In [1]:

```
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

# Linearer Kernel: K(x_i, x_j) = x_i * x_j
def linear_kernel(x_i, x_j):
    return np.dot(x_i, x_j)

# Kernel-Matrix berechnen
def compute_kernel_matrix(X):
    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] = linear_kernel(X[i], X[j])
    return K

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

# Lagrange-Funktion: L = Sum(alpha_i) - 1/2 * Sum(alpha_i * alpha_j * y_i * y
_j * K(x_i, x_j))
def lagrange_function(alpha, y, K):
    return np.sum(alpha) - 0.5 * np.sum(alpha * alpha[:, None] * y * y[:, Non
e] * K)

# Randbedingungen: Sum(alpha_i * y_i) = 0
def equality_constraint(alpha, y):
    return np.dot(alpha, y)

# Bounds für die Multiplikatoren alpha (alpha_i >= 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_alph
a,
                  constraints={'type': 'eq', 'fun': lambda alpha: equality_co
nstraint(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:
 [[ 2.  0.  3. -1. -1.]
 [ 0.  2. -1. -3.  3.]
 [ 3. -1.  5.  0. -3.]
 [-1. -3.  0.  5. -4.]
 [-1.  3. -3. -4.  5.]]
Optimierte Lagrange-Multiplikatoren:
 [1.10721516e+05 1.10702248e+05 1.10706469e+05 7.31608626e+00
 1.10709978e+05]
Wert der Lagrange-Funktion:
 442426.72160310304
```