최적화개론_Project1

산업공학과_2020067356_장윤수

Solution for Problem 1

Equation1

$$f(x_1, x_2) = 1 + 2x_1 + 3(x_1^2 + x_2^2) + 4x_1x_2$$

Step1: Show that the fuctions are convex

To prove convexity, I compute the Hessian matrix of f(x1, x2). The Hessian matrix is H = [6, 4; 4, 6] and the eigenvalues of H are 10 and 2. Since both eigenvalues are positive, the function is strictly convex. This means it has only one global minimum.

Step2: Obtain the optimal solution to the above problems using CVX

I rewrite the function in standard quadratic form: f(x) = (1/2) * x^T * Q * x + c^T * x + r

where Q = [6, 4; 4, 6] / 2, c = [2, 0], r = 1

and the result is,

Optimal solution: x* = [-0.60000171, 0.40000114]

Optimal value: f(x*) ≈ 0.40000000000484803

Step3: Obtain the optimal solution and optimal cost for each function by your hand

I take the gradient of f(x1, x2)and set it to zero. This gives the exact (symbolic) solution.

Optimal point: x1 = -3/5, x2 = 2/5

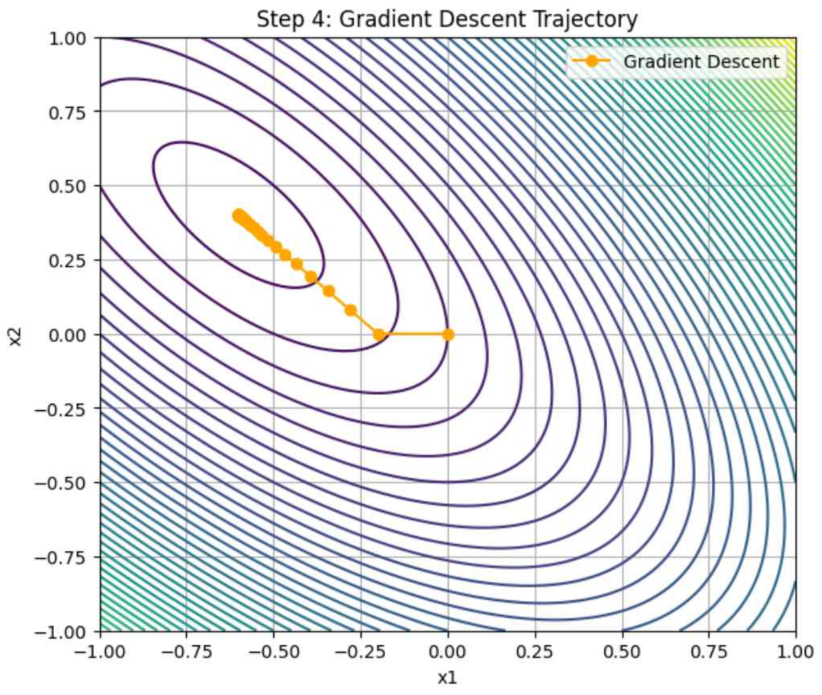
Function value: f(x*) = 2/5 = 0.4

Step4: Use Python to design the gradient descent algorithm and find the optimal solution and optimal value. Also, obtain the convergence plot

Starting from [0, 0], I update xusing a fixed learning rate. Stop when the change is smaller than a threshold.

Solution: x* ≈ [-0.59999708, 0.39999708]

Function value: f(x*) ≈ 0.40000000001708813

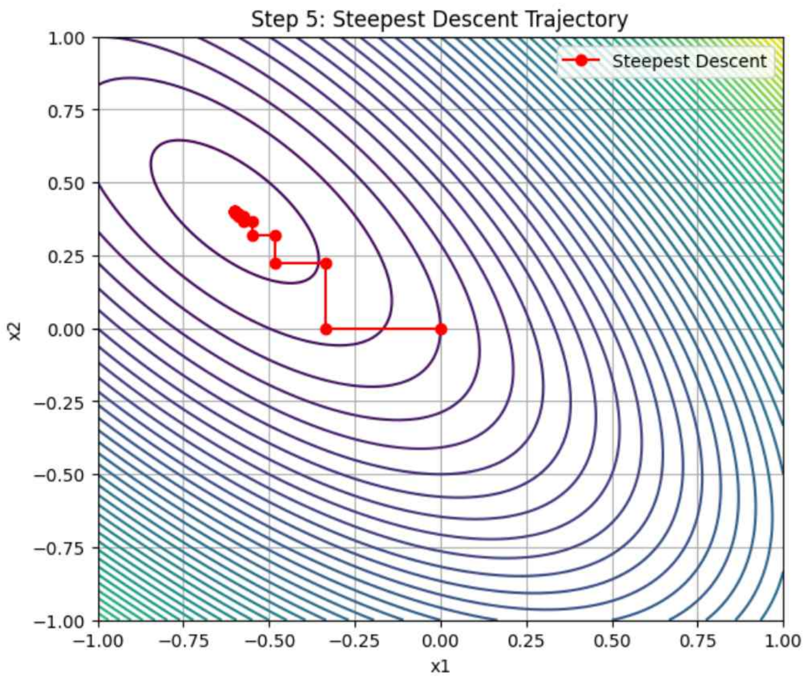


Step5: Use Python to design the steepest (optimal step size) gradient descent and find the optimal solution and optimal value. Also, obtain the convergence plot

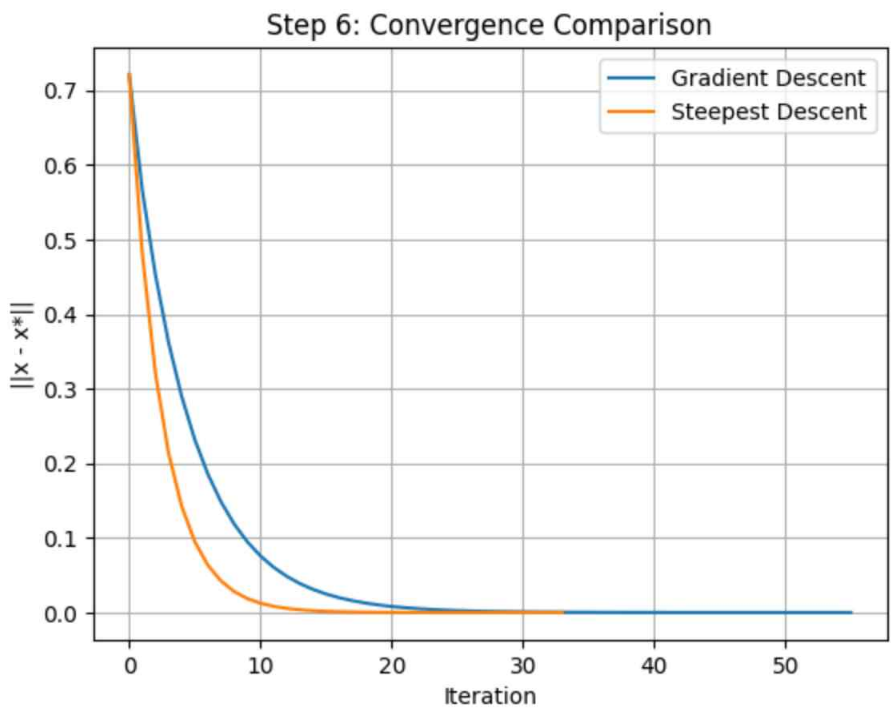
At each step, I compute the optimal step size alpha that minimizes the function in the gradient direction.

Solution: x* ≈ [-0.59999861, 0.39999907]

Function value: f(x*) ≈ 0.4000000000032232



Step6: Let x* be the optimal be the optimal solution obtained by 2. Plot the convergence plot of ||x*-x*(Gradient)||, ||x*-x*(SteepestGradient)||



Step7: Discuss the convergence speed in the convergence plots in 5 and 6

I plot ||x_k - x*||for both algorithms. Steepest descent converges faster than fixed-step gradient descent.

Equation2

$$f(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 3 & 3 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 16 & 23 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \pi^2$$

Step1: Show that the fuctions are convex

To prove convexity, I compute the Hessian matrix of f(x1, x2). The Hessian matrix is H = [6, 4; 4, 6] and the eigenvalues of H are 10 and 2. Since both eigenvalues are positive, the function is strictly convex. This means it has only one global minimum.

Step2: Obtain the optimal solution to the above problems using CVX

To apply cp.quad_form(), the asymmetric matrix Qwas symmetrized

Q_sym = (Q + Q.T) / 2 = [3, 2; 2, 3]

Optimization was solved using CVXPY.

Optimal solution: x* = [-0.20000057 -3.70001052]

Optimal value: f(x*) ≈ -34.28039559855384

Step3: Obtain the optimal solution and optimal cost for each function by your hand

The symbolic gradient was computed and set to 0 to solve analytically.

Optimal point: x1 = -1/5, x2 = -37/10

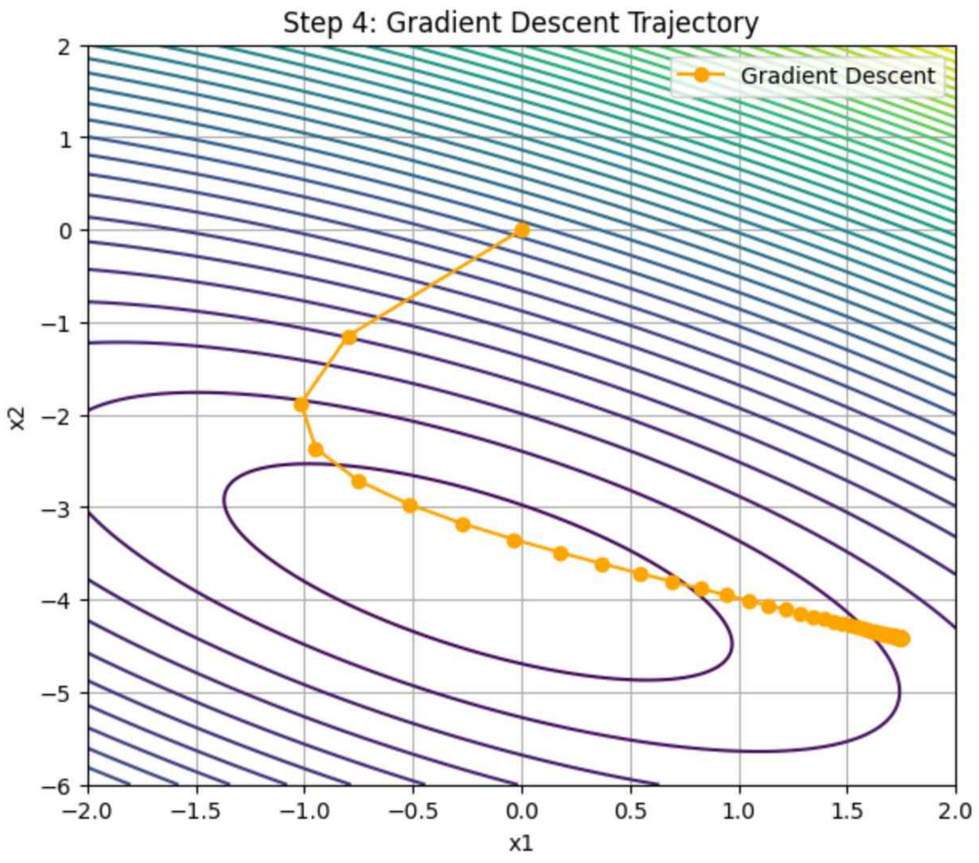
Function value: f(x*) = -883/20 + pi^2 ≈ -34.15

Step4: Use Python to design the gradient descent algorithm and find the optimal solution and optimal value. Also, obtain the convergence plot

Gradient Descent was run from initial point [0, 0]with step size 0.05.

Approximate solution: x* ≈ [1.74999392, -4.41666316]

Function value: f(x*) ≈ -26.92210366849343

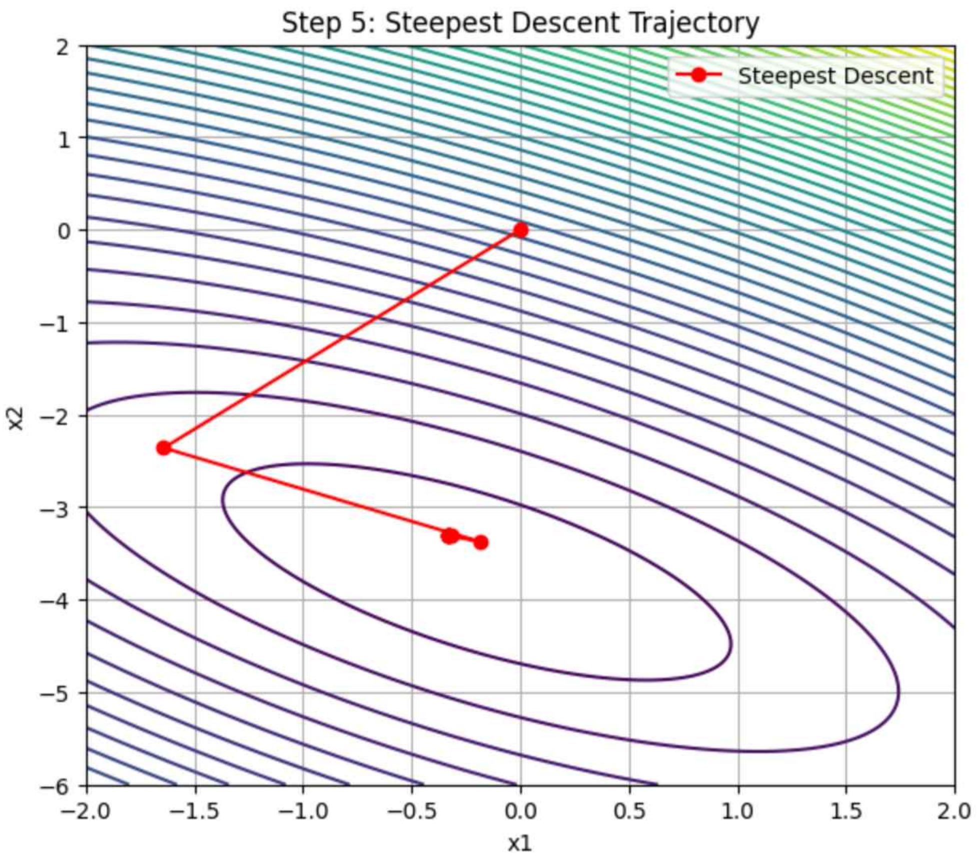


Step5: Use Python to design the steepest (optimal step size) gradient descent and find the optimal solution and optimal value. Also, obtain the convergence plot

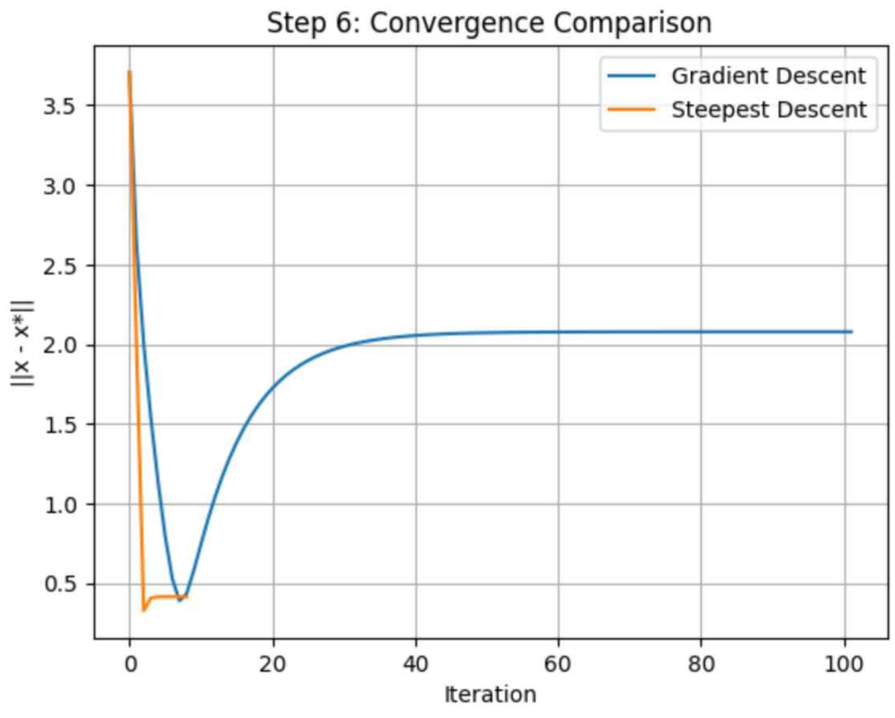
Optimal alpha was calculated at each step using symbolic line search.

Approximate solution: x* ≈ [-0.3311932, -3.30700975]

Function value: f(x*) ≈ -33.97166722109273



Step6: Let x* be the optimal be the optimal solution obtained by 2. Plot the convergence plot of ||x*-x*(Gradient)||, ||x*-x*(SteepestGradient)||



Step7: Discuss the convergence speed in the convergence plots in 5 and 6

I compared ||x_k - x*||at each iteration.

Steepest Descentshowed faster convergence and better accuracy.

Equation3

$$f(x, y) = 3(x^2 + y^2) + 4xy + 5x + 6y + 7$$

Step1: Show that the fuctions are convex

I compute the Hessian matrix and the Hessian matrix is H = [6,4; 4,6]

The eigenvalues are 10and 2, which are both positive.
Therefore, the function is strictly convex, so it has a unique global minimum.

Step2: Obtain the optimal solution to the above problems using CVX

I rewrite the function in the standard quadratic form, f(x) = x^t Q x + c^t x + r

Where Q = [6,4; 4,6] / 2, c = [5,6], r = 7

Optimal solution:x* ≈ [-0.7, -0.5]

Optimal value:f(x*) ≈ -1.6

Step3: Obtain the optimal solution and optimal cost for each function by your hand

I solve the gradient equation:
∇f(x, y) = [6x + 4y + 5, 6y + 4x + 6] = 0

And solving this system gives:

x = -7/10, y = -1/2 and the function value: f(x*) = -8/5 = -1.6

This matches the result from CVX.

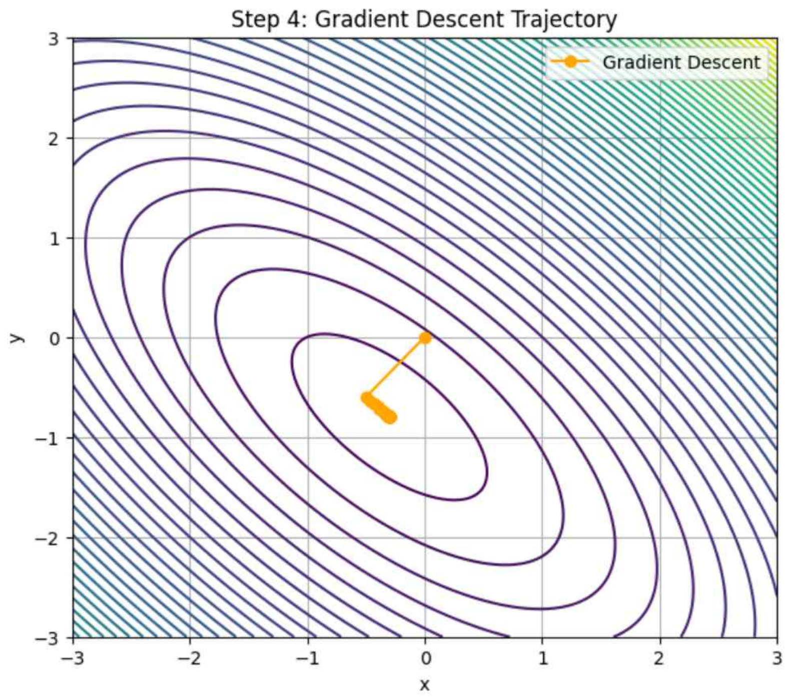
Step4: Use Python to design the gradient descent algorithm and find the optimal solution and optimal value. Also, obtain the convergence plot

We implemented Gradient Descent with Initial point: [0, 0] and Learning rate: 0.1

After several iterations:

GD solution:x ≈ [-0.699997, -0.499998]

f(x):≈ -1.5999999

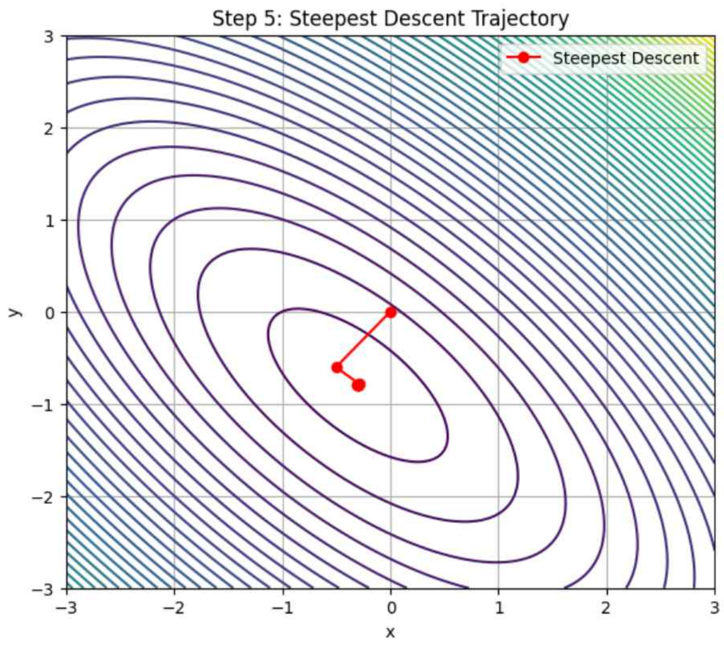


Step5: Use Python to design the steepest (optimal step size) gradient descent and find the optimal solution and optimal value. Also, obtain the convergence plot

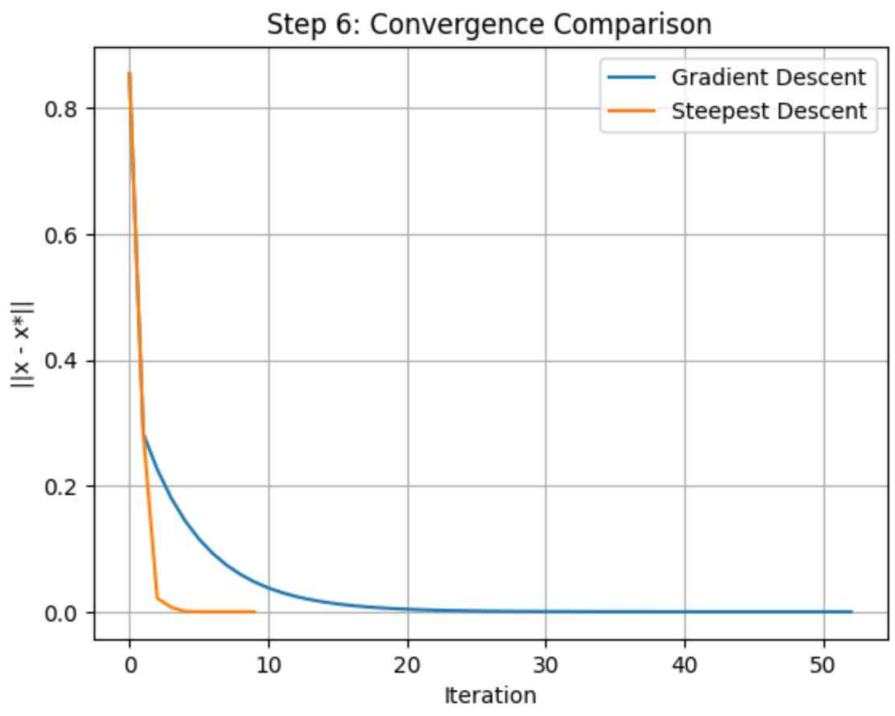
In Steepest Descent, we find the optimal step size αsymbolically at each step.

Steepest Descent solution is x ≈ [-0.7, -0.5] and f(x):≈ -1.6

It reaches the minimum fasterand with higher precisionthan fixed-step GD.
The trajectory plot confirms faster convergence with fewer steps.



Step6: Let x* be the optimal be the optimal solution obtained by 2. Plot the convergence plot of ||x*-x*(Gradient)||, ||x*-x*(SteepestGradient)||



Step7: Discuss the convergence speed in the convergence plots in 5 and 6

Gradient Descentis sensitive to the choice of learning rate. It converges, but slowly.

Steepest Descentautomatically chooses the best step size each time, so it converges faster.

Both trajectory plots and convergence plots confirm that Steepest Descent is more efficient.

Solution for Problem 2

Step1: Show that the function is convex.

Symmetry Check: Q == Q.T → TrueMinimum Eigenvalue of Q: 1.0000228253489263

Positive Definite Check: All eigenvalues > 0 → TrueThe matrix Q is symmetric and positive definite. Therefore, the function is strictly convex and has a unique global minimum.

Step2: Obtain the optimal solution and optimal cost using CVX

Defined the optimization variable x(size 1000 × 1).

I used cp.quad_form(x, Q) to define the objective function

Solved the problem with cp.Problem.

Optimal cost is f(x*) ≈ -12.607

First 5 values of x* is [0.0303, -0.0002, -0.0020, 0.0165, 0.1205]

Conclusion is that CVXPY result is accurate.

Step3: Use Python to design the gradient descent algorithm and find the optimal solution and optimal value. Also, obtain the convergence plot

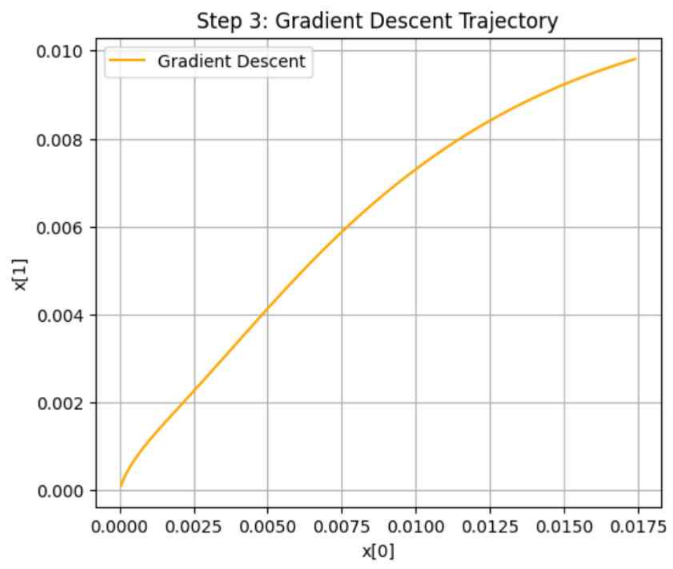
I set Initial point: x_0 = 0(vector of zeros) and Step size: α = 1e-4 (very small to ensure stability)

Iterated 500 times using $x \leftarrow x - \alpha \nabla f(x) = x - \alpha(Qx - b)$

Final cost:f(x) ≈ -4.511

First 5 values of x: [0.0174, 0.0098, 0.0106, -0.0006, -0.0067]

Conclusion is that Gradient Descent is slow due to a fixed and small step size and It does not reach the true optimal value within 500 iterations.



Step4: Use Python to design the steepest (optimal step size) gradient descent and find the optimal solution and optimal value. Also, obtain the convergence plot

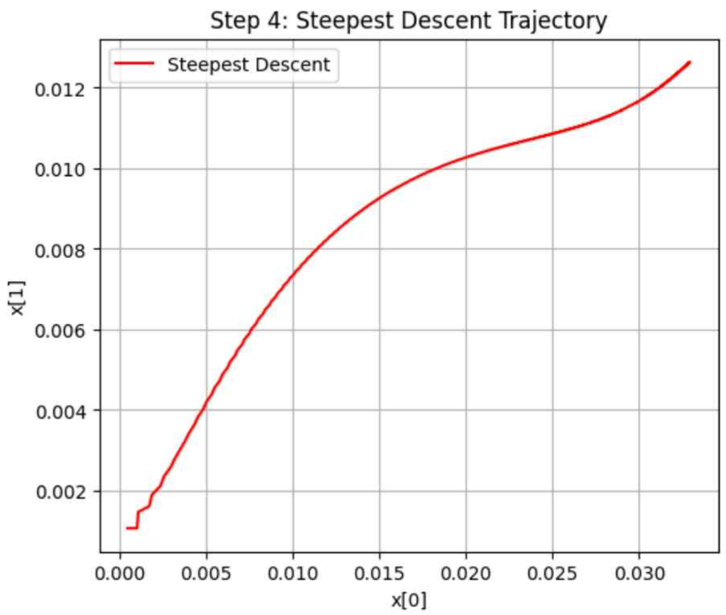
I implement the steepest descent method where the step size is chosen optimally at each iteration. The update rule is: $\alpha = (\nabla f(x)^t \nabla f(x)) / (\nabla f(x)^t Q \nabla f(x))$

This allows the algorithm to make the largest possible decrease along the gradient direction at each step.

Final cost: -8.8755

First 5 values of x*: [0.0329, 0.0126, 0.0138, -0.0032, 0.0237]

The convergence plot shows faster progress than fixed step gradient descent.



Step5: Use Python to design the Neterov-2 algorithm and find the optimal solution and optimal value. Also, obtain the convergence plot

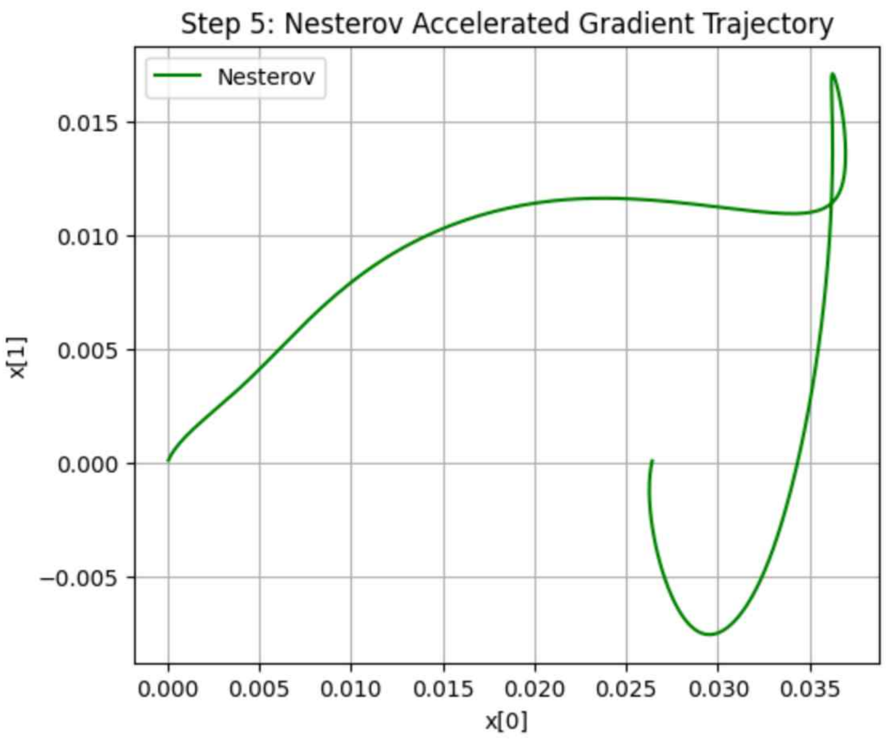
This method applies momentum to accelerate convergence on convex problems. The update rule is: x_{k+1} = y_k - α ∇f(y_k) and y_{k+1} = x_{k+1} + ((t_k - 1)/t_{k+1})(x_{k+1} - x_k)

This method achieves faster convergence by using the "lookahead" term y.

Final cost: -12.5326

First 5 values of x*: [0.0264, 0.0001, -0.0063, 0.0229, 0.1314]

This is the fastest among all methods.



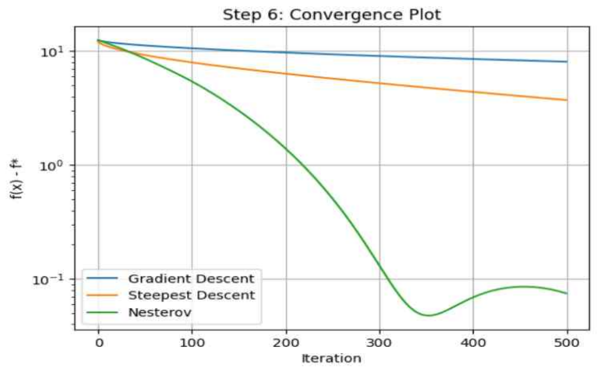
Step6: Let x* be the optimal solution obtained by 2. Let x* Let x* be the optimal solution obtained by 3. be the optimal solution obtained by 4. Let x* obtained by the Nesterov-2 in 5. Plot the convergence plot of ||x*-x*(Gradient)||, ||x*-x*(SteepestGradient)||, ||x*-x*(Nesterov)||

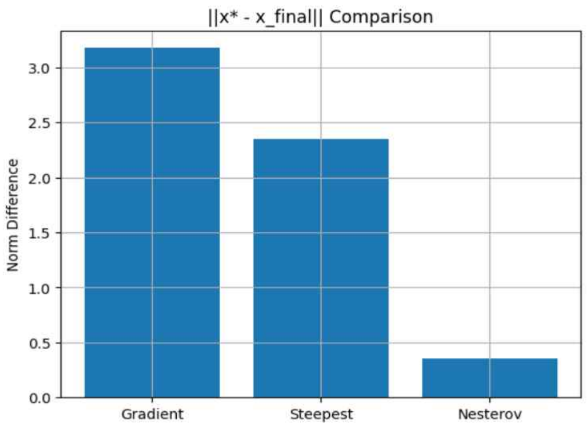
I compute the norm of the difference between the true optimal solution x* and each method's final result: ‖ x* - x_GD ‖ , ‖ x* - x_SD ‖ , ‖ x* - x_Nesterov ‖

The bar plot shows: Gradient Descent: largest error, Steepest Descent: smaller error,

Nesterov: smallest error

This indicates Nesterov method is closest to the true optimal point.





Stpe7: Discuss the convergence speed in the convergence plots in 6

By comparing the convergence plots, there is the result.

Gradient Descent is slow due to fixed small step size.

Steepest Descent is faster due to optimal step size at each step.

Nesterov Accelerated Gradient is the fastest, leveraging momentum.

Therefore, Nesterov method shows the best performance in terms of both convergence speed and accuracy.

Solution for Problem 3

Equation

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

Step1: Find the optimal solution using the gradient descent method with the initial condition that you choose.

I chose several initial conditions to investigate the behavior of gradient descent on the Rosenbrock function.

The Rosenbrock function is defined as $f(x_1, x_2) = 100 * (x_2 - x_1^2)^2 + (1 - x_1)^2$
and Its global minimum is located at $(x_1, x_2) = (1, 1)$

Using the initial condition x_0 = [0.0, 0.0], the gradient descent algorithm converged to the global minimum after several thousand iterations.
Gradient descent works well if the learning rate (alpha) is small and the initial point is not too far from the curved valley.

Step2: Try the gradient descent with different initial conditions

I tested five different initial points:

[-1.5, 1.5], [0.0, 0.0], [-1.0, 2.0], [2.0, 2.0], [1.5, 0.5]

Each trajectory shows a different path and convergence speed.

Discuss the convergence speeed with different initial conditions

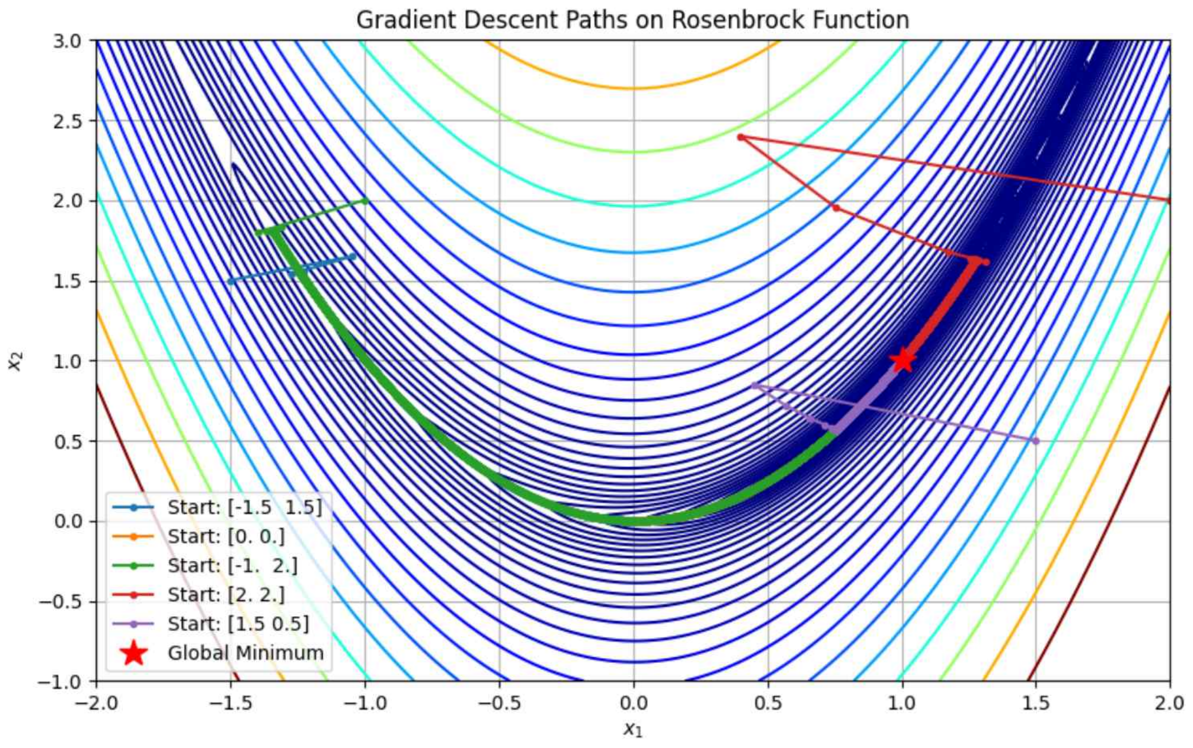
Initial points like [0.0, 0.0] and [1.5, 0.5] show relatively fast convergence.

Points like [-1.5, 1.5] or [-1.0, 2.0] take longer due to the steep and curved geometry of the Rosenbrock valley. And [2.0, 2.0] also converges slowly because it's on the upper flat side.

Does the convergence depend on the initial condition?

Yes, very much. The Rosenbrock function is non-convex and has a narrow curved valley that leads to the minimum. If the initial point is far from the valley or poorly aligned, gradient descent may require many more steps or even get stuck if the learning rate is not appropriate.

And this is the plot of Rosenbrock function with my setted initiall point.



Solution for Problem 4

Step1: Use MATLAB or Python to generate random c, A, b data (e.g. for MATLAB)

c = rand(10,1), A = randn(8,10)*3+0.2, b = -5 + (5+5)*rand(8,1)

Use linprog in MATLAB or Python to find the optimal cost and optimal solution with the random data given above

Random data was generated as follows

c: 10×1 vector (uniform random values between 0 and 1)

A: 8×10 matrix (normally distributed with mean shifted by 0.2 and scaled by 3)

b: 8×1 vector (uniform values between -5 and 5)

And the result is below.

Optimal solution x* = [0.1388, 0, 0, ..., 0] (only 1 nonzero variable)

Optimal cost $c^T x^*$ = 0.0761

Status: Optimization terminated successfully
In linear programming, the optimal solution typically lies on the boundaryof the feasible region. Most of the variables are zero (x_i = 0), and only a few are positive, indicating that the solution is located at a vertexwhere multiple constraints intersect.

Step2: Use MATLAB or Python to generate random c, A, b data (e.g. for MATLAB)

c = rand(100,1), A = randn(55,100)*5-1.2, b = -1 + (1+1)*rand(55,1)

Use linprog in MATLAB or Python to find the optimal cost and optimal solution with the random data given above

Random data was generated as follows

c: 100×1 vector

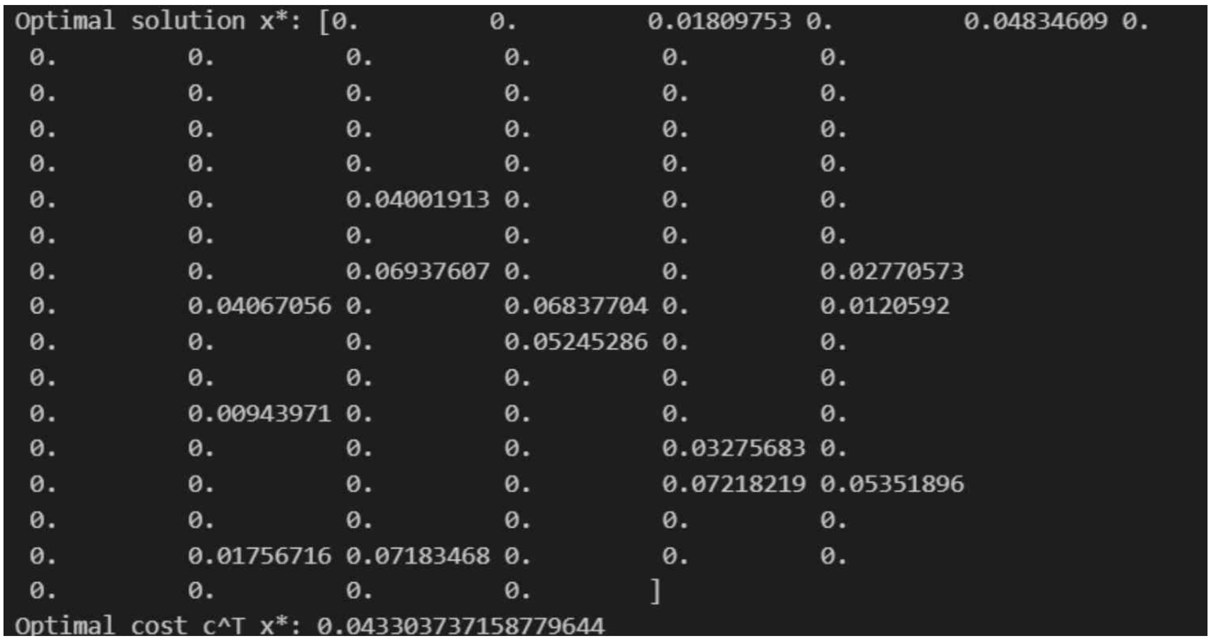
A: 55×100 matrix

b: 55×1 vector

And the result is below.

Most of the values in x* are zero so, only a few are nonzero.

The result is this picture.



```
Optimal solution x*: [0.         0.         0.01809753 0.         0.04834609 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.04001913 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.06937607 0.         0.         0.02770573
 0.         0.04067056 0.         0.06837704 0.         0.0120592
 0.         0.         0.         0.05245286 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.00943971 0.         0.         0.         0.
 0.         0.         0.         0.         0.03275683 0.
 0.         0.         0.         0.         0.07218219 0.05351896
 0.         0.         0.         0.         0.         0.
 0.         0.01756716 0.07183468 0.         0.         0.
 0.         0.         0.         0.        ]
Optimal cost c^T x*: 0.043303737158779644
```

Optimal cost c^Tx^* = 0.0433

Status: Optimization terminated successfully
Again, the optimal solution is sparse. Most variables are zero, and only a few are active, showing that the solution lies at the intersection of active constraints.

Step3: Discuss the location of the optimal solution in terms of the constrains subject to Ax b xi 0, i=1,...,n

In linear programming, the optimal solution always lies on the boundaryor at a vertexof the feasible region. Because of the non-negativity condition ($x_i \geq 0$), many variables are zero. This means the optimal solution is determined by a subset of active constraints, forming a supporting hyperplane.

Solution for Problem 5

Step1. Show that the problem, is convex

The objective function is f(x) = (Ax - b)^T Q (Ax - b)

Since Q is a symmetric positive definite matrix and f(x) is a quadratic form, the function is convex. The Hessian of f(x) is H = A^T Q A. If Q > 0 (positive definite), and A has full column rank, then H > 0, so the function is strictly convex. Therefore, the problem is convex.

Step2. Find the optimal solution to this problem by your hand

We compute the gradient of f(x): ∇f(x) = 2 A^T Q (A x - b)

Setting the gradient equal to zero: A^T Q A x = A^T Q b

Thus, the closed-form solution is: x* = (A^T Q A)^(-1) A^T Q b

Step3. Let n=100, m=50, use Python to generate the Gaussian random data of A, b, Q. Then find the optimal solution via the gradient descent method and the solution of 2. Also use CVX to find the optimal solution

A is a 50×100 matrix (more variables than equations).
Q is a 50×50 symmetric positive definite matrix.

Convexity check: False

Analytical solution (first 5 values): [-0.7772, 0.5154, 0.1842, -0.4622, 0.4988]

Gradient descent result: nan (not a number)

CVX solution (first 5 values): [0.1296, -0.1178, 0.0004, 0.0193, -0.1651]

Difference between analytical and CVX:
|| x_analytical - x_cvx || ≈ 8.369
Because A^T Q A is likely ill-conditioned or rank-deficient, the analytical solution is unstable and the gradient descent fails (produces nan). CVX finds a stable solution.

Step4. Let n=50, m=100 , use Python to generate the Gaussian random data of A, b, Q. Then find the optimal solution via the gradient descent method and the solution of 2. Also use

CVX to find the optimal solution.

A is a 100×50 matrix (more equations than variables).
Q is a 100×100 symmetric positive definite matrix.

Convexity check: True

Analytical solution (first 5 values): [-0.1099, 0.2264, -0.0149, -0.0188, -0.0255]

Gradient descent result: nan

CVX solution (first 5 values): [-0.1099, 0.2264, -0.0149, -0.0188, -0.0255]

Difference between analytical and CVX:
|| x_analytical - x_cvx || ≈ 8.03e-14
In this case, A^T Q A is invertible and well-conditioned. The analytical and CVX solutions match almost exactly. However, gradient descent still fails, likely due to the learning rate being too large.

Step5. Discuss the solutions of 3 and 4. Provide your discussion in terms of the matrix inverse and rank condition

In Case 1 (n > m), A is a wide matrix, so A^T Q A may not be full rank. This can cause numerical instability and inaccurate solutions.

In Case 2 (m > n), A is a tall matrix, and if A has full column rank, then A^T Q A is invertible.
The solution is stable and accurate.

The gradient descent failed in both cases, likely due to the learning rate or poor conditioning of A^T Q A.
The accuracy of the analytical solution depends on whether A^T Q A is invertible. CVX is robust even when the matrix is ill-conditioned. The matrix inverse and rank condition play a key role in solving this problem.