增广拉格朗日乘子法

在拉格朗日乘子法的基础上增加一个二次惩罚项。由于拉格朗日法需要求解拉格朗日函数梯度等于0的方程，再高阶问题中需要用迭代法进行求解，这就导致要求两次导，拉格朗日一次牛顿法一次。而ALM则通过引入惩罚项使问题变为凸的，此时就可以利用传统梯度法求解一次了。

优化问题的定义

我们希望最小化一个目标函数，目标函数是一个二次函数，约束条件包括一个等式约束和一个不等式约束：

$$\min_{x_1,x_2} f(x) = x_1^2 + x_2^2$$

约束条件

1. 等式约束：$x_1 + x_2 = 1$
2. 不等式约束：$x_1^2 + x_2^2 \leq 2$

这个问题是一个典型的包含等式约束和不等式约束的优化问题，我们将通过增广拉格朗日乘子法进行求解。

步骤 1：构建拉格朗日函数

拉格朗日乘子法的拉格朗日函数是将目标函数与约束条件结合的方式，通过引入拉格朗日乘子来处理约束。我们需要引入等式约束的拉格朗日乘子 λ 和不等式约束的拉格朗日乘子 μ。

目标函数和约束条件的拉格朗日函数为：

$$\mathcal{L}(x_1, x_2, \lambda, \mu) = x_1^2 + x_2^2 + \lambda(x_1 + x_2 - 1) + \mu(x_1^2 + x_2^2 - 2)$$

步骤 2：构建增广拉格朗日函数

增广拉格朗日乘子法通过引入增广项来增强对约束的惩罚项，从而确保收敛到约束满足的解。增广拉格朗日函数为：

$$\mathcal{L}_a(x_1, x_2, \lambda, \mu, \rho) = x_1^2 + x_2^2 + \lambda(x_1 + x_2 - 1) + \mu(x_1^2 + x_2^2 - 2) + \frac{\rho}{2}\left[(x_1 + x_2 - 1)^2 + (x_1^2 + x_2^2 - 2)^2\right]$$

其中，ρ 是增广项的惩罚系数，通常在优化过程中逐渐增大。

步骤 3：优化过程

我们使用交替最小化的策略来求解增广拉格朗日函数：

1. **固定 λ 和 μ，最小化增广拉格朗日函数**关于 x_1 和 x_2。
2. **固定 x_1 和 x_2，更新 λ 和 μ。**
3. **更新 ρ**，使惩罚项逐渐增加，逼近约束的满足。

步骤 4：迭代优化

通过不断的迭代，优化过程会逐渐逼近最优解，直到满足收敛条件。

```python
import numpy as np

# 定义目标函数
def objective(x):
    return x[0]**2 + x[1]**2

# 等式约束：x1 + x2 - 1 = 0
def equality_constraint(x):
```

```
    return x[0] + x[1] - 1

# 不等式约束：x1^2 + x2^2 - 2 <= 0
def inequality_constraint(x):
    return x[0]**2 + x[1]**2 - 2

# 增广拉格朗日函数
def augmented_lagrangian(x, lam, mu, rho):
    f = objective(x)
    h = equality_constraint(x)
    g = inequality_constraint(x)
    return f + lam * h + mu * g + 0.5 * rho * (h**2 + g**2)

# 增广拉格朗日函数的梯度
def gradient_augmented_lagrangian(x, lam, mu, rho):
    grad_f = np.array([2*x[0], 2*x[1]])  # 目标函数梯度
    grad_h = np.array([1, 1])            # 等式约束的梯度
    grad_g = np.array([2*x[0], 2*x[1]])  # 不等式约束的梯度

    # 增广拉格朗日的梯度
    grad = grad_f + lam * grad_h + mu * grad_g + rho * (equality_constraint(x) *
grad_h + inequality_constraint(x) * grad_g)
    return grad

# 初始值
x = np.array([0.5, 0.5])  # 初始猜测
lambda_val = 0.0          # 等式约束的拉格朗日乘子
mu_val = 0.0              # 不等式约束的拉格朗日乘子
rho_val = 10.0            # 增广项的惩罚系数

# 迭代优化
max_iter = 1000
for i in range(max_iter):
    # 计算梯度
    grad = gradient_augmented_lagrangian(x, lambda_val, mu_val, rho_val)

    # 使用梯度下降更新x
    x = x - 0.01 * grad  # 这里简单使用固定步长的梯度下降

    # 更新拉格朗日乘子
    lambda_val += rho_val * equality_constraint(x)
    mu_val += rho_val * inequality_constraint(x)

    # 增大惩罚系数
    rho_val *= 1.1  # 增大rho

    # 收敛条件检查
    if np.linalg.norm(grad) < 1e-6:
        break

print("优化结果:", x)
```