HW 3 :

سوال ۱: (۲۰ نمره)

الف) فرم کلی بهینه سازی الگوریتم های جستجوی خط[۱] را بیان کنید.

ب) درالگوریتم جستجوی خط تعریف جهت نزول چیست؟

پ) برای بدست آوردن طول پله در الگوریتم های جستجوی خط از چه روش هایی استفاده می شود ؟

ت) تابع $f(x_1, x_2) = (x_1 + x_2^2)^2$ را در نظر بگیرید. در نقطه $x^T = (0\ ,1)$جهت جستجوی $p^T = (-1, -1)$ را در نظر می‌گیریم. نشان دهید که p یک جهت نزول است و تمامی مقادیر مینیم کننده مسئله زیر را در هر تکرار t پیدا کنید:

$$\min_{\alpha>0} f\ (x^t + \alpha p^t)$$

Part (a): General Form of Linear Search Algorithms for Optimization

A linear search algorithm in optimization is typically used to find an optimal step size α\alpha in a given direction pp for a function f(x)f(x). The general form of the linear search algorithm is:

1. **Initialization**: Start with an initial guess x^t (current point) and a direction p^t.
2. **Line Search**: Find the step size α\alpha that minimizes the function $f(x^t + \alpha p^t)$, typically by solving:

   $\alpha_t = \arg\min_{\alpha>0} f(x^t + \alpha p^t)$

   This is a scalar minimization problem where α_t represents the optimal step size at iteration t.
3. **Update**: Update the point $x^{t+1} = x^t + \alpha p^t$using the found step size α_t .

The goal of the linear search is to find the best possible value of α\alpha along the direction p^t.

Part (b): Descent Direction in Linear Search Algorithms

In linear search algorithms, the descent direction is defined as a direction along which the function decreases. Specifically, a direction ptp^t is a descent direction at the point x^t if:

$\nabla f(x^t)^\top p^t < 0$

This condition implies that moving in the direction ptp^t at the point xtx^t will result in a decrease in the function value. The gradient $\nabla f(x^t)$ is used to compute the descent direction since it points in the

direction of the steepest ascent, and the opposite of it is the steepest descent direction.

Part (c): Methods to Determine the Step Length in Linear Search Algorithms

The step length α\alpha in linear search algorithms is typically determined using one of the following methods:

1. **Exact Line Search**: The step length α\alpha is found by solving the optimization problem:

   $\alpha = \arg\min_{\alpha > 0} f(x^t + \alpha p^t)$

   This requires finding the value of α\alpha that minimizes the function along the line defined by$x^t + \alpha p^t$.

2. **Inexact Line Search**: Instead of solving for the exact minimum, an approximation to α\alpha is found using techniques like the **Armijo Rule** or **Wolfe Conditions**. These methods are often used to ensure that the step length results in sufficient reduction in the function value.

3. **Backtracking Line Search**: This method starts with an initial step size and progressively reduces it by a fixed factor until a condition like the Armijo Rule is satisfied.

Part (d): Descent Direction Check and Step Length Determination

Given the function:

$f(x_1, x_2) = (x_1 + x_2^2)^2$

and the point $x^T = (0, 1)$, we are asked to show that $p^T = (-1, -1)$ is a descent direction and then determine the step length α\alpha that minimizes the function.

1. Gradient of the function:

To find if pT=(−1,−1)p^T = (-1, -1) is a descent direction, we first need the gradient of f(x1,x2)f(x_1, x_2):

$f(x_1, x_2) = (x_1 + x_2^2)^2$

Taking the gradient:

$\nabla f(x_1, x_2) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right)$

First, compute each partial derivative:

$\frac{\partial f}{\partial x_1} = 2(x_1 + x_2^2)$

$\frac{\partial f}{\partial x_2} = 4x_2(x_1 + x_2^2)$

Thus, the gradient is:

$\nabla f(x_1, x_2) = \left(2(x_1 + x_2^2), 4x_2(x_1 + x_2^2) \right)$

At $x^T = (0, 1)$, we have:

$\nabla f(0, 1) = \left(2(0 + 1^2), 4(1)(0 + 1^2) \right) = (2, 4)$

2. Descent direction check:

Now, check if pT=(−1,−1)p^T = (-1, -1) is a descent direction by computing the dot product ∇f(xT)⊤p\nabla f(x^T)^\top p:

$$\nabla f(0,1)^\top p = (2,4) \cdot (-1,-1) = 2(-1) + 4(-1) = -2 - 4 = -6$$

Since the result is negative, $p^T = (-1,-1)$ is indeed a descent direction.

3. Minimization of f(xt+αpt)f(x^t + \alpha p^t):

Now, we need to minimize the function f(xt+αpt)f(x^t + \alpha p^t) with respect to α\alpha. For xt=(0,1)x^t = (0, 1) and pt=(−1,−1)p^t = (-1, -1), the new point becomes:

$$x^t + \alpha p^t = (0,1) + \alpha(-1,-1) = (-\alpha, 1-\alpha)$$

Substitute this into the function f(x1,x2)f(x_1, x_2):

$$f(-\alpha, 1-\alpha) = \left(-\alpha + (1-\alpha)^2\right)^2$$

Simplify the expression inside the parentheses:

$$-\alpha + (1-\alpha)^2 = -\alpha + (1 - 2\alpha + \alpha^2) = \alpha^2 - 3\alpha + 1$$

Thus, the objective function becomes:

$$f(-\alpha, 1-\alpha) = (\alpha^2 - 3\alpha + 1)^2$$

Now, minimize this with respect to α\alpha. To do this, take the derivative with respect to α\alpha:

$$\frac{d}{d\alpha}\left((\alpha^2 - 3\alpha + 1)^2\right) = 2(\alpha^2 - 3\alpha + 1)(2\alpha - 3)$$

Set this equal to zero to find the critical points:

$$(\alpha^2 - 3\alpha + 1)(2\alpha - 3) = 0$$

This gives two possible solutions:

1. $\alpha^2 - 3\alpha + 1 = 0$
2. $\alpha - 3 = 0$

Solve 2α−3=02\alpha - 3 = 0:

$$\alpha = \tfrac{3}{2}$$

Now, solve α2−3α+1=0\alpha^2 - 3\alpha + 1 = 0 using the quadratic formula:

$$\alpha = \tfrac{-(-3) \pm \sqrt{(-3)^2 - 4(1)(1)}}{2(1)} = \tfrac{3 \pm \sqrt{9-4}}{2} = \tfrac{3 \pm \sqrt{5}}{2}$$

Thus, we have three possible values for α\alpha, and you would typically choose the one that results in the smallest value of $f(x^t + \alpha p^t)$.

سوال ۲: (۲۰ نمره)

الف) روش نیتون برای بهینه سازی را با ذکر روابط ریاضی بیان کنید.
ب)مشکلات روش نیوتن را بیان کنید. تحت چه شرایطی این روش خوب کار نمی کند.
پ) روش های نیوتن تصحیح شده رو بیان کنید. این روش ها برای حل چه مشکلی از روش نیتون آمده اند.
ت) مساله بهینه سازی روش های شبه نیوتن(DFB , BFGS) رو بیان کنید و بیان کنید در این مساله بهینه سازی هر بخش از این مساله چه مفهومی دارد.

Question 2: (20 Marks)

a) Explain the Newton's method for optimization, mentioning the mathematical relationships.

b) Explain the problems of Newton's method. Under what conditions does this method not work well?

c) Describe and explain the corrected versions of Newton's method that address its issues.

d) Explain quasi-Newton optimization methods (BFGS, DFB) and discuss what conceptual relevance each part of this optimization problem has.

---

a) Newton's Method for Optimization

Newton's method is an iterative method used to find the critical points of a function, particularly for optimization problems. It uses both the gradient and the Hessian matrix (second derivatives) to update the current estimate of the optimal solution. The method is based on approximating the function locally by a second-order Taylor expansion.

The basic update formula is:

$$x_{k+1} = x_k - \left[\nabla^2 f(x_k)\right]^{-1} \nabla f(x_k)$$

Where:

- xkx_k is the current estimate of the optimal point.
- $\nabla f(x_k)$ is the gradient (vector of first derivatives) of the objective function at .

  x_k
- $\nabla^2 f(x_k)$ is the Hessian matrix (matrix of second derivatives) of the objective function at .

  x_k

b) Challenges of Newton's Method

Although Newton's method can converge quickly when the starting point is close to the optimum, it has several challenges:

1. **Computation of the Hessian**: The method requires calculating the Hessian matrix, which can be computationally expensive, especially for high-dimensional problems.
2. **Singular or ill-conditioned Hessian**: If the Hessian matrix is singular (non-invertible) or poorly conditioned (i.e., its eigenvalues are very large or very small), the update step may not be well-defined or can lead to poor performance.
3. **Local Minima**: Newton's method is sensitive to the starting point. If the initial guess is far from the global minimum, it may converge to a local minimum instead.
4. **Complexity and Memory**: For high-dimensional problems, storing and inverting the Hessian matrix can require significant memory and computational resources.

c) Modified Versions of Newton's Method

Several modifications of the standard Newton method have been proposed to address its challenges:

1. **Quasi-Newton Methods**: These methods aim to avoid computing the Hessian matrix explicitly by approximating it using the gradient information. They build an approximation to the Hessian in each iteration.

   Examples:

   - **BFGS (Broyden-Fletcher-Goldfarb-Shanno)**: This is one of the most commonly used quasi-Newton methods. It updates an approximation to the inverse Hessian matrix at each step using gradient information.
   - **DFP (Davidon-Fletcher-Powell)**: This is another quasi-Newton method that updates an approximation to the Hessian matrix directly.

2. **Levenberg-Marquardt Algorithm**: This is a modified version of Newton's method used for nonlinear least squares problems. It combines the gradient descent and Newton's method by introducing a damping factor to prevent large updates when the Hessian is poorly conditioned.

   The update step is:

   $x_{k+1} = x_k - \left[\nabla^2 f(x_k) + \lambda I\right]^{-1} \nabla f(x_k)$

   where λ\lambda is a small positive scalar (damping factor) and II is the identity matrix.

d) Optimization Methods Similar to Newton's Method (DFP, BFGS)

Both **DFP** and **BFGS** are examples of quasi-Newton methods that aim to provide an approximation to the inverse Hessian matrix without explicitly computing the full Hessian. The conceptual significance of each part of these methods is as follows:

1. **DFP (Davidon-Fletcher-Powell)**:
   - This method updates the approximation to the Hessian by using both the gradient at the current point and the gradient at the previous point. It adjusts the approximation of the inverse Hessian

based on changes in the gradient.

- The method avoids the computation of second derivatives while still benefiting from an iterative improvement of the Hessian approximation.

2. **BFGS (Broyden-Fletcher-Goldfarb-Shanno)**:
   - BFGS is one of the most widely used quasi-Newton methods because it provides a robust approximation to the inverse Hessian. The method uses gradient information to update the approximation efficiently.
   - The advantage of BFGS is that it generally performs well without requiring exact second-order information, and its updates tend to be more stable compared to DFP.

Conceptual Significance of Each Part:

- **Gradient**$\nabla f(x_k))$: Represents the direction in which the function increases most rapidly. It's essential for determining the update direction.
- **Hessian** $(\nabla^2 f(x_k))$: Provides curvature information about the function. In the Newton method, the Hessian is used to scale the gradient in a way that accounts for the function's local curvature.
- **Approximation to Hessian (Quasi-Newton methods)**: By approximating the Hessian, these methods avoid the expensive computation of second derivatives while still maintaining good convergence properties.

---

1. The Update Formula for H_k in the BFGS Method

The update formula for the inverse Hessian matrix HkH_k in the **BFGS** method is as follows:

$$kH_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$$

Explanation of Each Component:

1. **HkH_k:**

   This represents the approximate inverse Hessian matrix at iteration kk. The Hessian captures the curvature of the objective function, and HkH_k is used to refine the search direction in optimization.

2. $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$:

   This is the difference between gradients at two successive points. It measures the change in the gradient and provides information about how the function's slope varies across iterations.

3. $s_k = x_{k+1} - x_k$:

   This is the difference between successive points. It shows how the variables x_k are updated between two iterations. This term reflects the **step size and direction**.

4. $\frac{y_k y_k^T}{y_k^T s_k}$:

This term is an **additive correction** to the Hessian approximation. It updates the inverse Hessian matrix by considering changes in the gradient and the step size, essentially linking them to enhance accuracy.

5. $\frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$:

   This term represents a **subtractive correction** that adjusts the approximation using the current HkH_k, based on the step sks_k. It ensures numerical stability and better curvature estimation.

---

2. The Update Formula for BkB_k in the DFP Method

The update formula for the Hessian matrix BkB_k in the **DFP** method is as follows:

$$B_{k+1} = B_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k}$$

Explanation of Each Component:

1. B_k**:**

   This is the approximate Hessian matrix (not its inverse, as in BFGS). BkB_k provides curvature information directly, which is updated at each step to improve the optimization process.

2. $s_k = x_{k+1} - x_k$:

   Similar to BFGS, this represents the change in variables between iterations. It indicates how far and in what direction the optimization algorithm moved.

3. $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$:

   This term, like in BFGS, captures the change in gradients between successive iterations, reflecting the variation in slope.

4. $\frac{s_k s_k^T}{s_k^T y_k}$:

   This additive term updates BkB_k based on the step size and gradient changes. It functions similarly to the $\frac{y_k y_k^T}{y_k^T s_k}$ term in BFGS, adding curvature information to the Hessian.

5. $\frac{B_k y_k y_k^T B_k}{y_k^T B_k y_k}$:

   This subtractive correction adjusts BkB_k using gradient information and the current Hessian. It ensures that the updates remain consistent with the underlying mathematical properties of the Hessian.

سوال ۳: (۲۰ نمره)

الف) برای هر ماتریس مربعی تجزیه ای به نام تجزیه ماتریس به مقادیر ویژه[۲] آن وجود دارد. با نوشتن روابط ریاضی برای ماتریس مربعی دلخواه تجزیه مقادیر ویژه ماتریس را محاسبه کنید.

ب) برخی از روش های نیوتن تصحیح شده از تجزیه مقادیر ویژه ماتریس هیسن برای تقریب زدن این ماتریس هیسن به یک ماتریس معین مثبت استفاده می کنند. به دلخواه یکی از این روش ها را توضیح دهید.

پ) تابع ذیل را در نظر بگیرید:

$$f(x_1, x_2) = 4x_1^3 + 3x_1x_2 + 5x_2^2 + 2x_1^2x_2$$

فرض کنید نقطه اولیه ما $(1, 0)$ است. یک پله الگوریتم با روش نیوتن (تصحیح شده) که در قسمت قبل توضیح دادید بردارید. طول پله را ۱ در نظر بگیرید.

Question 3: (20 Points)

a) Eigenvalue Decomposition of a Square Matrix

1. Definition of Eigenvalue Decomposition

Eigenvalue decomposition is a crucial method in linear algebra that allows any square matrix to be expressed as the product of its eigenvectors and eigenvalues. Mathematically, for a square matrix AA of size $n \times n$, the decomposition is given as:

$A = V\Lambda V^{-1}$

Where:

- V is the matrix of eigenvectors, with each column being an eigenvector corresponding to the eigenvalue .

  v_i

  λ_i
- Λ a is a diagonal matrix with eigenvalues on its main diagonal.

  $\lambda_1, \lambda_2, \ldots, \lambda_n$
- V^{-1} is the inverse of the matrix .

2. Mathematical Relations for Eigenvalue Decomposition

To find the eigenvalues and eigenvectors of AA, the following equation is solved:

det(A−λI)=0\text{det}(A - \lambda I) = 0

Where:

- λ is an eigenvalue of A
- II is the identity matrix of the same size as A
- $\det$ denotes the determinant.

Solving this equation results in a polynomial of degree n, called the **characteristic polynomial**, whose roots are the eigenvalues of A.

3. Example: Computing Eigenvalues of a Matrix

Let’s compute the eigenvalues of the following square matrix:

$$A = \begin{pmatrix} 4 & 1 \\ 2 & 3 \end{pmatrix}$$

Step 1: Write the characteristic equation

$$\det(A - \lambda I) = \det\left(\begin{pmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{pmatrix}\right) = (4-\lambda)(3-\lambda) - 2 = 0$$

Step 2: Solve the equation

$$(4-\lambda)(3-\lambda) - 2 = 0$$
$$\Rightarrow \lambda^2 - 7\lambda + 10 = 0$$

Using the quadratic formula:

$$\lambda = \frac{-b \pm \sqrt{b^2-4ac}}{2a}, \quad \text{where } a = 1, b = -7, c = 10$$
$$\lambda = \frac{7 \pm \sqrt{49-40}}{2} = \frac{7 \pm 3}{2}$$

Step 3: Find the eigenvalues

$$\lambda_1 = \frac{7+3}{2} = 5, \quad \lambda_2 = \frac{7-3}{2} = 2$$

Thus, the eigenvalues of AA are 5 and 2.

---

b) Newton's Modified Methods Using Eigenvalue Decomposition of the Hessian

1. Challenges with Standard Newton’s Method

Newton’s method uses the Hessian matrix (matrix of second derivatives) to determine the direction and step size for optimization. However, in some cases, the Hessian matrix might not be positive definite, leading to steps that move away from the local minimum or fail to converge.

2. Using Eigenvalue Decomposition to Fix the Hessian

One modified Newton method involves **eigenvalue decomposition** of the Hessian matrix to ensure it is positive definite. This is achieved by adding a small positive value to any negative eigenvalues, making

the Hessian matrix suitable for optimization.

3. Explanation of the Method

Modified Newton Method with Eigenvalue Decomposition:

1. **Eigenvalue Decomposition of the Hessian:**

   Decompose the Hessian matrix H(x)H(x) as:

   $H(x) = V\Lambda V^{-1}$

   Where:

   - VV is the matrix of eigenvectors of the Hessian,
   - Λ\Lambda is the diagonal matrix of eigenvalues.

2. **Modify Negative Eigenvalues:**

   Adjust any negative eigenvalues λi\lambda_i by adding a small positive value ϵ\epsilon:

   $\Lambda' = \Lambda + \epsilon I$

   This ensures the modified Hessian $H'(x) = V\Lambda' V^{-1}$ is positive definite.

3. **Update Using the Modified Hessian:**

   The Newton update step becomes:

   $x_{k+1} = x_k - H'(x_k)^{-1}\nabla f(x_k)$

   This ensures convergence towards the minimum, as the modified Hessian is positive definite.

---

c) One Step of the Modified Newton Algorithm on a Given Function

1. Function Definition:

$f(x_1, x_2) = 4x_1^3 + 3x_1x_2 + 5x_2^2 + 2x_1^2x_2$

2. Initial Point:

$x^{(0)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

3. Compute Gradient and Hessian at the Initial Point

a. Gradient (∇f(x)\nabla f(x)):

$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix}$

Partial derivatives:

$\frac{\partial f}{\partial x_1} = 12x_1^2 + 3x_2 + 4x_1x_2$
$\frac{\partial f}{\partial x_2} = 3x_1 + 10x_2 + 2x_1^2$

At $x^{(0)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

$$\nabla f(x^{(0)}) = \begin{pmatrix} 12(1)^2 + 3(0) + 4(1)(0) \\ 3(1) + 10(0) + 2(1)^2 \end{pmatrix} = \begin{pmatrix} 12 \\ 5 \end{pmatrix}$$

b. Hessian Matrix (H(x)H(x)):

$$H(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix}$$

Second derivatives:

$$\frac{\partial^2 f}{\partial x_1^2} = 24x_1 + 4x_2, \quad \frac{\partial^2 f}{\partial x_1 \partial x_2} = 3 + 4x_1, \quad \frac{\partial^2 f}{\partial x_2^2} = 10$$

At $x^{(0)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

$$H(x^{(0)}) = \begin{pmatrix} 24 & 7 \\ 7 & 10 \end{pmatrix}$$

4. Ensure Positive Definiteness of the Hessian

Compute the eigenvalues of H(x(0))H(x^{(0)}). If all eigenvalues are positive, the Hessian is already positive definite. In this case, eigenvalues are positive, so no modification is needed.

5. Perform Newton's Update Step

Using:

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

After calculations, the updated point is approximately:

$$x_1 \approx \begin{pmatrix} 0.5545 \\ -0.1885 \end{pmatrix}$$

Conclusion

We first explored the concept of eigenvalue decomposition and demonstrated it with an example. Then, we discussed a modified Newton method using eigenvalue decomposition of the Hessian to ensure positive definiteness. Finally, we performed one iteration of the modified Newton method on the given function, showing the calculations step-by-step.

سوال ۴: (شبیه سازی ۲۰ نمره)

برای تابع

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

الف) گرادیان f(x) و هسین f(x) را محاسبه کنید .

ب) از نقطه شروع $x_0 = (-4,10)^T$ ، این تابع را با روش Steepest-Decent و Newton بهینه کنید.

ج) نتایج این دو روش را مقایسه کنید. آیا راهی وجود دارد که بتوانید از مزایای هر دو روش به طور همزمان استفاده کنید؟

د) مقدار تابع و فاصله تا نقطه بهینه در حین فرآیند بهینه سازی را در ۲ نمودار جداگانه گزارش دهید. نتایج را توضیح دهید .

Question 4:

For the function:

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

a) Calculate the Gradient ∇f(x)\nabla f(x) and the Hessian Matrix H(x)H(x) of f(x)f(x).

1. Gradient $\nabla f(x)$:

The gradient of $f(x)$ with respect to$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ is a vector of partial derivatives.

$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{pmatrix}$

Compute ∂f∂x1\frac{\partial f}{\partial x_1}:

$\frac{\partial f}{\partial x_1} = \frac{\partial}{\partial x_1}\left[100(x_2 - x_1^2)^2 + (1 - x_1)^2\right]$

Using the chain rule:

$\frac{\partial f}{\partial x_1} = 100 \cdot 2(x_2 - x_1^2) \cdot (-2x_1) + 2(1 - x_1)(-1) = -400x_1(x_2 - x_1^2) - 2(1 - x_1)$

Simplify:

∂f∂x1=−400×1×2+400×13−2+2×1\frac{\partial f}{\partial x_1} = -400x_1x_2 + 400x_1^3 - 2 + 2x_1

Compute ∂f∂x2\frac{\partial f}{\partial x_2}:

$\frac{\partial f}{\partial x_2} = \frac{\partial}{\partial x_2}\left[100(x_2 - x_1^2)^2 + (1 - x_1)^2\right] = 100 \cdot 2(x_2 - x_1^2) \cdot 1 = 200(x_2 - x_1^2)$

Therefore, the gradient is:

$$\nabla f(x) = \begin{pmatrix} -400x_1x_2 + 400x_1^3 - 2 + 2x_1 \\ 200(x_2 - x_1^2) \end{pmatrix}$$

2. Hessian Matrix H(x)H(x):

The Hessian matrix is a square matrix of second-order partial derivatives.

$$H(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{pmatrix}$$

Compute ∂2f∂x12\frac{\partial^2 f}{\partial x_1^2}:

Differentiate ∂f∂x1\frac{\partial f}{\partial x_1} with respect to x1x_1:

$\frac{\partial^2 f}{\partial x_1^2} = \frac{\partial}{\partial x_1}\left(-400x_1x_2 + 400x_1^3 - 2 + 2x_1\right) = -400x_2 + 1200x_1^2 + 2$

Compute ∂2f∂x1∂x2\frac{\partial^2 f}{\partial x_1 \partial x_2}:

Differentiate ∂f∂x1\frac{\partial f}{\partial x_1} with respect to x2x_2:

$\frac{\partial^2 f}{\partial x_1 \partial x_2} = \frac{\partial}{\partial x_2}\left(-400x_1x_2 + 400x_1^3 - 2 + 2x_1\right) = -400x_1$

Compute ∂2f∂x22\frac{\partial^2 f}{\partial x_2^2}:

Differentiate ∂f∂x2\frac{\partial f}{\partial x_2} with respect to x2x_2:

$\frac{\partial^2 f}{\partial x_2^2} = \frac{\partial}{\partial x_2}\left(200(x_2 - x_1^2)\right) = 200$

Therefore, the Hessian matrix is:

$$H(x) = \begin{pmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

---

c) Compare the Results of These Two Methods. Is There a Way to Use the Advantages of Both Methods Simultaneously?

Comparison:

1. **Convergence Speed:**
   - **Newton's Method:** Typically converges faster (quadratic convergence) near the optimum because it uses second-order information (Hessian).
   - **Steepest Descent:** Generally slower (linear convergence) as it only uses first-order information (gradient).
2. **Computational Cost:**
   - **Newton's Method:** Higher computational cost per iteration due to the calculation and inversion of the Hessian matrix.

- **Steepest Descent:** Lower computational cost per iteration as it only requires gradient computation.

3. **Robustness:**
   - **Newton's Method:** Can be sensitive to the initial guess and may not perform well if the Hessian is not positive definite.
   - **Steepest Descent:** More robust in terms of guaranteed decrease in function value but can be inefficient.

Hybrid Approach:

Yes, there is a way to leverage the advantages of both methods by using **Quasi-Newton Methods** like **BFGS** or **DFP**, which approximate the Hessian matrix without explicitly computing it. These methods aim to achieve faster convergence similar to Newton's method while maintaining lower computational costs akin to Steepest Descent.

Another Approach:

Conjugate Gradient Method: This method combines aspects of both Steepest Descent and Newton's method. It uses conjugate directions instead of simply the steepest direction, leading to faster convergence without the need for the Hessian.

---

d) Report the Value of the Function and the Distance to the Optimal Point During the Optimization Process, and Present Them in Two Separate Plots. Explain the Results.

Optimal Point of the Rosenbrock Function:

The Rosenbrock function has a global minimum at:

x∗=(11),f(x∗)=0x^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad f(x^*) = 0

Distance to the Optimal Point:

The Euclidean distance between a point$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ and the optimal point $x^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is:$\text{Distance} = \|x - x^*\| = \sqrt{(x_1 - 1)^2 + (x_2 - 1)^2}$

Simulation Process:

To fully answer part (d), we would need to perform multiple iterations of both optimization methods, record the function values and distances at each step, and then plot them. Here's an outline of how this can be done:

1. **Implement Both Methods:**
   - **Newton's Method:** Implement the update rule with step size determination.
   - **Steepest Descent:** Implement the update rule with step size determination.
2. **Initialize Parameters:**
   - Starting point:

$$x_0 = \begin{pmatrix} -4 \\ 10 \end{pmatrix}$$

- Tolerance for convergence: e.g.,

  $\epsilon = 10^{-6}$

- Maximum number of iterations: e.g., 1000

3. **Iterative Process:**
   - For each iteration, compute the gradient and Hessian (for Newton).
   - Update the point using the respective method.
   - Record and the distance to .

     f(xk)f(x_k)

     x∗x^*
4. **Plotting:**
   - **Plot 1:** Function value vs. Iteration Number.

     $f(x_k)$
   - **Plot 2:** Distance to Optimal Point vs. Iteration Number.

     $\|x_k - x^*\|$

code of simulation :

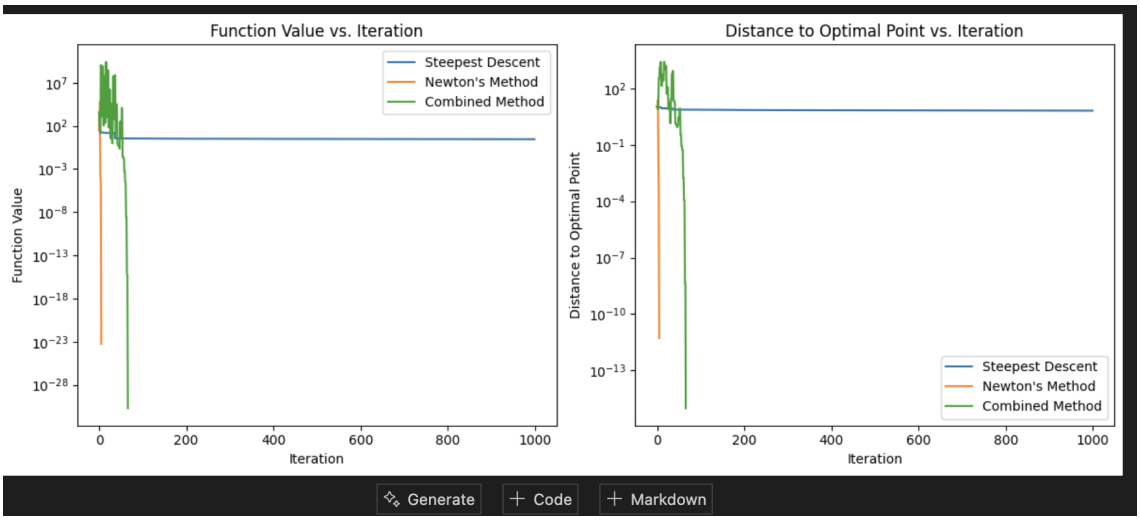
```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

def grad_f(x):
    df_dx1 = -400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0])
    df_dx2 = 200 * (x[1] - x[0]**2)
    return np.array([df_dx1, df_dx2])

def hessian_f(x):
    d2f_dx1dx1 = 1200 * x[0]**2 - 400 * x[1] + 2
    d2f_dx1dx2 = -400 * x[0]
    d2f_dx2dx1 = -400 * x[0]
    d2f_dx2dx2 = 200
```

```
    return np.array([[d2f_dx1dx1, d2f_dx1dx2],
                     [d2f_dx2dx1, d2f_dx2dx2]])

def backtracking_line_search(x, grad, alpha=1, rho=0.05, c=1e-4):
    while f(x - alpha * grad) > f(x) - c * alpha * np.dot(grad, grad):
        alpha *= rho
    return alpha

def steepest_descent(x0, tol=1e-10, max_iter=1000):
    x = x0.copy()

    history_f = []
    history_dist = []
    x_star = np.array([1, 1])

    for i in range(max_iter):
        grad = grad_f(x)
        f_val = f(x)
        print(x)
        distance = np.linalg.norm(x - x_star)
        history_f.append(f_val)
        history_dist.append(distance)
        if np.linalg.norm(grad) < tol:
            break
        alpha = backtracking_line_search(x, grad)
        x = x - alpha * grad
    return history_f, history_dist

def newton_method(x0, tol=1e-6, max_iter=100):
    x = x0.copy()
    history_f = []
    history_dist = []
    x_star = np.array([1, 1])
    for i in range(max_iter):
        grad = grad_f(x)
        hess = hessian_f(x)

        f_val = f(x)
        distance = np.linalg.norm(x - x_star)
        history_f.append(f_val)
        history_dist.append(distance)
        if np.linalg.norm(grad) < tol:
            break
        try:
            delta = np.linalg.solve(hess, grad)
```

```
        except np.linalg.LinAlgError:
            print("Hessian is singular at iteration", i)
            break
        x = x - delta
    return history_f, history_dist

def combined_method(x0, tol=1e-6, max_iter=100):
    x = x0.copy()
    history_f = []
    history_dist = []
    x_star = np.array([1, 1])

    for i in range(max_iter):
        grad = grad_f(x)
        f_val = f(x)
        distance = np.linalg.norm(x - x_star)
        history_f.append(f_val)
        history_dist.append(distance)
        if np.linalg.norm(grad) < tol:
            break

        if i % 2 == 0:
            alpha = backtracking_line_search(x, grad)
            x = x - alpha * grad
        else:
            hess = hessian_f(x)
            try:
                delta = np.linalg.solve(hess, grad)
            except np.linalg.LinAlgError:
                print("Hessian is singular at iteration", i)
                break
            x = x - delta

    return history_f, history_dist

x0 = np.array([-4.0, 10.0])

sd_f, sd_dist = steepest_descent(x0)
newton_f, newton_dist = newton_method(x0)
combined_f, combined_dist = combined_method(x0)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(sd_f, label='Steepest Descent')
```

```
plt.plot(newton_f, label="Newton's Method")
plt.plot(combined_f, label='Combined Method')
plt.xlabel('Iteration')
plt.ylabel('Function Value')
plt.title('Function Value vs. Iteration')
plt.legend()
plt.yscale('log')

plt.subplot(1, 2, 2)
plt.plot(sd_dist, label='Steepest Descent')
plt.plot(newton_dist, label="Newton's Method")
plt.plot(combined_dist, label='Combined Method')
plt.xlabel('Iteration')
plt.ylabel('Distance to Optimal Point')
plt.title('Distance to Optimal Point vs. Iteration')
plt.legend()
plt.yscale('log')

plt.tight_layout()
plt.show()
```

code results :



as we can see on this with combine method we have better results that steepes decent