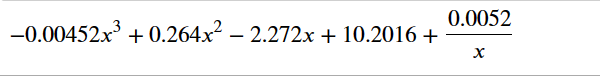
## 问题背景和建模

某大型化工厂需要采购甲乙两种原材料总共10吨，但是甲乙原材料之间是可以通过化学反应相互转换的。由于市场波动的原因，甲乙原材料的进价并不是固定的，以下列出了市场可能预期出现的情况。我们希望找出如何分配甲乙进货的数量，能够使得总共进货10吨，成本最小。

|  |  |  |
| --- | --- | --- |
| 甲的价格函数（万元） | 乙的价格函数（万元） | 出现的概率 |
|  |  | 0.48 |
|  |  | 0.52 |

这个问题是典型的最优化函数求极值的问题，求出平均期望函数的最小值就是成本最低价。

对问题进行建模，得到函数：



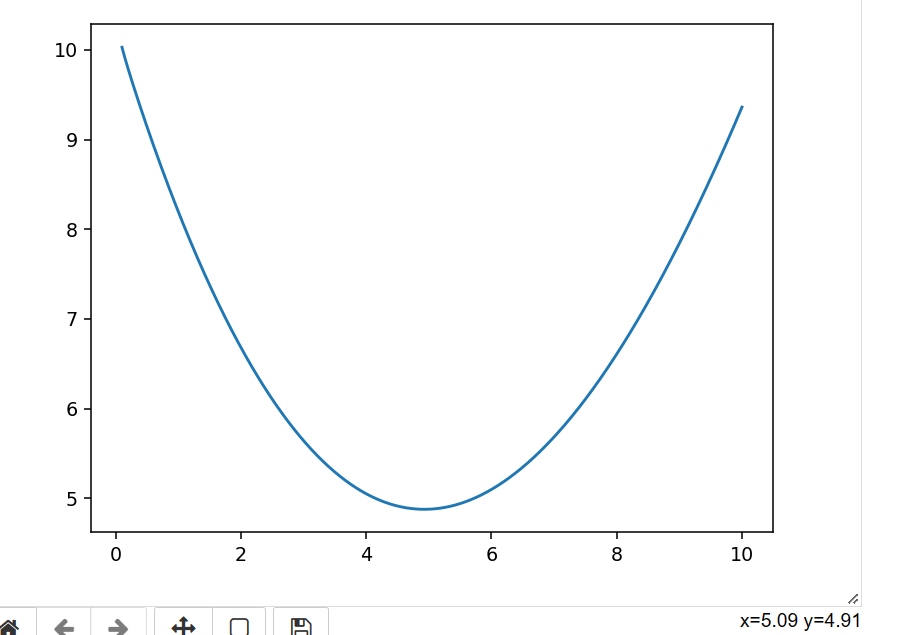
其中，是一个正实数。

以上就是问题的背景和建模。

## 问题求解

在这一部分我们求解问题，上面的方程很明显是一个非线性方程，为了实验不同最优化方法的效果，采用Python科学计算库scipy.optimazation库进行求解，调用不同的最优化方法并且进行对比。Scipy是一个非常有名的科学计算库，几乎涵盖了所有的最优化计算方法，例如梯度下降法、拟牛顿法、牛顿法、最小二乘法等等。

首先画出这个非线性函数在的图像情况。可以看到最小值是附近。



接下来采用不同的最优化方法收敛到以上函数图像的极值。

这里我们采用两种算法：

1. BFGS拟牛顿法

2. SLSQP序列最小二乘规划算法。

并且对于以上三种最优化算法，都设置1/2/5/10四种收敛迭代次数，所以优化代码会循环运行8次，产生不同的最优化结果。但上面的函数比较简单，所以无论是BFGS拟牛顿法还是Scipy默认的SLSQP都能取得不错的效果。

这里请同学自己写一下上面两种算法的原理，丰富一下内容。

## 实验结果分析

python optimization.py，得到以下运行结果和图像：

-------------------------------------------

本次循环中，使用的最优化方法是BFGS,最大收敛迭代次数设置为1,得到的结果和图像如下：

fun: 6.847111151157497

hess\_inv: array([[2.01450614]])

jac: array([-1.33273536])

message: 'Maximum number of iterations has been exceeded.'

nfev: 4

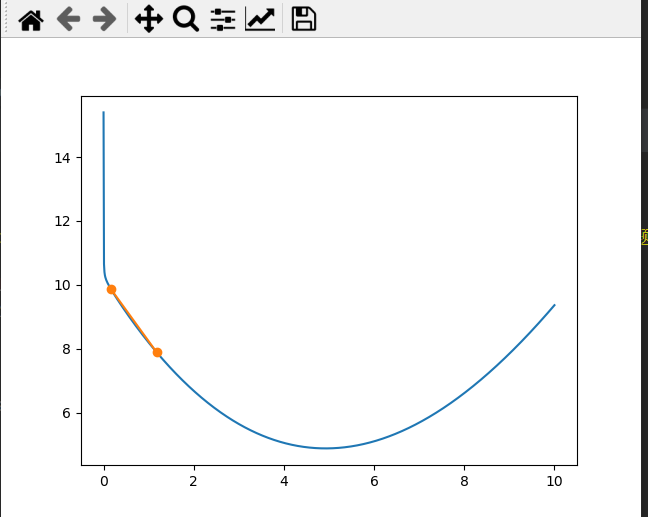
nit: 1

njev: 2

status: 1

success: False

x: array([1.87169074])



-------------------------------------------

本次循环中，使用的最优化方法是BFGS,最大收敛迭代次数设置为2,得到的结果和图像如下：

fun: 5.190397514259551

hess\_inv: array([[2.16715625]])

jac: array([-0.51164913])

message: 'Maximum number of iterations has been exceeded.'

nfev: 6

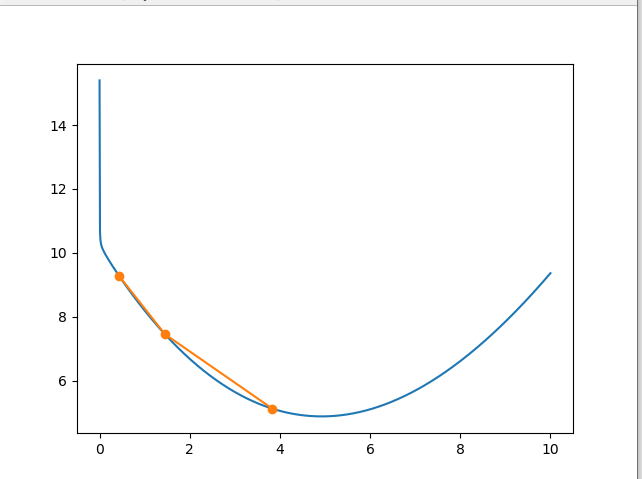
nit: 2

njev: 3

status: 1

success: False

x: array([3.68310408])



-------------------------------------------

本次循环中，使用的最优化方法是BFGS,最大收敛迭代次数设置为5,得到的结果和图像如下：

fun: 4.876585712431289

hess\_inv: array([[2.53404945]])

jac: array([-3.10540199e-05])

message: 'Maximum number of iterations has been exceeded.'

nfev: 12

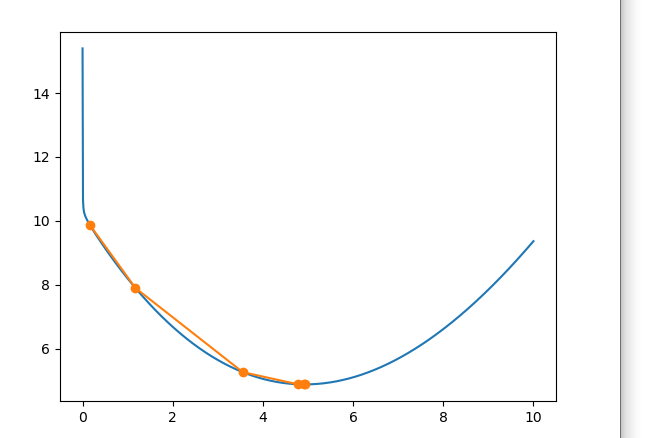
nit: 5

njev: 6

status: 1

success: False

x: array([4.92674809])



-------------------------------------------

本次循环中，使用的最优化方法是BFGS,最大收敛迭代次数设置为10,得到的结果和图像如下：

fun: 4.876585711305757

hess\_inv: array([[2.52345023]])

jac: array([-8.64267349e-06])

message: 'Optimization terminated successfully.'

nfev: 12

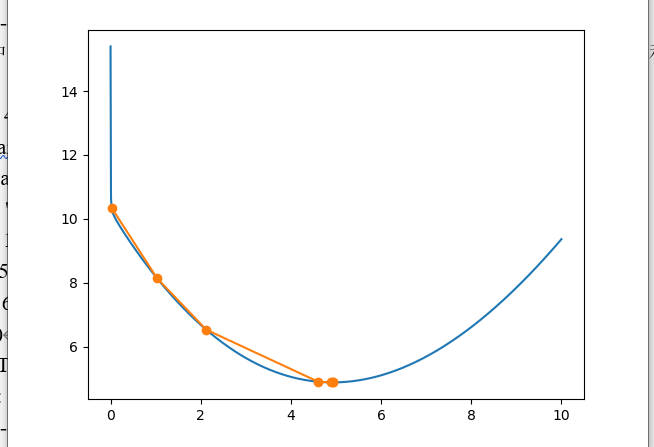
nit: 5

njev: 6

status: 0

success: True

x: array([4.92680473])



-------------------------------------------

本次循环中，使用的最优化方法是SLSQP,最大收敛迭代次数设置为1,得到的结果和图像如下：

fun: 9.278495860340923

jac: array([-2.07350802])

message: 'Iteration limit reached'

nfev: 2

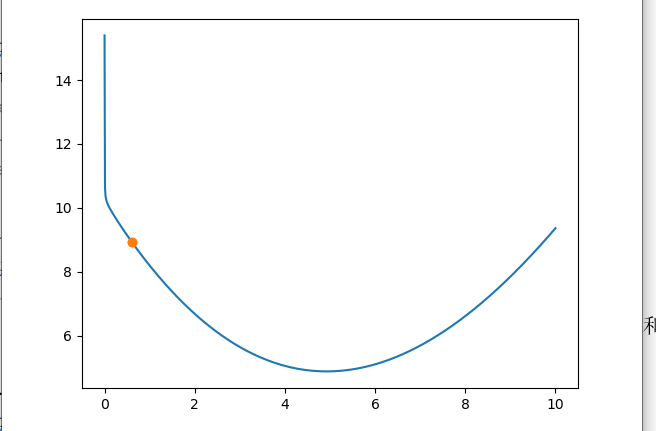
nit: 1

njev: 1

status: 9

success: False

x: array([0.43322547])



-------------------------------------------

本次循环中，使用的最优化方法是SLSQP,最大收敛迭代次数设置为2,得到的结果和图像如下：

fun: 5.95325336481049

jac: array([-0.96907985])

message: 'Iteration limit reached'

nfev: 4

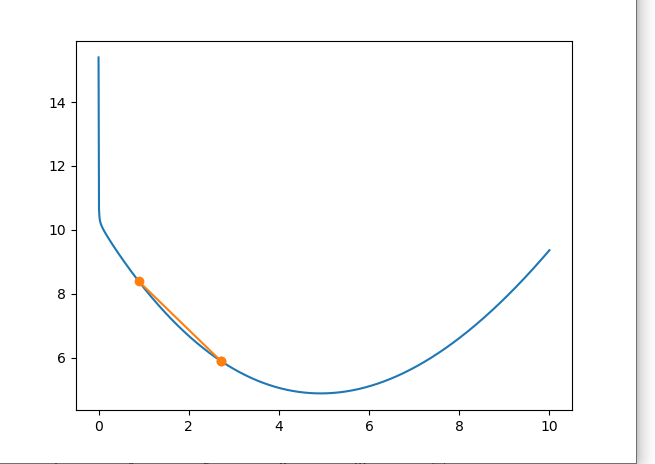
nit: 2

njev: 2

status: 9

success: False

x: array([2.64931201])



-------------------------------------------

本次循环中，使用的最优化方法是SLSQP,最大收敛迭代次数设置为5,得到的结果和图像如下：

fun: -4.6475246898528706e+39

jac: array([-1.38138651e+26])

message: 'Iteration limit reached'

nfev: 10

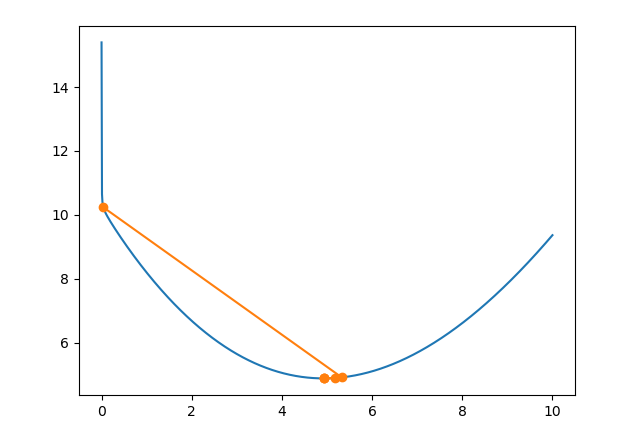
nit: 5

njev: 5

status: 9

success: False

x: array([1.00931739e+14])



-------------------------------------------

本次循环中，使用的最优化方法是SLSQP,最大收敛迭代次数设置为10,得到的结果和图像如下：

fun: 4.876585724620867

jac: array([-0.00010294])

message: 'Optimization terminated successfully'

nfev: 10

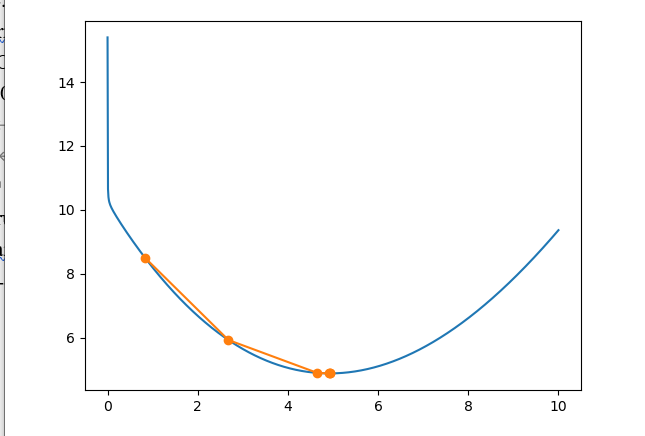
nit: 5

njev: 5

status: 0

success: True

x: array([4.92656602])



观察上述图像，可以得到如下结论：

在最大下降次数为1次和2次的时候，BFGS和SLSQP都无法逼近最优x值4.9625，因为迭代次数太少了。

在最大下降次数为5次的时候，BFGS拟牛顿法可以得到最优的x值，表现较好，但SLSQP方法在下降过程中左右横移，未能下降到最优解。

对于这个比较简单的函数而言，迭代次数设置为10，已经能保证两种方法都趋近最优解。至此最优化问题得解，以上方程x取4.92656602，得到最低成本4.876585724620867。