使用 scipy.optimize.differential_evolution 进行带约束的优化

1 简介

SciPy 库中的 scipy.optimize.differential_evolution 函数基于差分进化算法，用于全局优化。这种算法适用于连续变量优化问题，可以在不需要梯度信息的情况下找到全局最优解。

2 基本用法

下面是使用 differential_evolution 最小化一个简单目标函数的示例：

```python
from scipy.optimize import differential_evolution

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

# 定义变量的边界
bounds = [(-5, 5), (-5, 5), (-5, 5)]

# 使用差分进化算法进行优化
result = differential_evolution(objective_function, bounds)

# 输出优化结果
print('最优解:', result.x)
print('最优值:', result.fun)
```

3 主要参数

- func: 目标函数。
- bounds: 变量的边界，格式为 $(\min, \max)$ 的列表。
- args: 传递给目标函数的附加参数。
- strategy: 差分进化策略，默认是 'best1bin'。
- maxiter: 最大迭代次数，默认是 1000。
- popsize: 种群规模，是变量数量的倍数，默认是 15。

- `tol`: 收敛容差，默认是 0.01。
- `mutation`: 变异因子，可以是浮点数或 $(\min, \max)$ 元组。
- `recombination`: 交叉概率，默认是 0.7。
- `seed`: 随机数生成器的种子。
- `callback`: 每次迭代结束时调用的函数。
- `disp`: 布尔值，设置为 True 时打印收敛消息。
- `polish`: 布尔值，设置为 True 时在结束时进行局部优化，默认是 True。
- `init`: 指定初始种群。

4 Rosenbrock 函数

Rosenbrock 函数，又称为 Rosenbrock 山谷或 Rosenbrock 香蕉函数，是优化算法的常见测试问题。其定义如下：

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \tag{1}$$

其中，$\mathbf{x} = (x_1, x_2, \ldots, x_n)$。
该函数在 $(1, 1, \ldots, 1)$ 处具有全局最小值，其中 $f(\mathbf{x}) = 0$。

5 带约束的优化

为了处理约束，`differential_evolution` 可以结合边界约束和非线性约束使用。

5.1 非线性约束的示例

假设我们要在以下约束条件下优化 Rosenbrock 函数：

$$x_1 + x_2 \leq 1$$
$$x_1^2 + x_2^2 \leq 1$$

实现代码如下：

```
from scipy.optimize import differential_evolution, NonlinearConstraint

# 定义目标函数
def rosenbrock(x):
    return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

# 定义约束函数
def constraint1(x):
    return x[0] + x[1] - 1

def constraint2(x):
```

```
    return x[0]**2 + x[1]**2 - 1

# 创建非线性约束
nlc1 = NonlinearConstraint(constraint1, -np.inf, 0)
nlc2 = NonlinearConstraint(constraint2, -np.inf, 0)

# 使用差分进化算法进行优化
result = differential_evolution(rosenbrock, bounds=[(-2, 2), (-2, 2)], constraints
    =(nlc1, nlc2))

# 输出优化结果
print('最优解:', result.x)
print('最优值:', result.fun)
```