

```

In [5]: import numpy as np
        from scipy.optimize import minimize

        # Definierte Datenpunkte und Klassen
        X = np.array([[1, 1], [-1, 1], [2, 1], [1, -2], [-2, 1]]) # A, B, C, D, E
        y = np.array([1, 1, -1, -1, -1]) # Labels: +1 für A und B, -1 für C, D und E

        # Polynomialer Kernel vom Grad 3:  $K(x_i, x_j) = (x_i \cdot x_j + 1)^3$ 
        def polynomial_kernel(x_i, x_j, degree=3):
            return (np.dot(x_i, x_j) + 1) ** degree

        # Kernel-Matrix berechnen
        def compute_kernel_matrix(X, degree=3):
            n_samples = X.shape[0]
            K = np.zeros((n_samples, n_samples))
            for i in range(n_samples):
                for j in range(n_samples):
                    K[i, j] = polynomial_kernel(X[i], X[j], degree)
            return K

        # Berechne die Kernel-Matrix für die Datenpunkte
        K = compute_kernel_matrix(X, degree=3)
        print("Kernel-Matrix:\n", K)

        # Lagrange-Funktion:  $L = \sum(\alpha_i) - \frac{1}{2} \cdot \sum(\alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(x_i, x_j))$ 
        def lagrange_function(alpha, y, K):
            return np.sum(alpha) - 0.5 * np.sum(alpha * alpha[:, None] * y * y[:, None] * K)

        # Randbedingungen:  $\sum(\alpha_i \cdot y_i) = 0$ 
        def equality_constraint(alpha, y):
            return np.dot(alpha, y)

        # Bounds für die Multiplikatoren alpha ( $\alpha_i \geq 0$ )
        bounds = [(0, None) for _ in range(X.shape[0])]

        # Anfangswerte für alpha
        initial_alpha = np.zeros(X.shape[0])

        # Optimierung der Lagrange-Funktion
        result = minimize(lambda alpha: -lagrange_function(alpha, y, K), initial_alpha,
                          constraints={'type': 'eq', 'fun': lambda alpha: equality_constraint(alpha, y)},
                          bounds=bounds)

        # Optimierte Lagrange-Multiplikatoren
        optimal_alpha = result.x
        print("Optimierte Lagrange-Multiplikatoren:\n", optimal_alpha)

        # Ausgabe der Lagrange-Funktion
        lagrange_value = lagrange_function(optimal_alpha, y, K)
        print("Wert der Lagrange-Funktion:\n", lagrange_value)

```

Kernel-Matrix:

```
[[ 27.  1. 64.  0.  0.]  
[  1. 27.  0. -8. 64.]  
[ 64.  0. 216.  1. -8.]  
[  0. -8.  1. 216. -27.]  
[  0. 64. -8. -27. 216.]]
```

Optimierte Lagrange-Multiplikatoren:

```
[0.02864146 0.02861853 0.02104154 0.01346675 0.0227517 ]
```

Wert der Lagrange-Funktion:

```
0.057238979631991434
```