运筹学
编程作业

2019010485　自 91　刘祖炎

2022 年 5 月 29 日

1　等值线

目标函数

$$(1-x)^2+2(x^2-y)^2$$

等值线如图1所示。

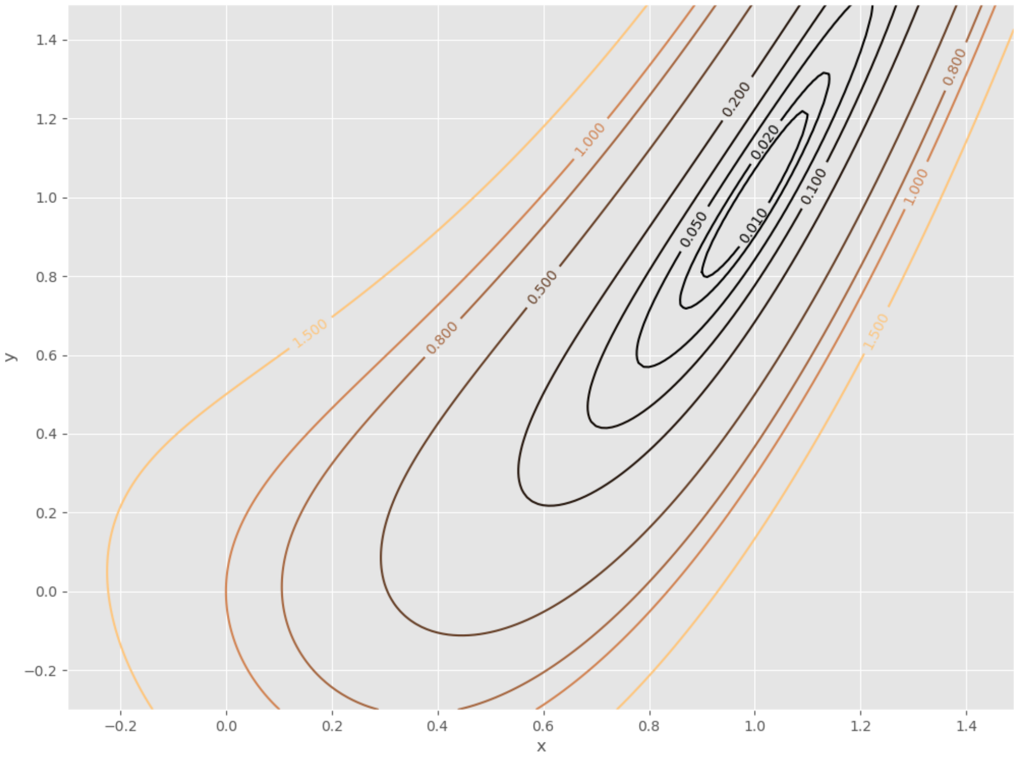

图 1: 目标函数等值线

2　最速下降方向

2.1　算法流程

- 设置初始点为 $(0,0)$，并开始迭代。

- 在第 k 次迭代时，求解 (x_k, y_k) 处的最速下降方向 $\mathbf{D}_k$，根据题意，可分别求解 l_1, l_2, l_∞ 意义下的最速下降方向。
- 根据一维精确搜索的方法，求解参数 t，并更新 $(x_{k+1}, y_{k+1}) = (x_k, y_k) + t \cdot \mathbf{D}_k$。
- 根据求得的参数 t，判断是否满足迭代终止条件 $||\nabla f(x)||_2 \leq 10^{-4}$，若满足，则停止迭代，若不满足，则继续迭代。

2.2 实现细节

根据目标函数，可求得梯度表示式为

$$
\begin{aligned}
\nabla f(x) &= 2(x-1) + 8x(x^2 - y) \\
\nabla f(y) &= 4(y - x^2)
\end{aligned}
\tag{1}
$$

在 l_1 范数下，最速下降方向的实现代码为

```python
if (abs(delta_x) > abs(delta_y)):
    d = np.array([-np.sign(delta_x), 0])
else:
    d = np.array([0, -np.sign(delta_y)])
```

在 l_2 范数下，最速下降方向的实现代码为

```python
delta_f = math.sqrt(delta_x ** 2 + delta_y ** 2)
d = np.array([-delta_x, -delta_y]) / delta_f
```

在 l_∞ 范数下，最速下降方向的实现代码为

```python
d = np.array([-np.sign(delta_x), -np.sign(delta_y)])
```

一维精确搜索利用 sympy 库实现，定义变量 t 后，按照公式进行求解。由于求解的方程为一元三次方程，可能存在复数解，因而优先取最小的正实数解，若无实数解，则取所有复数解中最小的实部。实现代码为

```python
t = Symbol('t')
delta_xt, delta_yt = delta(x + t * d[0], y + t * d[1])
result = solve([delta_xt * d[0] + delta_yt * d[1]], [t])
if len(result) == 1:
    t = result[t]
elif len(result) > 1:
    t = complex(result[0][0]).real
x = float(x + t * d[0])
y = float(y + t * d[1])
```

2.3 求解结果

求解结果如表1所示。

表 1: 求解结果

方法	迭代次数	最优解	最优值
l_1 范数	134	$(0.99990496, 0.99978617)$	1.016180×10^{-8}
l_2 范数	134	$(0.99990496, 0.99978617)$	1.016180×10^{-8}
l_∞ 范数	101	$(0.99990365, 0.99979125)$	9.798739×10^{-9}

l_1 范数下，决策变量的更新过程如图2所示，函数值更新过程如图3所示。

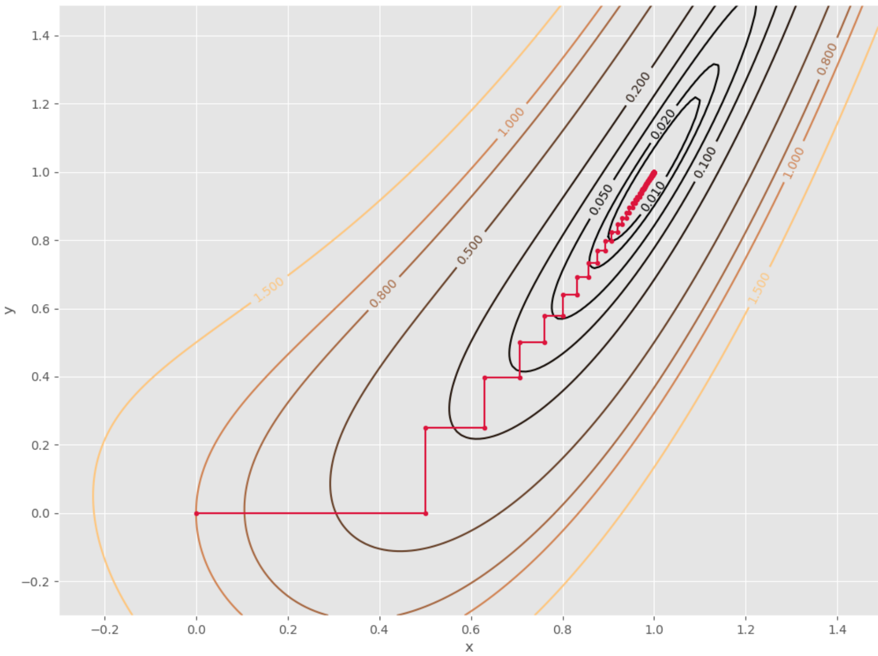

图 2: 决策变量更新过程 (l_1 范数)

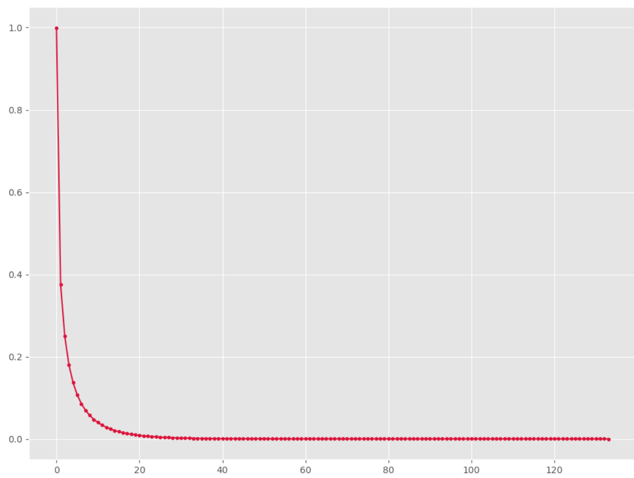

图 3: 函数值更新过程 (l_1 范数)

l_2 范数下，决策变量的更新过程如图4所示，函数值更新过程如图5所示。

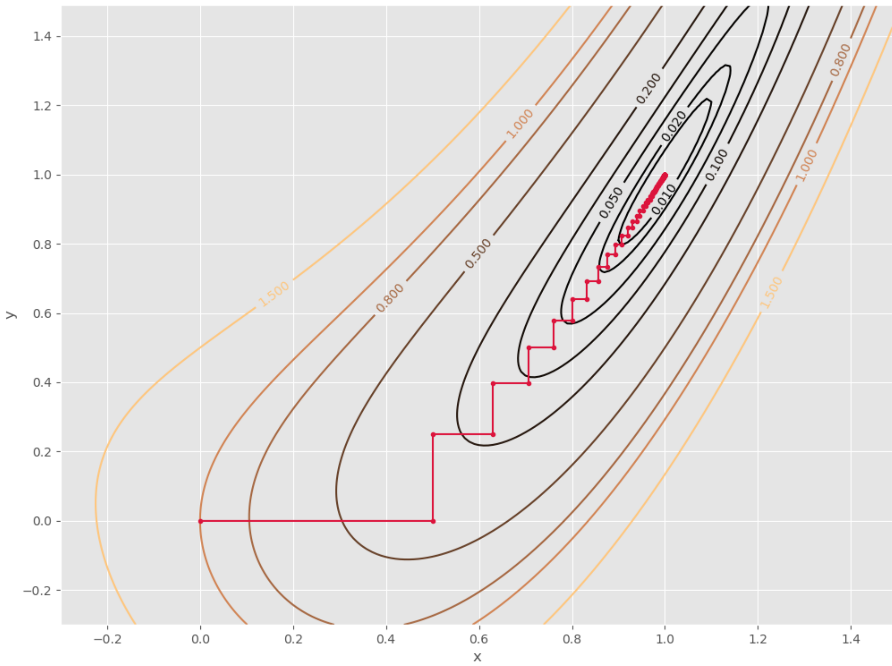

图 4: 决策变量更新过程 (l_2 范数)

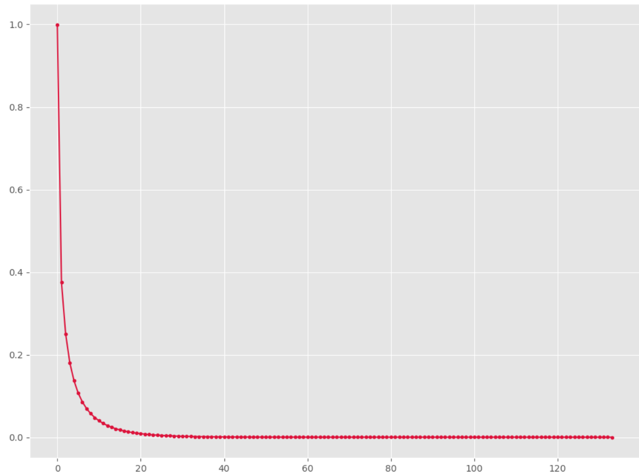

图 5: 函数值更新过程 (l_2 范数)

l_∞ 范数下，决策变量的更新过程如图6所示，函数值更新过程如图7所示。

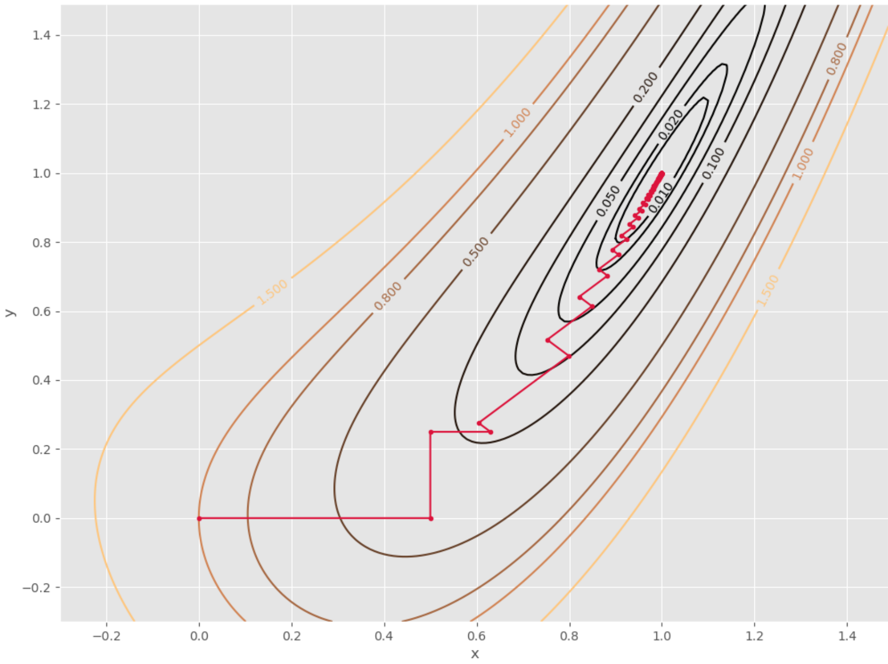

图 6: 决策变量更新过程 (l_∞ 范数)

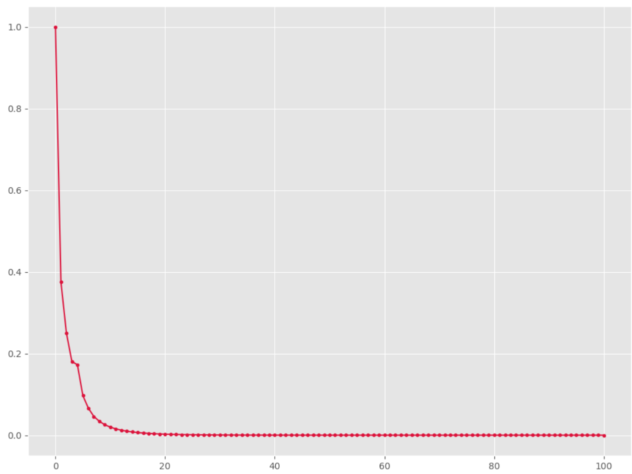

图 7: 函数值更新过程 (l_∞ 范数)

3 共轭梯度法

3.1 算法流程

共轭梯度法的算法流程与最速下降方向方法相同。

3.2 实现细节

利用 Fletcher-Reeves 方法时，实现代码为

```
alpha = (delta_x ** 2 + delta_y ** 2) / (delta_oldx ** 2 + delta_oldy ** 2)
d = -np.array([delta_x, delta_y]) + alpha * d
```

利用 Polak-Ribiere 方法时，实现代码为

```
alpha = np.array([delta_x, delta_y]).T @ (np.array([delta_x, delta_y])
        - np.array([delta_oldx, delta_oldy])) / (delta_oldx ** 2 + delta_oldy ** 2)
d = -np.array([delta_x, delta_y]) + alpha * d
```

其余实现细节与最速下降方法相同。

3.3 求解结果

Fletcher-Reeves 方法下，决策变量的更新过程如图8所示，函数值更新过程如图9所示。

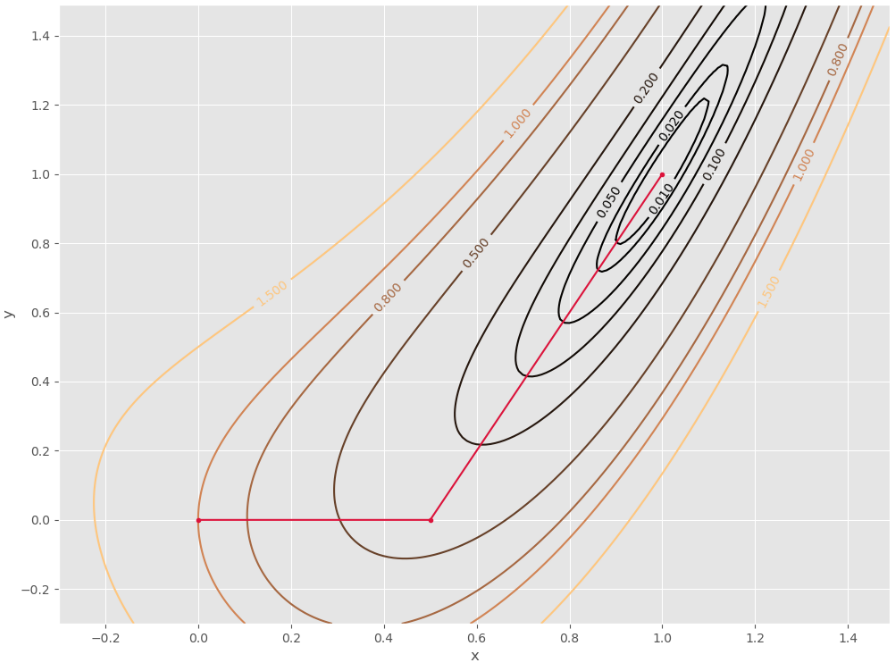

图 8: 决策变量更新过程 (Fletcher-Reeves 方法)

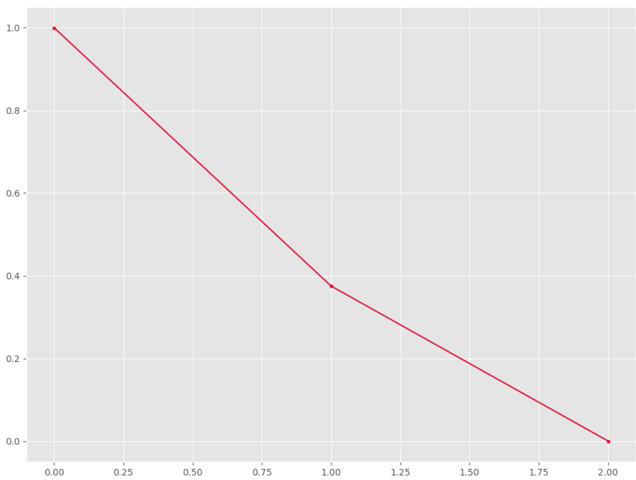

图 9: 函数值更新过程 (Fletcher-Reeves 方法)

Polak-Ribiere 方法下，决策变量的更新过程如图10所示，函数值更新过程如图11所示。

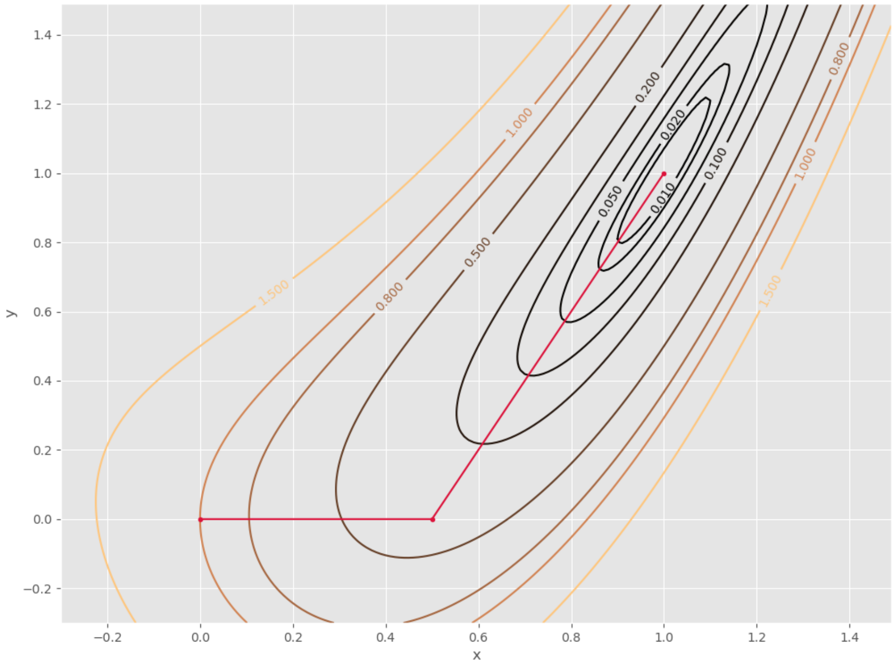

图 10: 决策变量更新过程 (Polak-Ribiere 方法)

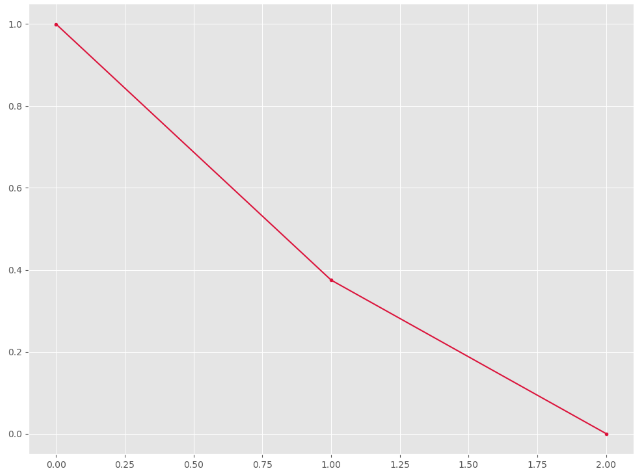

图 11: 函数值更新过程 (Polak-Ribiere 方法)

上述两种方法经过两次迭代均可以求得最优解。

4 分析比较

- 三种最速下降方法的求解结果类似，其中无穷范数意义下迭代次数较少。
- 两种共轭梯度方法求解结果相同，经过两次迭代均可以求得最优解。
- 共轭梯度方法的迭代次数明显少于最速下降方法，这是由于最速下降方法的步长会逐渐缩短，共轭梯度方法无此限制。