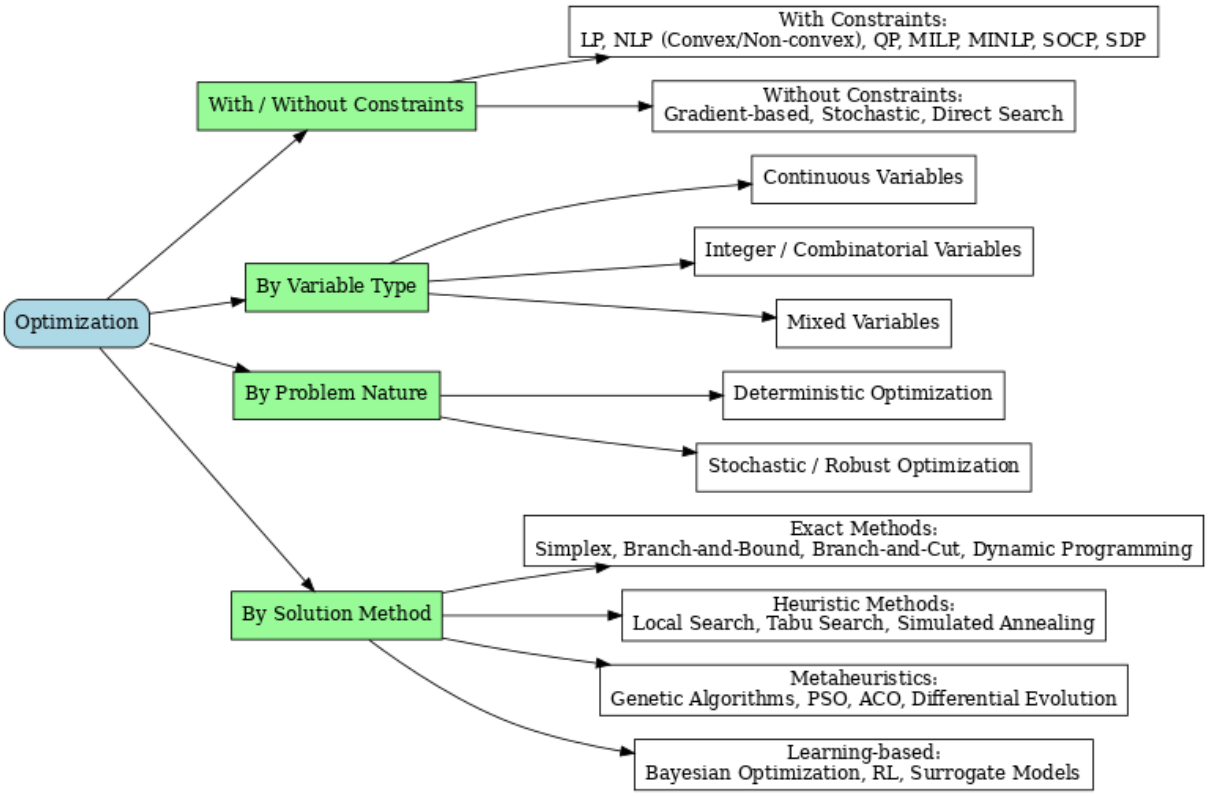

Abbreviations:
LP: Linear Programming | NLP: Nonlinear Programming | QP: Quadratic Programming
MILP: Mixed-Integer Linear Programming | MINLP: Mixed-Integer Nonlinear Programming
SOCP: Second-Order Cone Programming | SDP: Semidefinite Programming
B&B: Branch-and-Bound | B&C: Branch-and-Cut | SA: Simulated Annealing
GA: Genetic Algorithms | PSO: Particle Swarm Optimization | ACO: Ant Colony Optimization
DE: Differential Evolution | RL: Reinforcement Learning

Figure 1: Comprehensive diagram of the main types of optimization and their relationships, with abbreviations explained below the figure.

Chapter 3

Algorithms

Introduction

In this chapter, we will present some algorithms that allow us to compute (approximately) the solution(s) of the initial problem (P). Optimization algorithms are mathematical procedures that aim to determine the set of input parameters of a function giving it its maximum or minimum value. More precisely, they seek to solve:

$$\inf_{x \in \mathbb{R}} f(x) \tag{3.1}$$

where f is a real-valued function called the objective function.

With $X^* = (x_1^*, x_2^*, x_3^*, \ldots)$ being the coordinates of the critical point. Optimization algorithms are used to solve problems of various kinds, such as finding the zeros of nonlinear functions, fitting experimental data using linear and nonlinear least squares criteria, solving systems of equations with one or more variables, etc.

In general, the search for extrema is carried out by computing the first derivatives (the gradient of the function) and the second derivatives (the Hessian of the function). Metaheuristics are a class of optimization algorithms that attempt to obtain an approximate value of the global optimum in difficult optimization problems. However, they do not provide any guarantee on the reliability of the result.

We assume that x^* exists (possibly unique) and aim to find a numerical approximation of x^* by constructing a sequence:

$$\{x^{(k)}\}_{k \in \mathbb{N}} \subset \mathbb{R}^n \tag{3.2}$$

such that

$$x^{(k)} \to x^* \quad \text{as } k \to +\infty. \tag{3.3}$$

The principle is to construct an iterative algorithm of the form:

$$x_{k+1} = x_k - \rho_k d_k \tag{3.4}$$

where d_k is the descent direction and ρ_k is the step size. The step size can be fixed (possibly the same for all iterations, in which case it is called a variable step size method) or computed at each iteration to minimize f in the direction d_k (in which case it is called an optimal step size method).

To approach the optimal solution of problem (3.2) (in the general case, this is a point where the necessary optimality conditions of f are satisfied with a certain precision), one naturally moves from the point x_k in the direction of decrease of the function f.

Definition 3.1 (Algorithm).
An algorithm is defined as an application A from $\mathbb{R}^n$ allowing the generation of a sequence of elements in $\mathbb{R}^n$ by the formula:
Initialization step:
Given $x_0 \in \mathbb{R}^n$, set $k = 0$.
Iteration:

$$x_{k+1} = A(x_k) \tag{3.5}$$

Increment $k = k + 1$.
Writing an algorithm is defining a sequence $(x_k)_{k\in\mathbb{N}}$ in $\mathbb{R}^n$. Studying its convergence means studying the convergence of $(x_k)_{k\in\mathbb{N}}$.

Definition 3.2 (Convergence of an algorithm).
An algorithm A converges if the sequence $(x_k)_{k\in\mathbb{N}}$ converges to a limit x^*.

The error is defined as:

$$e_k = x_k - x^* \tag{3.6}$$

Definition 3.3 (Rate of convergence of an algorithm).
Let $(x_k)_{k\in\mathbb{N}}$ be a sequence converging to x^* generated by algorithm A. Its convergence is:
Linear if

$$\exists C \in]0,1[, \exists k_0 \in \mathbb{N}, \forall k \geq k_0, \quad \|e_{k+1}\| \leq C\|e_k\| \tag{3.7}$$

Superlinear if

$$\lim_{k\to+\infty} \frac{\|e_{k+1}\|}{\|e_k\|} = 0 \tag{3.8}$$

If $\frac{\|e_{k+1}\|}{\|e_k\|}$ converges geometrically to zero, the algorithm's convergence is geometric.
Of order p if

$$\exists C > 0, \exists k_0 \in \mathbb{N}, \forall k \geq k_0, \quad \|e_{k+1}\| \leq C\|e_k\|^p \tag{3.9}$$

If $p = 2$, the convergence is quadratic.

Convergence is **local** if it only occurs for starting points x_0 in a neighborhood of x^*. Otherwise, it is **global**.

Remark 3.1.
If $e_k \neq 0$, linear convergence means $\frac{e_{k+1}}{e_k} = O(1)$, while superlinear convergence implies $\frac{e_{k+1}}{e_k} = o(1)$. Similarly, an algorithm of order p satisfies $\frac{e_{k+1}}{e_k^p} = O(1)$.

It is desirable to have the highest possible convergence rate to achieve the solution

with minimal iterations for a given accuracy.

3.1 Algorithms and Methods for Solving Unconstrained Optimization Problems

Iterative optimality methods (or algorithms) are part of a broader class of numerical methods called **descent methods**.

A **descent direction algorithm** is a differentiable optimization algorithm designed to minimize a real differentiable function defined on a Euclidean space (for example, $\mathbb{R}^n$, equipped with an inner product) or, more generally, on a Hilbert space. The algorithm is *iterative* and thus proceeds by successive improvements. At each current point, a move is made along a descent direction in order to decrease the function value.

3.1.1 Descent Direction Method – One Iteration

Algorithm 1 Descent Direction Method – One Iteration

1: **Step 0 (Initialization):** At the beginning of iteration k, we have an iterate $x_k \in \mathbb{R}^n$.
2: **Step 1 (Stopping test):** If $\nabla f(x_k) \simeq 0$, stop the algorithm.
3: **Step 2 (Choice of descent direction):** Choose a descent direction $d_k \in \mathbb{R}^n$.
4: **Step 3 (Line search):** Determine a step size $\rho_k > 0$ along d_k such that f decreases sufficiently.
5: **Step 4 (Update):** If the line search succeeds, set

$$x_{k+1} = x_k + \rho_k d_k.$$

Replace k by $k+1$ and return to Step 1.

Examples of choosing a descent direction:

1. If
$$d_k = -\nabla f(x_k), \quad \text{and } \nabla f(x_k) \neq 0, \tag{3.10}$$
we obtain the **gradient method**.

2. In the case
$$d_k = -(H(x_k))^{-1} \nabla f(x_k), \tag{3.11}$$
we obtain **Newton's method**, where the Hessian matrix $H(x_k)$ is positive definite.

Example of choosing the step size ρ_k:
Generally, ρ_k is chosen optimally, that is, it must satisfy:

$$f(x_k + \rho_k d_k) \leq f(x_k + \rho d_k), \quad \forall \rho \in [0, +\infty[. \tag{3.12}$$

In other words, at each iteration we solve a **one-variable minimization problem**, called a **line search**.

Gradient Descent Example

Problem:
Minimize

$$f(x, y) = x^2 + xy + y^2 - 6x - 9y. \tag{3.13}$$

Gradient:

$$\nabla f(x, y) = \begin{pmatrix} 2x + y - 6 \\ x + 2y - 9 \end{pmatrix}.$$

Setting the gradient to zero yields the minimum at $(1, 4)$.

Gradient Descent Iterations (Step size $\rho = 0.1$):

- **Iteration 0:**

$$\begin{aligned} x_0 = 0, \quad y_0 = 0 \\ \nabla f(0, 0) = (-6, -9) \\ d_0 = (6, 9) \\ x_1 = 0 + 0.1 \times 6 = 0.6 \\ y_1 = 0 + 0.1 \times 9 = 0.9 \end{aligned}$$

- **Iteration 1:**

$$\begin{aligned} \nabla f(0.6, 0.9) = (-3.9, -6.6) \\ d_1 = (3.9, 6.6) \\ x_2 = 0.6 + 0.1 \times 3.9 = 0.99 \\ y_2 = 0.9 + 0.1 \times 6.6 = 1.56 \end{aligned}$$

- **Iteration 2:**

$$\begin{aligned} \nabla f(0.99, 1.56) = (-2.46, -4.89) \\ d_2 = (2.46, 4.89) \\ x_3 = 0.99 + 0.1 \times 2.46 = 1.236 \\ y_3 = 1.56 + 0.1 \times 4.89 = 2.049 \end{aligned}$$

The method converges to the minimum at (1,4).

Conceptual Schematic:

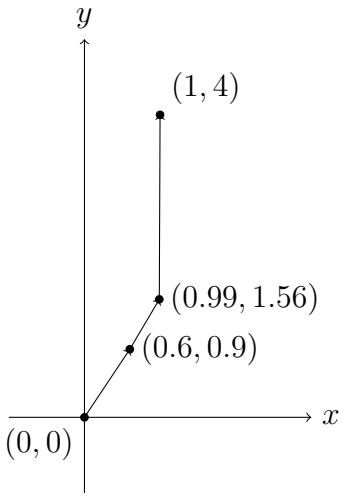

Figure 3.1: Conceptual schematic of the illustration.

MATLAB Code: Gradient Descent Example

Below is the MATLAB program implementing the gradient descent method for:

$$f(x, y) = x^2 + xy + y^2 - 6x - 9y.$$

Program:

```
% Gradient Descent for f(x,y) = x^2 + xy + y^2 -6x -9y

% Initialization
x = 0;
y = 0;
rho = 0.1;

for k = 1:10
    % Compute gradient
    gradx = 2*x + y - 6;
    grady = x + 2*y - 9;

    % Display iteration
    fprintf('Iteration %d: x=%.4f, y=%.4f, f=%.4f\n', k, x, y, x^2 + x*y
        + y^2 -6*x -9*y);

    % Check stopping criterion
    if norm([gradx; grady]) < 1e-4
        break;
    end

    % Descent direction
    dx = -gradx;
    dy = -grady;

    % Update
    x = x + rho*dx;
    y = y + rho*dy;
end

```

```
fprintf('Minimum approximated at x=%.4f, y=%.4f\n', x, y);
```

Listing 3.1: Gradient Descent for a two-variable quadratic function

—

Sample Output:

```
Iteration 1: x=0.0000, y=0.0000, f=0.0000
Iteration 2: x=0.6000, y=0.9000, f=-7.4100
Iteration 3: x=0.9900, y=1.5600, f=-9.6857
Iteration 4: x=1.2360, y=2.0490, f=-10.4976
...
Minimum approximated at x=1.0000, y=4.0000
```

The algorithm converges to the optimal point at $(1, 4)$.

Theoretical Example: Gradient Descent

Consider the problem

$$\min_{x \in \mathbb{R}^n} f(x),$$

where f is convex and differentiable with L-Lipschitz continuous gradient. The gradient descent update is:

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

If f is additionally strongly convex with parameter $\mu > 0$, then

$$\|x_k - x^*\| \leq (1 - \rho\mu)^k \|x_0 - x^*\|,$$

showing linear convergence to the minimizer x^*.

Example: Quadratic Function
Let

$$f(x) = \frac{1}{2} x^T Q x - b^T x,$$

where Q is symmetric positive definite. Then,

$$\nabla f(x) = Qx - b, \quad x^* = Q^{-1} b.$$

Gradient descent iteration:

$$x_{k+1} = x_k - \rho(Qx_k - b).$$

Convergence is guaranteed if $0 < \rho < \frac{2}{L}$, where $L = \lambda_{\max}(Q)$ is the largest eigenvalue of Q.

Gradient Descent Example in 3D

Problem:
Minimize

$$f(x, y, z) = x^2 + y^2 + z^2 + xy + yz + xz - 4x - 5y - 6z.$$

—

Gradient:

$$\nabla f(x, y, z) = \begin{pmatrix} 2x + y + z - 4 \\ x + 2y + z - 5 \\ x + y + 2z - 6 \end{pmatrix}.$$

—

Solution:
Solving

$$\nabla f(x, y, z) = 0$$

yields

$$(x^*, y^*, z^*) = (0.25, 1.25, 2.25).$$

—

Conceptual 3D Schematic:

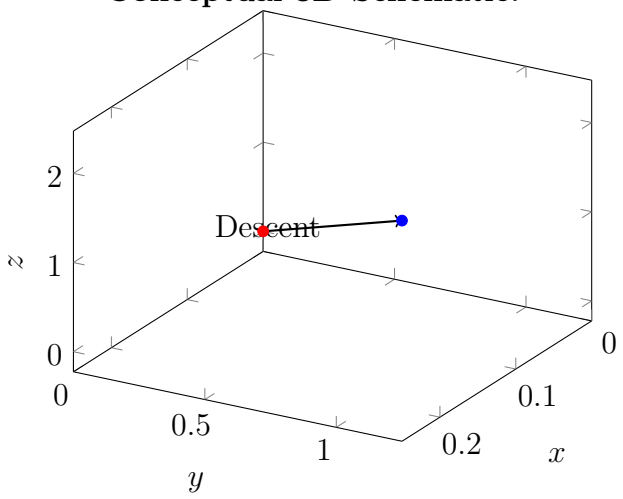

Figure 3.2: Conceptual 3D schematic of the illustration.

Advantages and Disadvantages of Gradient Descent Method

Gradient descent is widely used due to its simplicity and general applicability [14, 15]. However, it also presents limitations that motivate the development of advanced optimization algorithms [16].

Advantages:

1. Simple implementation: easy to code and understand for various optimization problems [14].
2. Low computational cost per iteration: requires only gradient evaluation [15].
3. Applicable to a wide range of differentiable problems (convex and non-convex) [14].
4. Flexible descent directions (steepest descent, Newton, quasi-Newton) [16].
5. Guaranteed convergence under convexity with proper step size [15].
6. Foundation for advanced methods such as conjugate gradient and quasi-Newton [14].

Disadvantages:

1. Slow convergence rate, especially for ill-conditioned problems [14].
2. Sensitive to step size: small steps cause slow progress, large steps cause divergence [15].
3. Sensitive to initial guess: may converge to local minima in non-convex problems [16].
4. Line search (optimal step size) may be computationally expensive [14].
5. Does not exploit curvature information (no second derivatives used) [15].
6. Limited performance in high-dimensional problems [16].

3.1.2 Gradient Method

Principle

The **gradient method** is an iterative descent technique used to find the minimum of a given function. The most natural way to determine a descent direction is by using the derivative. Thus, to find the direction of descent, we use the opposite direction of the derivative. In this approach, we take:

$$\Delta X = -\nabla f(X)^T.$$

The corresponding iterative relation for the gradient method is:

$$f(X_{k+1}) \approx f(X_k) + \rho_k \nabla f(X_k) \Delta X_k.$$

If ρ_k is constant, this method is called the *fixed step gradient method*. When ρ_k varies, it is called the *variable step gradient method*.

Algorithm 2 Gradient Algorithm

1: **Step 0 (Initialization):** Set $k = 0$. Choose initial point $x_0 \in \mathbb{R}^n$ and step size $\rho_0 > 0$.
2: **Step 1 (Iteration):**

$$x_{k+1} = x_k - \rho_k \nabla J(x_k).$$

3: **Step 2 (Stopping criterion):** If

$$\|x_{k+1} - x_k\| < \epsilon,$$

stop the algorithm; otherwise, set $k = k + 1$ and return to Step 1.

Throughout what follows, ϵ is a given small positive real number representing the desired precision. This method has the advantage of being very easy to implement. Unfortunately, its convergence conditions are quite restrictive (mainly strict convexity), and in general, the method is rather slow.

Theorem 3.1.1 (Convergence Criterion [17])**.** *Let J be a C^1 function from $\mathbb{R}^n$ to $\mathbb{R}$, coercive and strictly convex. Suppose there exists a strictly positive constant M such that*

$$J(x) - J(y) \geq M\|x - y\|^2, \quad \forall x, y \in \mathbb{R}^n.$$

Then, if the step size ρ_k is chosen in an interval $[\alpha_1, \alpha_2]$ such that

$$0 < \alpha_1 < \alpha_2 < \frac{2}{M},$$

the gradient method converges to the minimum of J.

Remark. When J satisfies the above condition, the constant step gradient algorithm can also be interpreted as the method of successive approximations applied to finding the fixed point of the function

$$S(x) = x - \rho \nabla J(x),$$

where $\nabla J(x) = 0$. Indeed, S is Lipschitz continuous with constant $(1 - \rho M)$. Thus, it is a strict contraction if the contraction factor is in $(0, 1)$. Therefore, it has a unique fixed point, and the convergence is that of a geometric series with ratio $(1 - \rho M)$. This is optimal for $\rho = \frac{2}{M}$.

In practice, the *constant step gradient method* is most often used. However, the step can vary at each iteration, leading to the *variable step gradient method.*

The *optimal step gradient method* chooses the step size that minimizes the cost function along the chosen descent direction. More precisely, step 2 becomes:

$$x_{k+1} = x_k - \rho_k \nabla J(x_k),$$

where ρ_k minimizes

$$\rho \mapsto J(x_k - \rho \nabla J(x_k)).$$

In practice, the exact minimum is not computed, and ρ_k is determined by performing a line search according to a specific rule, for example.

Algorithm 3 Wolfe Line Search Rule

Step 1 (Initialization): Set $\rho = 1$, $\rho^+ = +\infty$, $\rho^- = 0$. Choose $0 < \alpha_1 < \alpha_2 < 1$.
Step 2 (Wolfe conditions test): If

$$\phi(\rho) \leq \phi(0) + \alpha_1 \rho \phi'(0),$$

and

$$\phi'(\rho) \geq \alpha_2 \phi'(0),$$

then stop: set $\rho_k = \rho$.
Step 3 (Otherwise):
if $\phi(\rho) > \phi(0) + \alpha_1 \rho \phi'(0)$ **then**
  Set $\rho^+ = \rho$.
else if $\phi(\rho) \leq \phi(0) + \alpha_1 \rho \phi'(0)$ **and** $\phi'(\rho) < \alpha_2 \phi'(0)$ **then**
  Set $\rho^- = \rho$.
end if Go to Step 4.
Step 4 (Choose a new ρ):
if $\rho^+ = +\infty$ **then**
  Set $\rho = 2\rho$.
else
  Set $\rho = (\rho^- + \rho^+)/2$.
end if Return to Step 2.

Example 3.1. The conditions of the theorem may seem complicated, so we provide an example. Let J be the function from $\mathbb{R}^n$ to $\mathbb{R}$, already mentioned several times (because it plays an important role), defined by:

$$J(x) = \frac{1}{2}(Ax, x) - (b, x),$$

where A is a square, symmetric, and positive definite matrix, and $b \in \mathbb{R}^n$. This function J satisfies the hypotheses of the above theorem with m and M being the smallest and largest eigenvalues of A (respectively).

Remark 3.3. The notion of ellipticity is very important because it determines the convergence of most algorithms that will be described later. However, the convergence conditions we provide are always sufficient conditions. The algorithm converges if they are satisfied, but it may still converge even if they are not...

In practice, m and M are not computed. To find the convergence interval of ρ, several tests are performed for different values. Non-convergence generally results in either an explosion of the solution (clearly tending to $+\infty$) or oscillations (periodic or not) that prevent the sequence of iterates from converging to a value.

Gradient Method with Optimal Step Size

The method is as follows:

$$d_k = -\nabla f(x_k), \quad x_{k+1} = x_k + \rho_k d_k,$$

where ρ_k is chosen by the minimization rule. It consists in choosing, at each iteration, ρ_k as the optimal solution of the one-dimensional minimization problem of f along the half-line defined by the point x_k and the direction d_k. Therefore, ρ_k is chosen such that:

$$f(x_k + \rho_k d_k) = \min_{\rho \in \mathbb{R}, \rho > 0} f(x_k + \rho d_k),$$

assuming that such a minimum exists.

Remark 3.0.1. We perform iteration (3.3) in the case where $\nabla f(x_k) \neq 0$, so the minimization problem (3.4) makes sense. We have the following result:

Theorem 3.1.2 (3.0.1)**.** *Let f be a C^1 function from $\mathbb{R}^n$ to $\mathbb{R}$, coercive and strictly convex. We suppose that there exists a constant $M > 0$ such that*

$$\forall (x, y) \in \mathbb{R}^n \times \mathbb{R}^n, \ \|\nabla f(x) - \nabla f(y)\| \leq M \|x - y\|.$$

Then, if we choose the step ρ_k in an interval $[\beta_1, \beta_2]$ with $0 < \beta_1 < \beta_2 < \frac{2}{M}$, the gradient method converges towards the minimum of f.

Proof. The function f admits a unique minimum x_0 on $\mathbb{R}^n$ characterized by $\nabla f(x_0) = 0$ since f is strictly convex. Let us show that the sequence (x_k) generated by the algorithm converges to x_0. We have:

$$f(y) = f(x) + (\nabla f(x), y - x) + \int_0^1 (\nabla f(x + t(y - x)) - \nabla f(x), y - x)\, dt.$$

Applying this relation to $y = x_{k+1}$, $x = x_k$, we obtain

$$f(x_{k+1}) = f(x_k) + (\nabla f(x_k), x_{k+1} - x_k) + \int_0^1 (\nabla f(x_k + t(x_{k+1} - x_k)) - \nabla f(x_k), x_{k+1} - x_k)\, dt.$$

Since $x_{k+1} = x_k - \rho_k \nabla f(x_k)$, we get

$$f(x_{k+1}) - f(x_k) \leq -\frac{1}{\rho_k} \|x_{k+1} - x_k\|^2 + \int_0^1 \|\nabla f(x_k + t(x_{k+1} - x_k)) - \nabla f(x_k)\| \cdot \|x_{k+1} - x_k\|\, dt.$$

Using the Lipschitz condition, we get:

$$\leq -\frac{1}{\rho_k} \|x_{k+1} - x_k\|^2 + \frac{M}{2} \|x_{k+1} - x_k\|^2 = \left(\frac{M}{2} - \frac{1}{\rho_k} \right) \|x_{k+1} - x_k\|^2.$$

If we choose the step ρ_k in an interval $[\beta_1, \beta_2]$ with $0 < \beta_1 < \beta_2 < \frac{2}{M}$, we obtain:

$$f(x_{k+1}) - f(x_k) \leq \left(\frac{M}{2} - \frac{1}{\rho_k} \right) \|x_{k+1} - x_k\|^2.$$

The sequence $f(x_k)$ is therefore strictly decreasing and bounded below because

$$f(x_k) \geq f(x_0), \quad \forall k.$$

Hence, it is convergent. This implies on the one hand that $(f(x_{k+1}) - f(x_k))$ tends to 0 and on the other hand that the sequence (x_k) is bounded (since f is coercive). We can thus extract a subsequence converging to x. Moreover,

$$\|x_{k+1} - x_k\|^2 \leq \left(\frac{M}{2} - \frac{1}{\rho_k}\right)^{-1} (f(x_k) - f(x_{k+1})).$$

Therefore, $\|x_{k+1} - x_k\| \to 0$. Consequently,

$$\nabla f(x_k) = \frac{x_k - x_{k+1}}{\rho_k} \to 0.$$

By continuity of ∇f, we deduce that x is the unique minimum x_0 of f. Since this is true for any accumulation point of the sequence (x_k), it proves that the entire sequence (x_k) converges towards x_0.

Fixed Step Gradient Method

We can use a step fixed a priori $\rho > 0$, for all k, and we then obtain the simple gradient method:

$$d_k = -\nabla f(x_k), \quad x_{k+1} = x_k + \rho d_k.$$

For $f \in C^1$, this method converges if ρ is chosen sufficiently small.

Choice of step:

- A well-chosen step gives results similar to those obtained by the steepest descent.
- A smaller step reduces the zigzags of the iterates but significantly increases the number of iterations.
- An excessively large step makes the method diverge.

Particular Case: Quadratic Functions

In this paragraph we assume that $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + c,$$

where $A \in M_n(\mathbb{R})$ is a symmetric positive definite (SPD) matrix, $b \in \mathbb{R}^n$, and $c \in \mathbb{R}$, so it is a quadratic form associated with an SPD matrix.

We must calculate $\rho_k \in \mathbb{R}$ which minimizes the function $g : \mathbb{R} \to \mathbb{R}$ given by

$$g(\rho) = f(x_k - \rho \nabla f(x_k)).$$

Then ρ_k necessarily satisfies

$$g'(\rho_k) = 0.$$

A simple calculation gives

$$g'(\rho) = -\langle \nabla f(x_k - \rho \nabla f(x_k)), \nabla f(x_k)\rangle.$$

That is, since $\nabla f(x_k) = Ax_k - b$,

$$g'(\rho) = -\|Ax_k - b\|^2 + \rho\langle A(Ax_k - b), Ax_k - b\rangle.$$

We thus obtain

$$\rho_k = \frac{\|Ax_k - b\|^2}{\langle A(Ax_k - b), Ax_k - b\rangle}.$$

Note that $\langle A(Ax_k - b), Ax_k - b\rangle > 0$ (because A is SPD and $Ax_k - b = \nabla f(x_k) \neq 0$). Therefore, the optimal step gradient method in the quadratic case is:

$$x_{k+1} = x_k - \rho_k(Ax_k - b),$$

with ρ_k given by (3.6), valid only for $Ax_k - b \neq 0$.

Example

Let $f(X)$ be a quadratic cost function in $\mathbb{R}^2$ given by:

$$f(X) = x^2 + 3y^2,$$

with

$$X = \begin{pmatrix} x \\ y \end{pmatrix}.$$

It is evident that the origin is the point minimizing this function. This function is illustrated in Figures .... and .....

Resolution with a fixed step

In this case, we search for an optimal step ρ which minimizes a certain function. We have:

$$\nabla f(X) = [2x, 6y],$$

$$\xi(\rho) = X - \rho\nabla f(X)^T = \begin{pmatrix} (1-2\rho)x \\ (1-6\rho)y \end{pmatrix}. \tag{2.7}$$

To obtain the parameter ρ, we consider the function $g(\rho)$ that we must minimize with respect to ρ:

$$g(\rho) = f(\xi(\rho)) = (1-2\rho)^2x^2 + (1-6\rho)^2y^2.$$

The derivative of this function with respect to the variable ρ gives:

$$g'(\rho) = -4(1-2\rho)x^2 - 36(1-6\rho)y^2.$$

The parameter ρ that minimizes the function $g(\rho)$ is the one satisfying $g'(\rho) = 0$, thus:

$$\rho = \frac{x^2 + 9y^2}{2x^2 + 54y^2}.$$

Therefore, the iterative equation to solve this problem is:

- Initialization:
$$X = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}.$$
- Repeat:
  1. $\Delta X \leftarrow -\begin{pmatrix} 2x \\ 6y \end{pmatrix}$.
  2. $\rho = \frac{x^2+9y^2}{2x^2+54y^2}$.
  3. $X \leftarrow X + \rho\Delta X$.
- Stop when $\|\Delta X\|_2^2 \leq 10^{-3}$.

Resolution with an iterative step

When we choose an iterative step, the second step in the algorithm is replaced by the following loop:

Let $\alpha = 0.25$, $\beta = 0.5$, $\rho = 1$, then:

For a given direction
$$\Delta X = -\begin{pmatrix} 2X(1) \\ 6X(2) \end{pmatrix},$$
do:

1. $Y = X + \rho\Delta X$,
2. $f(Y) = Y(1)^2 + 3Y(2)^2$,
3. $\nabla f(X) = [2X(1), 6X(2)]$,
4. if $f(Y) < f(X) + \alpha\rho\nabla f(X)\Delta X$, end,
5. else $\rho = \beta\rho$,
6. end.

3.1.3 Newton's Method

The Newton algorithm in optimization is a direct application of Newton's method for solving equations of the form $J(x) = 0$. In unconstrained optimization, the Newton algorithm seeks solutions of the equation $J(x) = 0$. This is a nonlinear equation (or rather a system of nonlinear equations) in $\mathbb{R}^n$, and we will use Newton's method to solve it. However, we will only obtain the critical points of J: it is then necessary to check that they are indeed minima.

Here $f = J$ is indeed a function from $\mathbb{R}^n$ to $\mathbb{R}^n$. The derivative of f is none other than the Hessian matrix of J: $H(x) = D^2J(x)$.

The Newton method is thus written as:

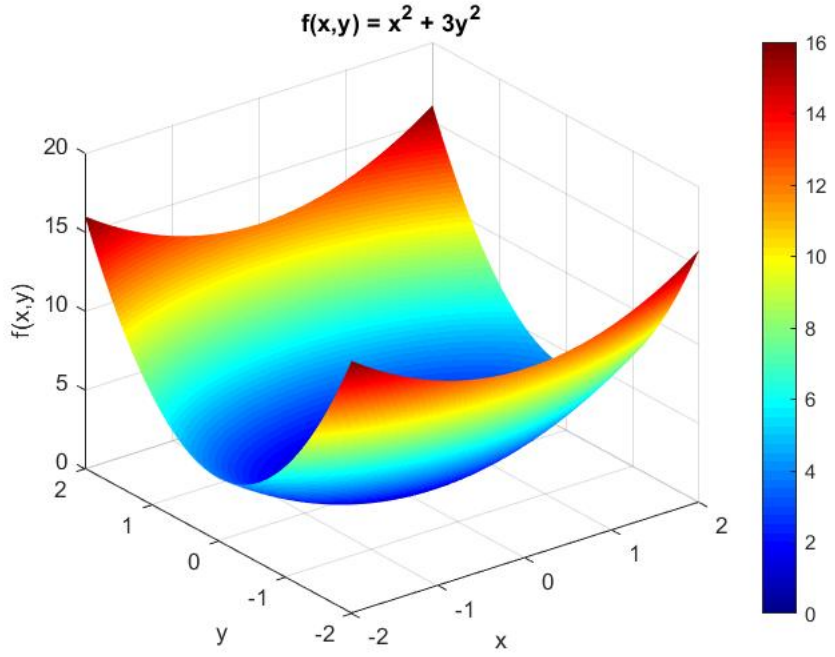

Figure 3.3: Shape of the function $f(x, y) = x^2 + 3y^2$.

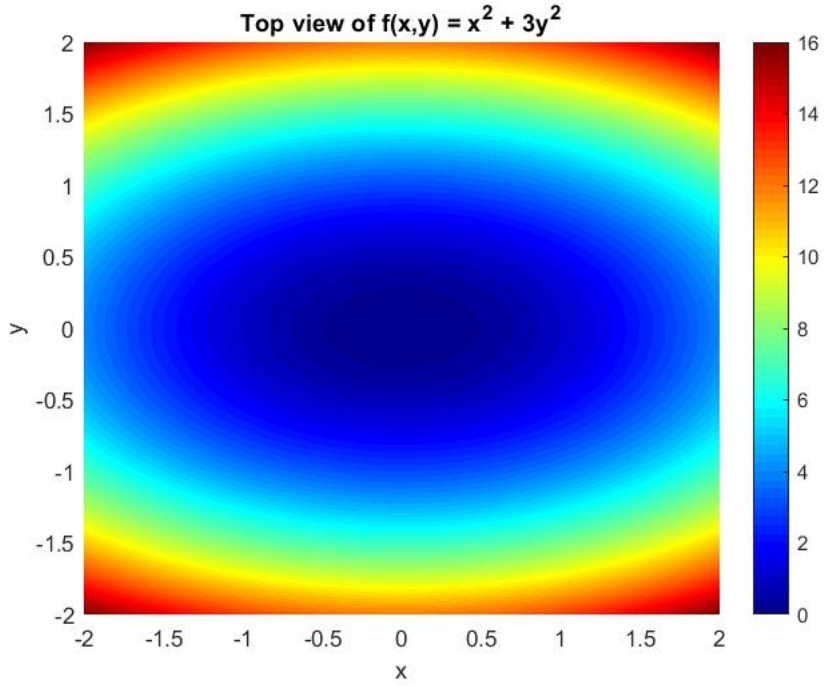

Figure 3.4: Top view of the function $f(x, y) = x^2 + 3y^2$.

Algorithm 4 Newton's Algorithm in $\mathbb{R}^n$

1: **Initialization:** $k = 0$; choose $x_0 \in \mathbb{R}^n$ in a neighborhood of x.
2: **Iteration** k**:**

$$x_{k+1} = x_k - [H(x_k)]^{-1} J(x_k).$$

3: **Stopping criterion:** if $\|x_{k+1} - x_k\| < \varepsilon$, STOP; otherwise, set $k = k + 1$ and return to step 2.

Step 2 of the method amounts to solving the following linear system:

$$H_k \Delta_k = J(x_k)$$

where $H_k = H(x_k)$, then setting

$$x_{k+1} = x_k - \Delta_k.$$

Newton's Method: Advanced 3D Example

Problem. Minimize:

$$f(x, y, z) = x^2 + xy + y^2 + yz + z^2 + xz - 6x - 9y - 15z.$$

Gradient.

$$\nabla f(x, y, z) = \begin{bmatrix} 2x + y + z - 6 \\ x + 2y + z - 9 \\ x + y + 2z - 15 \end{bmatrix}.$$

Hessian.

$$H(x, y, z) = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}.$$

Inverse Hessian.

$$H^{-1} = \frac{1}{4} \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}.$$

First iteration with $x_0 = [0, 0, 0]^T$.

$$\nabla f(x_0) = \begin{bmatrix} -6 \\ -9 \\ -15 \end{bmatrix}.$$

$$H^{-1} \nabla f(x_0) = \frac{1}{4} \begin{bmatrix} 6 \\ -6 \\ -30 \end{bmatrix} = \begin{bmatrix} 1.5 \\ -1.5 \\ -7.5 \end{bmatrix}.$$

$$x_1 = x_0 - H^{-1} \nabla f(x_0) = \begin{bmatrix} -1.5 \\ 1.5 \\ 7.5 \end{bmatrix}.$$

The algorithm converges in one iteration.

Minimum point: $(-1.5, 1.5, 7.5)$

Minimum value calculation.

$$f(-1.5, 1.5, 7.5) = -58.5.$$

Minimum value: $f(-1.5, 1.5, 7.5) = -58.5$.

Graphical Illustration

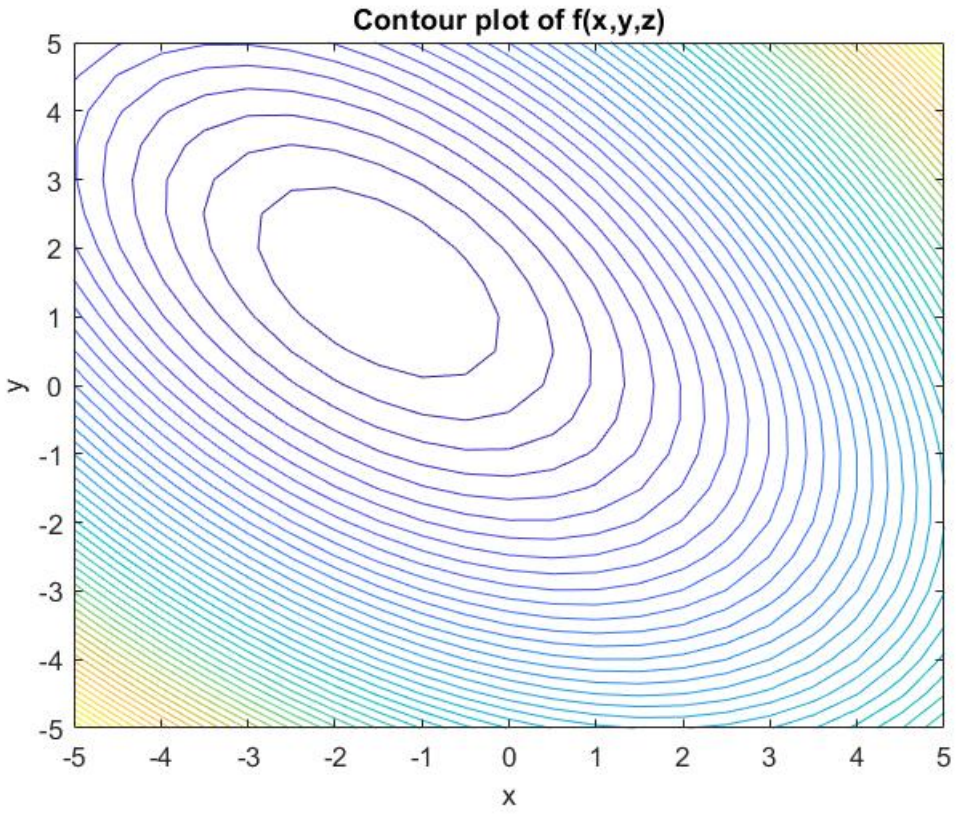

Figure 3.5: 3D surface plot of $f(x, y, z)$.

3.1.3.1 Disadvantages of Newton's Method

1. This method works very well for small dimensions ($1 \leq n \leq 10$) when it is easy to compute $H(x)$ and $H(x)^{-1}$. However, this calculation requires more numerous and costly iterations for large-scale problems.

2. Since $x_{k+1} = x_k - H(x_k)^{-1}\nabla f(x_k)$, the point x_{k+1} is not always well-defined, i.e., it is possible that $H(x_k)^{-1}$ does not exist (this typically occurs when the method reaches a region where f is linear, and thus its second partial derivatives are zero). Even if it exists, the direction $d_k = -H(x_k)^{-1}\nabla f(x_k)$ is not always a descent direction (if $H(x_k)$ is positive definite, then d_k is a descent direction).

3. The major disadvantage of the method is its sensitivity to the choice of the starting point x_0: if this point is poorly chosen (too far from the solution), the method can either diverge or converge to another solution. To choose the starting point x_0 sufficiently close to x^*, one tries to approach x^* by using a gradient-type method, and then applying Newton's method.

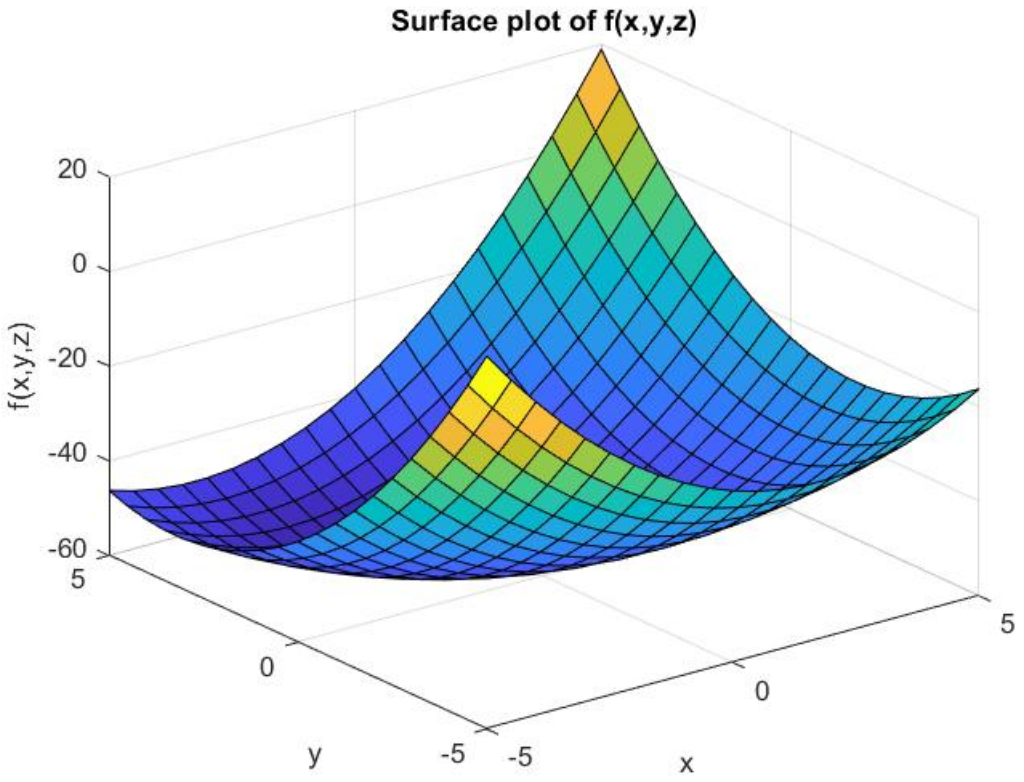

Figure 3.6: Contour (level) plot of $f(x, y, z)$ showing the minimum.

3.1.3.2 Quasi-Newton Method

Quasi-Newton methods are developed for optimization to overcome the drawbacks of Newton's method:

1. They maintain the speed of Newton's method.
2. They avoid the (costly) computation of the matrix $[H(x_k)]$ at each iteration.
3. They are more robust with respect to the starting point. There are methods called "trust region" methods that aim to make the method robust (i.e., not very sensitive) to x_0.
4. $(H(x))^{-1}$ is not necessarily known, it can be very expensive to compute, and $H(x_k)$ can be very difficult to invert. Thus, $(H(x_k))^{-1}$ is replaced by a matrix D_k, possibly constant, which is supposed to approximate $H(x_k)$ or its inverse. Sometimes, even $(H(x_k))^{-1}\nabla f(x_k)$ is replaced by an easily computable vector y_k.

Therefore, the algorithm of this method is given as follows:

Algorithm 5 Quasi-Newton Algorithm

1: **Initialization:** $k = 0$; choose x_0, α_0, ϵ.
2: **Iteration** k**:**

$$x_{k+1} = x_k - \alpha_k D_k \nabla f(x_k).$$

3: **Stopping criterion:** if $\|x_{k+1} - x_k\| < \epsilon$, STOP; otherwise, set $k = k + 1$ and return to step 2.

Remark 3.0.3. In this algorithm, we use an approximation D_k of $(H(x_k))^{-1}$ and then find α_k (by a line search) which minimizes the function $\varphi(\alpha) = f(x_k + \alpha_k d_k)$.

Problem

We consider the following difficult optimization problem in $\mathbb{R}^2$:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 + e^{x-y}.$$

We apply the **BFGS Quasi-Newton method** starting from $x_0 = [1; 2]$ with identity as initial inverse Hessian approximation.

Detailed Solution

Let $x_k = [x_k; y_k]$ at iteration k.

- **Gradient:**

$$\nabla f(x, y) = \begin{bmatrix} 4x(x^2 + y - 11) + 2(x + y^2 - 7) + e^{x-y} \\ 2(x^2 + y - 11) + 4y(x + y^2 - 7) - e^{x-y} \end{bmatrix}.$$

- **Update:**

$$x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k),$$

where H_k is the inverse Hessian approximation, updated via BFGS formula.

Algorithm

Algorithm 6 Quasi-Newton (BFGS) Algorithm

1: **Initialization:** $k = 0$; choose $x_0 \in \mathbb{R}^n$, set $H_0 = I$, tolerance $\epsilon = 10^{-5}$.
2: **Iteration** k**:**
3: Compute the descent direction:

$$d_k = -H_k \nabla f(x_k).$$

4: Perform line search to find α_k that minimizes $f(x_k + \alpha_k d_k)$.
5: Update:

$$x_{k+1} = x_k + \alpha_k d_k.$$

6: Compute:

$$s_k = x_{k+1} - x_k, \quad y_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

7: Update the inverse Hessian approximation H_k using the BFGS formula:

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

8: **Stopping criterion:** If $\|\nabla f(x_{k+1})\| < \epsilon$, STOP; otherwise, set $k = k + 1$ and return to step 2.

MATLAB Code: Quasi-Newton BFGS Advanced

Below is the MATLAB program implementing the Quasi-Newton BFGS method for:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 + e^{x-y}.$$

Program:

```
function quasi_newton_bfgs_advanced()
    % Function definition
    f = @(x,y) (x.^2 + y - 11).^2 + (x + y.^2 - 7).^2 + exp(x - y);

    % Gradient
    grad = @(x) [
        4*x(1)*(x(1)^2 + x(2) - 11) + 2*(x(1) + x(2)^2 - 7) + exp(x(1)-x
            (2));
        2*(x(1)^2 + x(2) - 11) + 4*x(2)*(x(1) + x(2)^2 - 7) - exp(x(1)-x
            (2))
    ];

    % Initialization
    xk = [1; 2];
    Hk = eye(2);
    epsilon = 1e-5;
```

```
    maxIter = 1000;
    path = xk';

    for k = 1:maxIter
        gk = grad(xk);
        if norm(gk) < epsilon
            break;
        end

        dk = -Hk * gk;

        % Backtracking line search
        alpha = 1;
        rho = 0.5;
        c = 1e-4;
        while f(xk(1)+alpha*dk(1), xk(2)+alpha*dk(2)) > ...
              f(xk(1),xk(2)) + c*alpha*gk'*dk
            alpha = rho * alpha;
        end

        % Update
        xk_new = xk + alpha * dk;
        sk = xk_new - xk;
        yk = grad(xk_new) - gk;

        % BFGS update formula
        Hk = (eye(2) - sk*yk'/(yk'*sk)) * Hk * ...
             (eye(2) - yk*sk'/(yk'*sk)) + (sk*sk')/(yk'*sk);
        xk = xk_new;
        path = [path; xk'];
    end

    % Plotting
    [X,Y] = meshgrid(-5:0.05:5, -5:0.05:5);
    Z = (X.^2 + Y - 11).^2 + (X + Y.^2 - 7).^2 + exp(X - Y);

    figure;
    contour(X,Y,Z,logspace(0,5,40)); hold on;
    plot(path(:,1), path(:,2), 'r.-', 'LineWidth', 1.5);
    xlabel('x'); ylabel('y');
    title('BFGS Optimization Path on Modified Himmelblau + Exponential')
        ;
    colorbar;

    % Save as PNG
    saveas(gcf, 'bfgs_himmelblau.png');
end
```

Listing 3.2: BFGS Optimization on Modified Himmelblau + Exponential

Graphical Illustration

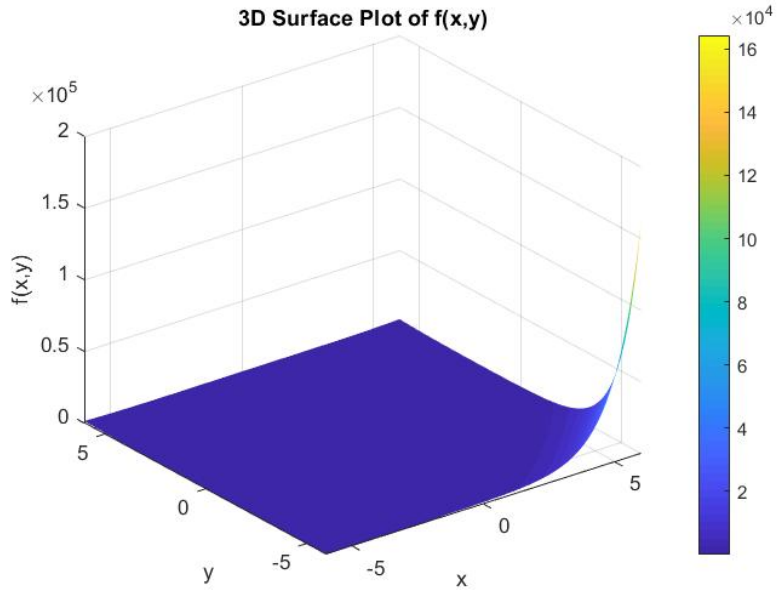

Figure 3.7: Optimization path using BFGS Quasi-Newton on the modified Himmelblau + exponential function.

3.1.4 Conjugate Gradient Method

Conjugate gradient methods are used to solve linear systems whose matrix is symmetric positive definite. They are also used to solve large linear systems. It is an iterative method that converges in a finite number of iterations (at most equal to the dimension of the linear system).

Definition 3.1.1. *Let A be an $n \times n$ symmetric positive definite matrix. Two vectors x and y in $\mathbb{R}^n$ are said to be A-conjugate (or conjugate with respect to A) if they satisfy:*

$$x^T A y = 0.$$

Principle of Conjugate Gradient Method

Let $\{d_0, d_1, ..., d_n\}$ be a family of A-conjugate vectors. We then call *conjugate direction method* any iterative method applied to a strictly convex quadratic function of n variables:

$$f(x) = \frac{1}{2} x^T A x + b^T x + c,$$

with $x \in \mathbb{R}^n$, $A \in \mathbb{M}_{n \times n}$ symmetric positive definite, $b \in \mathbb{R}^n$, and $c \in \mathbb{R}$, leading to the optimum in at most n steps.

They rely on the concept of conjugate directions because successive gradients are orthogonal to each other and to previous directions.

The idea of the method is, given an initial point x_0 in $\mathbb{R}^n$ and n conjugate directions, to iteratively construct mutually conjugate directions $d_1, ..., d_k$. We define the following scheme:

$$x_{k+1} = x_k + \rho_k d_k,$$

where ρ_k is the scalar minimizing $f(x)$ along the direction $x_k + \rho_k d_k$ and is defined by:

$$\rho_k = -\frac{r_k^T d_k}{d_k^T A d_k},$$

$$r_k = -\nabla f(x_k) = b - Ax_k.$$

The algorithm below solves $Ax = b$, where A is a real, symmetric, positive definite matrix. The input vector x_0 can be an approximation of the initial solution or zero.

Algorithm 7 Conjugate Gradient Method

1: **Step 0 (Initialization):** Set $k = 0$.
2: Choose $x_0 \in \mathbb{R}^n$; compute $r_0 = -\nabla f(x_0) = b - Ax_0$, set $p_0 = r_0$.
3: **Step 1 (Compute step size):**

$$\alpha_k = -\frac{r_k^T r_k}{p_k^T A p_k}.$$

4: **Step 2 (Update):**

$$x_{k+1} = x_k + \rho_k p_k, \quad r_{k+1} = r_k - \alpha_k A p_k.$$

5: **Step 3 (Stopping criterion):** If r_{k+1} is sufficiently small, STOP the algorithm.
6: **Step 4 (Direction update):**

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \quad p_{k+1} = r_{k+1} + \beta_k p_k.$$

7: Replace k by $k + 1$ and return to Step 1.

Theorem 3.1.3. *If $f : \mathbb{R}^n \to \mathbb{R}$ is quadratic and elliptic, the conjugate gradient method converges in at most n iterations, where n is the order of A.*

Example: Conjugate Gradient Method (Detailed)

Problem. Minimize:

$$f(x, y) = 4x^2 + xy + 3y^2 - 8x - 9y.$$

This can be written in quadratic form as:

$$f(X) = \frac{1}{2}X^T AX - b^T X$$

where

$$A = \begin{bmatrix} 8 & 1 \\ 1 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 8 \\ 9 \end{bmatrix}.$$

Set $x_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.
Compute:

$$r_0 = b - Ax_0 = b = \begin{bmatrix} 8 \\ 9 \end{bmatrix}, \quad p_0 = r_0.$$

Compute

$$\alpha_0 = \frac{r_0^T r_0}{p_0^T A p_0}.$$

Calculate:

$$r_0^T r_0 = 8^2 + 9^2 = 64 + 81 = 145.$$

$$Ap_0 = \begin{bmatrix} 8 & 1 \\ 1 & 6 \end{bmatrix} \begin{bmatrix} 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 73 \\ 62 \end{bmatrix}.$$

Then,

$$p_0^T A p_0 = [8, 9] \cdot [73, 62] = 8 * 73 + 9 * 62 = 584 + 558 = 1142.$$

Hence,

$$\alpha_0 = \frac{145}{1142} \approx 0.127.$$

$$x_1 = x_0 + \alpha_0 p_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.127 \begin{bmatrix} 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1.016 \\ 1.143 \end{bmatrix}.$$

$$r_1 = r_0 - \alpha_0 A p_0 = \begin{bmatrix} 8 \\ 9 \end{bmatrix} - 0.127 \begin{bmatrix} 73 \\ 62 \end{bmatrix} = \begin{bmatrix} -1.271 \\ 1.126 \end{bmatrix}.$$

$$\beta_0 = \frac{r_1^T r_1}{r_0^T r_0}.$$

Calculate:

$$r_1^T r_1 = (-1.271)^2 + (1.126)^2 = 1.615 + 1.268 = 2.883.$$

Hence,

$$\beta_0 = \frac{2.883}{145} \approx 0.0199.$$

$$p_1 = r_1 + \beta_0 p_0 = \begin{bmatrix} -1.271 \\ 1.126 \end{bmatrix} + 0.0199 \begin{bmatrix} 8 \\ 9 \end{bmatrix} = \begin{bmatrix} -1.112 \\ 1.305 \end{bmatrix}.$$

Continue with the same process:

$$\alpha_1 = \frac{r_1^T r_1}{p_1^T A p_1}, \quad x_2 = x_1 + \alpha_1 p_1, \quad r_2 = r_1 - \alpha_1 A p_1.$$

The method converges in at most $n = 2$ iterations for a quadratic problem.

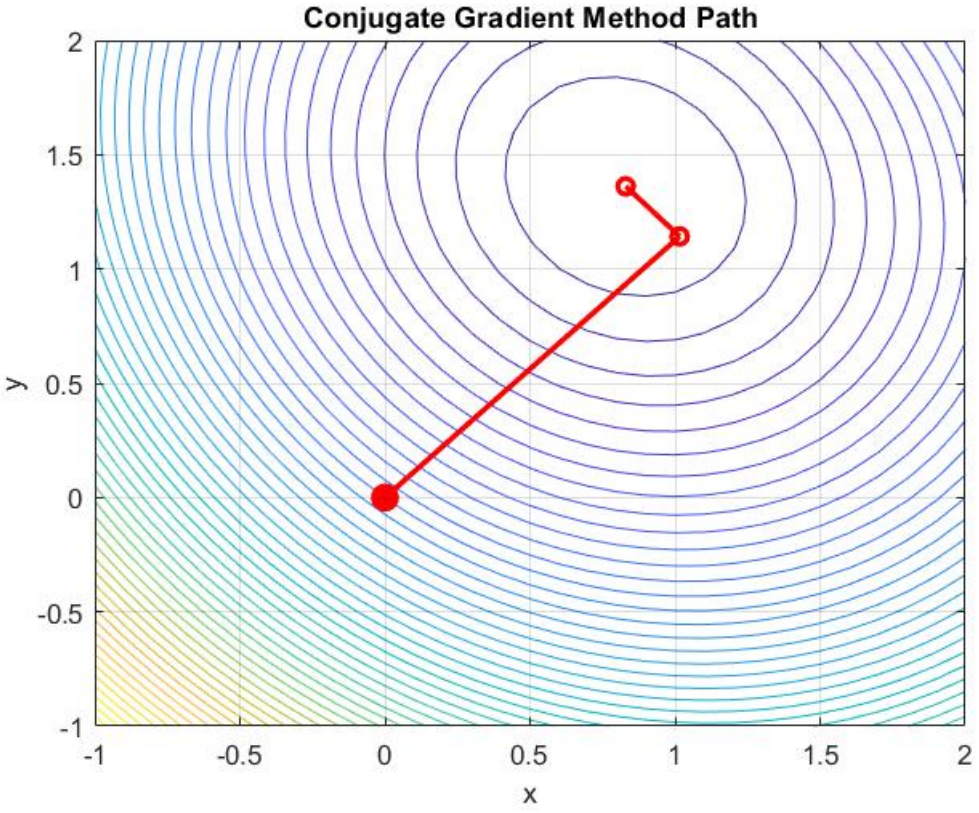

Figure 3.8: Conjugate Gradient method optimization path for $f(x, y) = 4x^2 + xy + 3y^2 - 8x - 9y$.

MATLAB Implementation: Conjugate Gradient Method Example

Problem

Solve: $f(x, y) = 4x^2 + xy + 3y^2 - 8x - 9y$ using the Conjugate Gradient Method.

MATLAB Code

```
% Conjugate Gradient Method Example

% Define A and b
A = [8 1; 1 6];
b = [8; 9];

% Initial guess
x = [0; 0];

% Compute initial residual
r = b - A*x;
p = r;
k = 0;

while norm(r) > 1e-6
    Ap = A*p;
    alpha = (r'*r) / (p'*Ap);
    x_new = x + alpha*p;
    r_new = r - alpha*Ap;

    if norm(r_new) < 1e-6
        break;
    end

    beta = (r_new'*r_new) / (r'*r);
    p = r_new + beta*p;

    % Update for next iteration
    x = x_new;
    r = r_new;
    k = k + 1;

    % Display each iteration
    fprintf('Iteration %d: x = [%f; %f], residual norm = %e\n', ...
            k, x(1), x(2), norm(r));
end

% Final solution
disp('Optimal x:')
disp(x)
```

Expected Output

Optimal solution $x^* \approx [0.967, 1.401]$ minimizing the quadratic function.

3.1.5 Relaxation Method

The last method we present allows reducing a minimization problem in $\mathbb{R}^n$ to the successive resolution of n minimization problems in $\mathbb{R}$ (at each iteration).

We seek to minimize $J : \mathbb{R}^n \to \mathbb{R}$; let us set

$$X = (x_1, x_2, \ldots, x_n).$$

The principle of the method is as follows:

Given an iterate X^k with coordinates $(x_1^k, x_2^k, \ldots, x_n^k)$, we fix all components except the first and we minimize over the first:

$$\min_{x \in \mathbb{R}} J(x, x_2^k, x_3^k, \ldots, x_n^k).$$

Thus, we obtain the first coordinate of the next iterate X^{k+1}, which we denote x_1^{k+1}. For this minimization in $\mathbb{R}$, one can use for example Newton's method in $\mathbb{R}$.

Then, we repeat by fixing the first coordinate to x_1^{k+1} and the last $n-2$ as before. We minimize over the second coordinate, and so on.

The resulting algorithm is as follows:

Algorithm 8 Successive Relaxation Method – One Iteration

1: **Step 0 (Initialization):** At the beginning of iteration k, choose $X^k \in \mathbb{R}^n$.
2: **Step 1 (Coordinate-wise minimization):**
3: For each coordinate $i = 1$ to n:
4: Minimize J with respect to x_i while keeping all other components fixed:

$$x_i^{k+1} = \min_x J(x_1^{k+1}, x_2^{k+1}, \ldots, x_{i-1}^{k+1}, x, x_{i+1}^k, \ldots, x_n^k).$$

5: **Step 2 (Stopping test):** If $\|X^{k+1} - X^k\| < \epsilon$, STOP; otherwise, set $k = k+1$ and return to Step 1.

Example: Successive Relaxation Method

Minimize:

$$f(x, y, z) = x^4 + y^4 + z^4 - 4xy + z^2 - x + y + 3.$$

Step-by-step solution:

1. Initialize: $x_0 = 0$, $y_0 = 0$, $z_0 = 0$.
2. Minimize with respect to x (fixing y, z):
$$x_{k+1} = x_k - \frac{4x_k^3 - 4y_k - 1}{12x_k^2}.$$
3. Minimize with respect to y (fixing x, z):
$$y_{k+1} = y_k - \frac{4y_k^3 - 4x_k + 1}{12y_k^2}.$$
4. Minimize with respect to z:
$$z_{k+1} = 0.$$
5. Check convergence:
$$\sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2 + (z_{k+1} - z_k)^2} < \epsilon.$$

Algorithm 9 Successive Relaxation for 3-variables

1: **Initialization:** Choose x_0, y_0, z_0, tolerance ϵ, set $k = 0$.
2: **repeat**
3: Update $x_{k+1} = \frac{2-y_k-z_k}{2}$.
4: Update $y_{k+1} = \frac{-x_{k+1}-z_k-3}{8}$.
5: Update $z_{k+1} = \frac{1-x_{k+1}-y_{k+1}}{18}$.
6: $k = k + 1$.
7: **until** $|x_{k+1} - x_k| + |y_{k+1} - y_k| + |z_{k+1} - z_k| < \epsilon$
8: **Output:** Optimal (x^*, y^*, z^*).

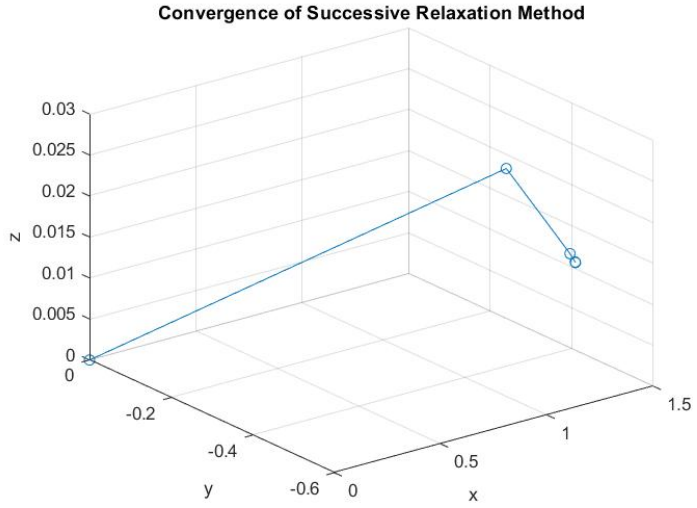

Figure 3.9: Convergence trajectory of Successive Relaxation in 3D space.