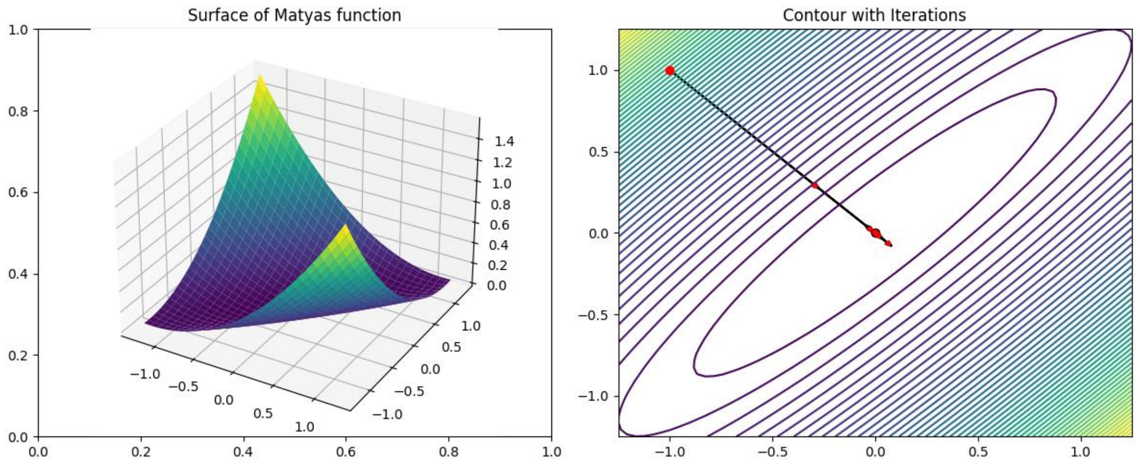
The right plot shows the trajectory of the optimization starting from the point (−4, 0). At the beginning, the Hessian is not positive definite, so the algorithm starts with Steepest Descent steps (blue arrows) to move downhill. As it reaches a smoother, convex area near one of the minima, the Hessian becomes positive definite, and the algorithm switches to Newton's method (red arrows). Overall, the algorithm successfully moves toward a nearby minimum and demonstrates how combining both methods provides stability (from Steepest Descent) and speed (from Newton's method).

Problem 2: Modify the code linesearch-steepestdescent.m so that adaptive step-length α_k is used instead of the fixed one. Design your own criteria regarding how to choose α_k at every step. Test your code with a function of your own choice.

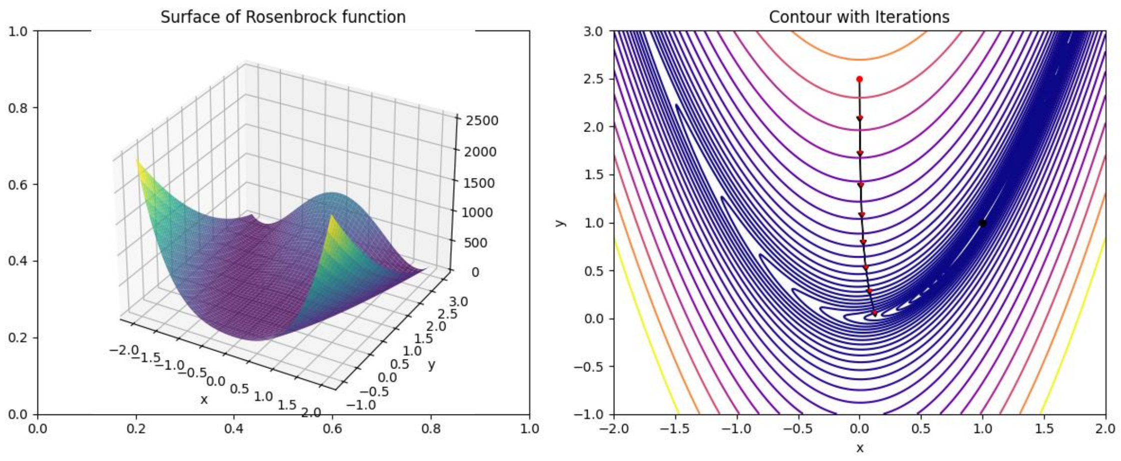
Answer:

In this problem, we should replace a constant step length with an adaptive one of our choice, testing on an arbitrary test function. I used two different algorithms for step length, one is a well-known method called Armijo backtracking line search, and the other one was totally arbitrary, in which the step length decays harmonically.

- Armijo backtracking: In this approach, I used the Armijo backtracking line search method to find an appropriate step length at each iteration for minimizing the Matyas function, which is a function with a single minimum at (0,0). Instead of using a fixed step size, the algorithm starts with an initial trial value of α and repeatedly reduces it by a constant factor until the Armijo condition is satisfied. This condition ensures that the new step gives a sufficient decrease in the function value.



- Harmonic decay: In this approach, I tested the harmonic decay step-length on the Rosenbrock function.The step length at each iteration was updated using $\alpha_k = \alpha_0/1 + \beta k$, where $\alpha_0 = 0.4$ and $\beta = 0.1$. It allows the algorithm to start with larger steps for faster progress at the beginning and then take smaller steps as it approaches the minimum. The 3D surface plot shows the Rosenbrock function, while the contour plot on the right shows the path of iterations gradually moving through the curved valley toward the true minimum. The steps are initially wide and become shorter as α_k decreases, which helps convergence.

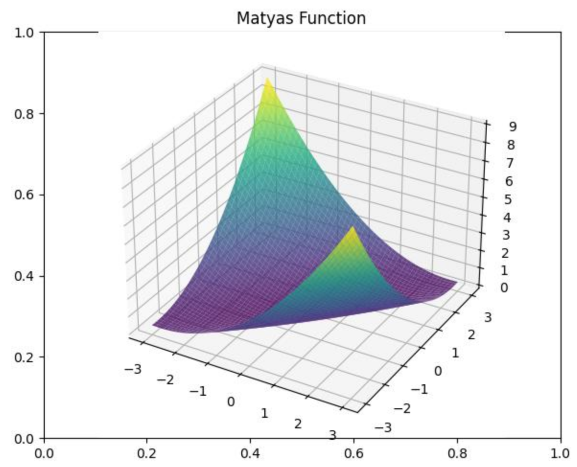


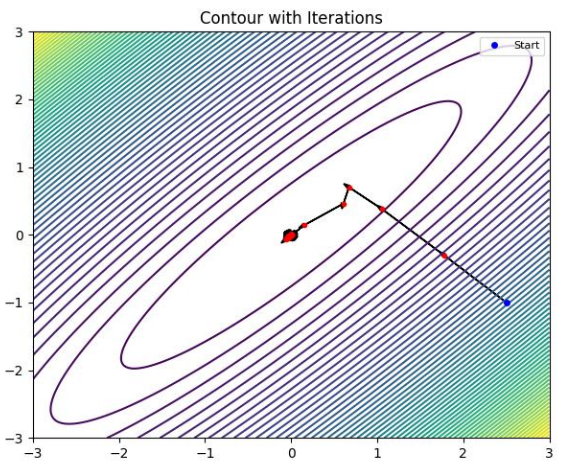
Problem 3: Implement the method on page 12 in lecture 4 for Steepest-descent method as the base method. Design your own criteria regarding how to choose c_1, c_2, c_3. Test your code with a function of your own choice. Does this method even work?

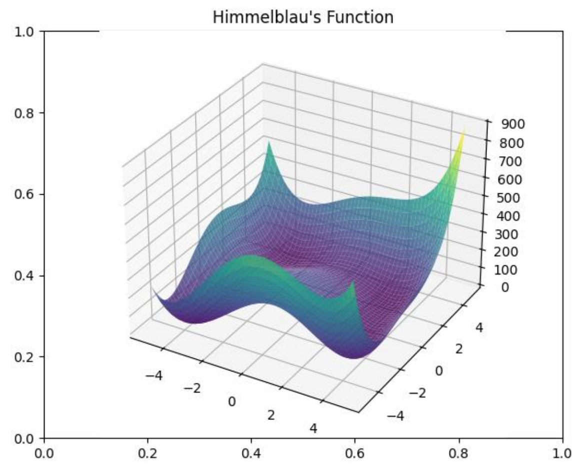
Answer:

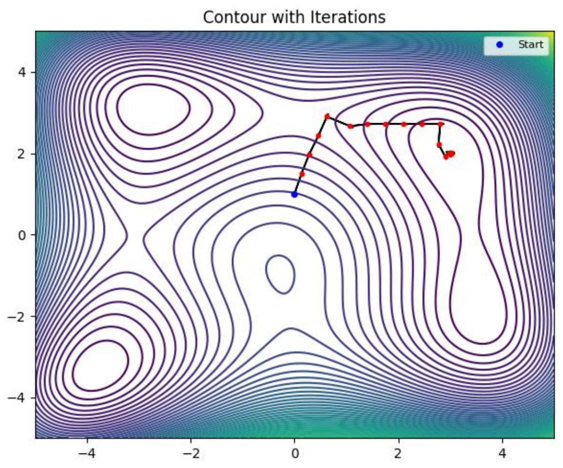
The goal of this problem was to improve the basic steepest descent method by combining the previous search directions instead of discarding them after each step. In my code, I used $c_1 = 0.6, c_2 = 0.3, c_1 = 0.1$, giving more weight to the most recent direction. For each step, the step length α_k was adaptively chosen using backtracking line search, ensuring sufficient decrease according to the Armijo condition.

For testing, I used Himmelblau's function and Matyas function. On the Himmelblau function, the hybrid direction produced smoother progress across the curved surface, and the red arrows on the contour plot show how the path avoids oscillations and converges toward a local minimum. On the Matyas function, which is simpler and convex, the algorithm quickly reached the minimum in only a few iterations, confirming that the chosen coefficients help maintain fast convergence without instability.









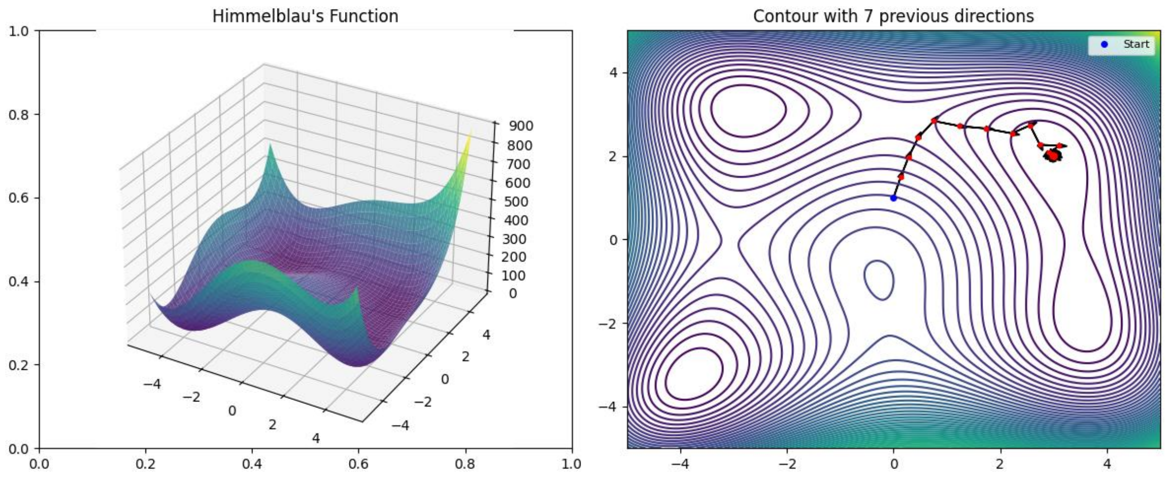
Problem 4: In Problem 3, do you see any improvement if you use 5 or 7 previous direction vectors? Test your code with a function of your own choice.

Answer:

In this problem, we need to extend the method from Problem 3 by using seven previous search directions instead of three to see whether including more direction history would improve convergence:

$$p_k = c_1 p_{k-1} + c_2 p_{k-2} + \cdots + c_7 p_{k-7}$$

where each coefficient determines how much influence each earlier step has on the new direction. The same backtracking line search was used to choose an appropriate step length at every iteration to ensure sufficient decrease in the function value.



From the results, the path on the contour plot shows a more stable and curved motion toward the local minimum compared to the case with only three directions. When using three directions, the method already improved over standard steepest descent, but there were still small zig-zags as the algorithm. With seven directions, the motion becomes noticeably smoother and more consistent, since the algorithm uses more historical information to predict an efficient descent direction.