基于一维搜索的辛烷值压缩比法测定自动化

研究背景：

辛烷值是发动机燃料（汽油）的抗爆性能好坏的一项重要指标。若辛烷值越高，抗爆性就越好，发动机就可以用更高的压缩比。也就是说，如果炼油厂生产的汽油的辛烷值不断提高，则汽车制造厂可随之提高发动机的压缩比，这样不仅可提高发动机功率，增加行车里程数，还可以节约燃料，对提高汽油的动力经济性能是有重要意义的。

主要内容：

本文选用压缩比法来测定辛烷值，并依据爆震强度最大的液位高度基本上不随压缩比的变化而变化的结论，将标准压缩比测试可以转化为以压缩比为变量的一元非线性方程的求解问题，并采用一维搜索算法来求解。本文首先设计了线性插值和黄金分割两种适用压缩比法测定自动化的一维搜索算法，然后对两种搜索算法进行了数值仿真并对比， 最后得出基于黄金分割的搜索算法性能更加优良。

具体实施:

由于期刊中的数据没有公开，故直接利用作者拟合的爆震强度与压缩比关系的模型，期刊中拟合的方程为：

$$y = 0.0000001552x^3 - 0.0003384x^2 + 0.3213x - 31.15$$

（1） 基于线性插值搜索算法的程序

```
#函数的定义
def f(x):
    return (0.0000001522*x**3-0.0003384*x**2+0.3213*x-31.15)
def s(a,b):
    x=round((50-f(a))/(f(a)-f(b))*(a-b)+a)
    #x = (50-f(a))/(f(a)-f(b))*(a-b)+a
    return x
#基于线性插值的标准爆震搜索算法
def Linear_interpolation(a,b):
    x1=a
    y1=f(x1)
    print('x1,y1',x1,y1)
    x2=b
    y2=f(x2)
    print('x2,y2',x2, y2)
```

```
    stepNum=2    #因为前面 x1,x2 的计算就属于两次了，故设置的初始值为 2
    while ((y1>52)|(y1<48))|((y2>52)|(y2<48)):
        stepNum=stepNum+1
        xs = s(x1, x2)
        ys = f(xs)
        print('xs,ys',xs,ys)
        if ys>52:
            x1=xs
            y1 = ys
        elif ys<48:
            x2=xs
            y2=ys
        else:
            break
    else:
        if 48<=y1<=52:
            stepNum-=1
            ys=y1
        else:
            stepNum -= 1
            ys=y2
    if ys<50:
        precision=ys/50
    else:
        precision=50/ys
    return (precision,stepNum)

a=300;b=600
print('初始区间为：',[a,b])
precision,stepNum=Linear_interpolation(a,b)
print('搜索次数为%d,搜索精度为%.2f%%'%(stepNum,precision*100))
```

```
a1=200;b1=600
print('初始区间为：',[a1,b1])
precision1,stepNum1=Linear_interpolation(a1,b1)
print('搜索次数为%d,搜索精度
为%.2f%%'%(stepNum1,precision1*100))

a2=150;b2=700
print('初始区间为：',[a2,b2])
precision2,stepNum2=Linear_interpolation(a2,b2)
print('搜索次数为%d,搜索精度
为%.2f%%'%(stepNum2,precision2*100))
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/zy1.py
初始区间为： [300, 600]
搜索次数为4,搜索精度为98.23%
初始区间为： [200, 600]
搜索次数为4,搜索精度为96.34%
初始区间为： [150, 700]
搜索次数为5,搜索精度为98.89%

Process finished with exit code 0
```

图 1 基于线性插值搜索算法的结果

(2) 基于黄金分割搜索算法的程序

```
def f(x):#函数的定义
    return (0.0000001522*x**3-0.0003384*x**2+0.3213*x-31.15)
#基于黄金分割法的标准爆震强度搜索算法
def goldenOpt(a,b,Theta_error):
    stepNum=1
    while abs(b-a)>Theta_error:
        stepNum += 1
        print('a,b',a,b)
        x1=round(a+0.382*(b-a))
        x2=a+b-x1
        y1=f(x1)
        print('x1,x2,y1',x1,x2,y1)
```

```
            if y1>52:
                b=x1
            elif y1<48:
                y2=f(x2)
                if y2>52:
                    a=x1
                    b=x2
                elif y2<48:
                    a=x2
                else:
                    ys=y2
                    break
            else:
                ys=y1
                break
        if ys<50:
            precision=ys/50
        else:
            precision=50/ys
        return (precision, stepNum)
a=300;b=600;Theta_error=0.005
print('初始区间为：',[a,b])
precision,stepNum=goldenOpt(a,b,Theta_error)
print('搜索次数为%d,搜索精度为%.2f%%'%(stepNum,precision*100))

a1=200;b1=600;Theta_error1=0.005
print('初始区间为：',[a1,b1])
precision1,stepNum1=goldenOpt(a1,b1,Theta_error1)
print('搜索次数为%d,搜索精度为%.2f%%'%(stepNum1,precision1*100))

a2=150;b2=700;Theta_error2=0.005
print('初始区间为：',[a2,b2])
precision2,stepNum2=goldenOpt(a2,b2,Theta_error2)
print('搜索次数为%d,搜索精度为%.2f%%'%(stepNum2,precision2*100))
```

```
'''
a3=372;b3=380;Theta_error3=0.005
print('初始区间为：',[a3,b3])
precision3,stepNum3=goldenOpt(a3,b3,Theta_error3)
print('搜索次数为%d,搜索精度为%.2f%%'%(stepNum3,precision3*100))
'''
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/zy2.py
初始区间为： [300, 600]
搜索次数为3,搜索精度为98.49%
初始区间为： [200, 600]
搜索次数为3,搜索精度为96.92%
初始区间为： [150, 700]
搜索次数为4,搜索精度为99.40%
```

图 2 基于黄金分割搜索算法的结果

高速公路收费费率模型研究

研究背景以及内容:

基于道路使用者路线选择行为理论、存在竞争线路条件下道路使用者选择行驶道路时的影响因素研究以及道路使用效益最大化和运营成本最小化的基本原则，从高速公路管理者和使用者的视角构建高速公路收费最优费率的双层优化模型，并给出模型的求解算法。根据沪蓉高速公路重庆长寿至万州段客货交通量预测结果，拟合出高速路的最优费率函数，并通过一维搜索求解出该高速公路的最优收费费率。

下面我将使用精确一维搜索求解其最优费率。从期刊中可以得知：最优费率函数为：

$$y = 1.1 \times \exp(-1.1912 \times x^2) \times x$$

由最优费率的实际意义可知，其取值一定大于 0.故做函数在[0,20]上的函数图像如下：

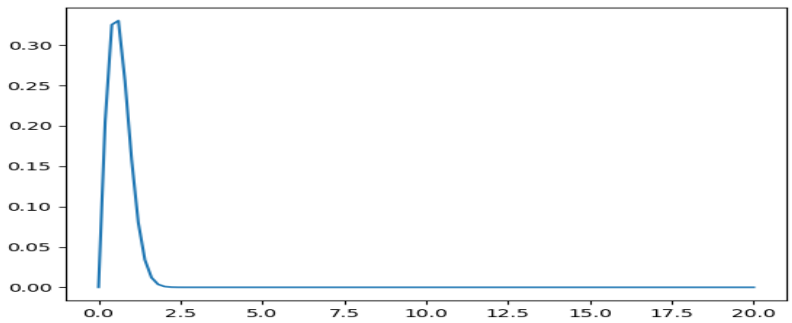


图 3 函数图像

从期刊的内容可知，求解最优费率也就是求解该目标函数的极大值。结合图3，就可以将求解最优费率的区间缩减到[0,2.5]之内。

（1） 使用黄金分割法求解该函数的极值问题的程序

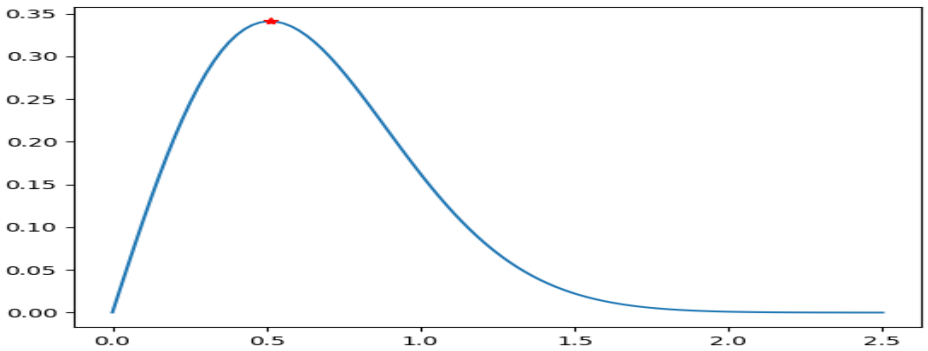
```
import math
import numpy as np
import matplotlib.pyplot as plt
#函数的定义
def f(x):
    return -1.1*np.exp(-1.912*x**2)*x
#黄金分割函数的定义
def goldenOpt(a,b,Theta_error):
    r=(3-math.sqrt(5))/2
    a1=a+r*(b-a)
    b1=a+(1-r)*(b-a)
    stepNum=0
    while abs(b-a)>Theta_error:
        stepNum=stepNum+1
        f1=f(a1)
        f2=f(b1)
        if f1>f2:
            a=a1
            f1=f2
            a1=b1
            b1=a+(1-r)*(b - a)
        else:
            b=b1
            f2=f1
            b1=a1
            a1=a+r*(b-a)
    x_opt=(a+b)/2
    f_opt=f(x_opt)
    return (x_opt, f_opt,stepNum,a,b)
```

```
a=0;b=2.5;Theta_error=0.003#初始值
x_opt,f_opt,stepNum,a,b=goldenOpt(a,b,Theta_error)
x=np.linspace(0, 2.5, 100)
y=-f(x)
plt.plot(x, y)
plt.plot(x_opt,-f_opt,'r*')
print('%f 是函数执行了%d 后的最优点'%(x_opt,stepNum))
print('区间长度为 %f'%(b-a))
plt.show()
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/p.py
0.512179是函数执行了14后的最优点
区间长度为 0.002966

Process finished with exit code 0
```



（2） 使用修正的斐波那契法求解该函数的极值问题的程序

```
import math
import numpy as np
import matplotlib.pyplot as plt

#函数的定义
def f(x):
    return -1.1*np.exp(-1.912*x**2)*x
#斐波那契数列
def func(num):
```

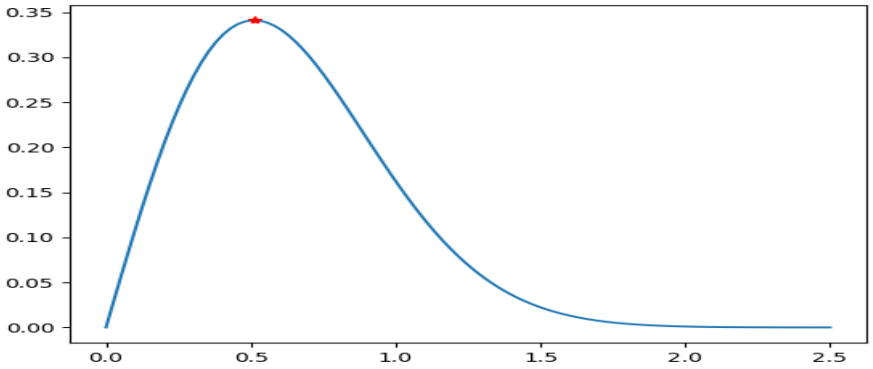
```
    numList=[0,1]
    for i in range(num - 2):
        numList.append(numList[-2] + numList[-1])
    return numList[2:]
F=func(50) #先随便生成一个斐波那契数列
def getn(ep,a,b,Theta_error):
    for n in range(len(F)):
        if F[n]>=((1+2*ep)*(b-a)/Theta_error):
            return n      #因为为位 F（N+1），故对应到此处就是
#斐波那契数列法
def Fibonacci(a,b,Theta_error,ep,ep1):
    n=getn(ep,a,b,Theta_error)
    stepNum=n
    r=1-F[n-1]/F[n]
    a1=a+r*(b-a)
    b1=a+(1-r)*(b-a)
    while n>=1:
        n=n-1
        f1=f(a1)
        f2=f(b1)
        if n>1:
            if f1>f2:
                a=a1
                f1=f2
                a1=b1
                r=1-F[n-1]/F[n]
                b1=a+(1-r)*(b - a)
            else:
                b=b1
                f2=f1
                b1=a1
                r=1-F[n-1]/F[n]
                a1=a+r*(b-a)
        elif n==1:
```

```
                if f1>f2:
                    a=a1
                    f1=f2
                    a1=b1
                    r=1-F[n-1]/F[n]
                    b1=a+(1-(r-ep1))*(b - a)
                else:
                    b=b1
                    f2=f1
                    b1=a1
                    r=1-F[n-1]/F[n]
                    a1=a+(r-ep1)*(b-a)
            else:
                if f1>f2:
                    a=a1
                    print(a, b)
                else:
                    b=b1
        x_opt=(a+b)/2
        f_opt=f(x_opt)
        return (x_opt, f_opt,stepNum,a,b)
    #函数的调用
    a=0;b=2.5;Theta_error=0.003;ep=0.1;ep1=0.05#初始值
    x_opt,f_opt,stepNum,a,b=Fibonacci(a,b,Theta_error,ep,ep1)
    x=np.linspace(0, 2.5, 100)
    y=-f(x)
    plt.plot(x, y)
    plt.plot(x_opt,-f_opt,'r*')
    print('%f 是函数执行了%d 后的最优点'%(x_opt,stepNum))
    print('区间长度为 %f'%(b-a))
    plt.show()
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/pp.py
0.511193是函数执行了15后的最优点
区间长度为 0.001722

Process finished with exit code 0
```



(3) 使用二分法求解该函数的极值问题的程序

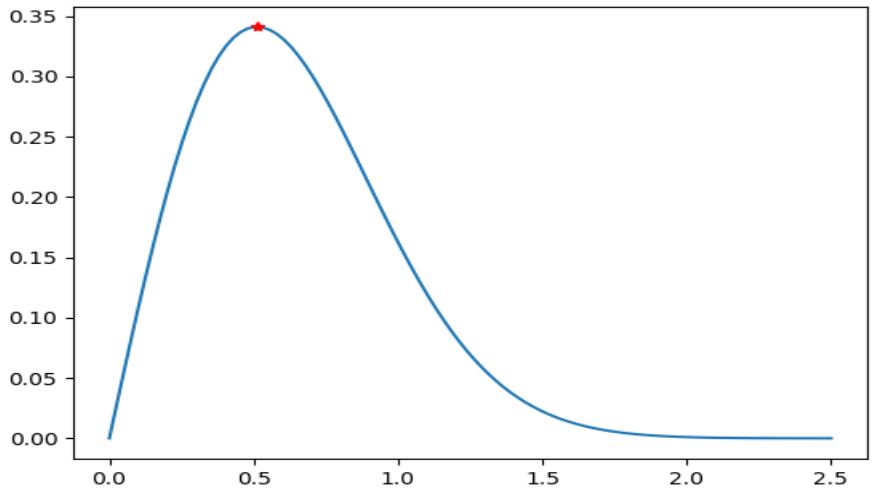
```
import sympy
import numpy as np
import matplotlib.pyplot as plt
def F(x):
    return -1.1*np.exp(-1.912*x**2)*x
def f(a):
    x = sympy.Symbol('x', real=True)
    return sympy.diff(-1.1*sympy.exp(-1.912*x**2)*x, x, 1).subs(x,
a)
def eff(a,b,Theta_error):
    stepNum=0
    while abs(b-a)>Theta_error:
        a1=(a+b)/2
        f1=f(a1)
        #print(a, b, f1)
        if f1>0:
            b=a1
        elif f1<0:
            a=a1
        else:
```

```
            print(a1,F(a1))
        stepNum = stepNum + 1
    return ((b+a)/2,F((b+a)/2),stepNum,a,b)
a=0;b=2.5;Theta_error=0.003#初始值
min_x,fmin_x,stepNum,a,b=eff(a,b,Theta_error)

x=np.linspace(0, 2.5, 100)
y=-F(x)
plt.plot(x, y)
plt.plot(min_x,-fmin_x,'r*')
print('%f 是函数执行了%d 后的最优点'%(min_x,stepNum))
print('区间长度为 %f'%(b-a))
plt.show()
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/ppp.py
0.511475是函数执行了10后的最优点
区间长度为 0.002441
```



(4) 使用牛顿法求解该函数的极值问题的程序

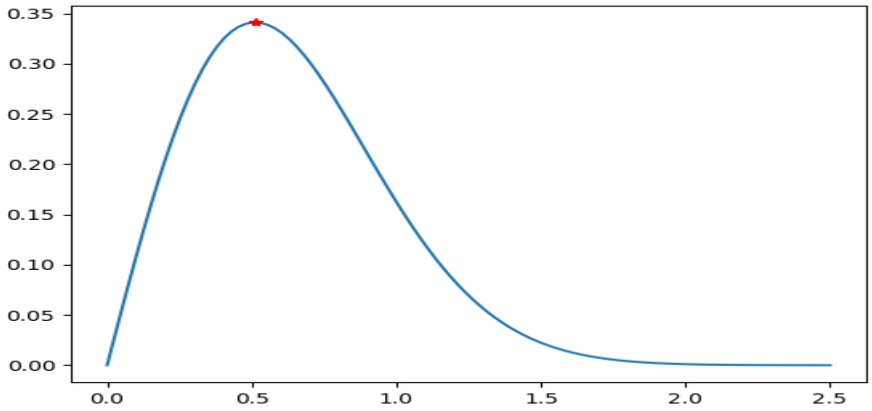
```
import sympy
import numpy as np
import math
import matplotlib.pyplot as plt
def F(x):
```

```
    return -1.1*math.exp(-1.912*x**2)*x
def f1(a):
    x = sympy.Symbol('x', real=True)
    return sympy.diff(-1.1*sympy.exp(-1.912*x**2)*x, x,
1).subs(x, a)
def f2(a):
    x = sympy.Symbol('x', real=True)
    return sympy.diff(-1.1*sympy.exp(-1.912*x**2)*x, x,
2).subs(x, a)
def newton(x,Theta_error):
    stepNum=0
    x1=x-f1(x)/f2(x)
    while abs(x1-x)>Theta_error:
        stepNum = stepNum+1
        x=x1
        x1=x-f1(x)/f2(x)
    return (x,x1,F(x1),stepNum)

x=0.25;Theta_error=0.003#初始值
minx_1,min_x,fmin_x,stepNum=newton(x,Theta_error)
x=np.linspace(0, 2.5, 100)
y=1.1*np.exp(-1.912*x**2)*x
plt.plot(x, y)
plt.plot(min_x,-fmin_x,'r*')
print('%f 是函数执行了%d 后的最优点'%(min_x,stepNum))
print('区间长度为 %f'%(min_x-minx_1))
plt.show()
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/pppp.py
0.511376是函数执行了2后的最优点
区间长度为 0.000826
```



(5) 使用割线法求解该函数的极值问题的程序

```
import sympy
import numpy as np
import math
import matplotlib.pyplot as plt
def F(x):
    return -1.1*math.exp(-1.912*x**2)*x
def f(a):
    x = sympy.Symbol('x', real=True)
    return sympy.diff(-1.1*sympy.exp(-1.912*x**2)*x, x, 1).subs(x,
a)

def secant_method(x,Theta_error):
    x0,x1=x
    stepNum=0
    x2=x1-(x1-x0)/(f(x1)-f(x0))*f(x1)
    while abs(x2-x1)>Theta_error:
        stepNum = stepNum+1
        x0=x1
        x1=x2
        x2=x1-(x1-x0)/(f(x1)-f(x0))*f(x1)
    return (x1,x2,F(x2),stepNum)

x=[0.1,0.2];Theta_error=0.003#初始值
minx_1,min_x,fmin_x,stepNum=secant_method(x,Theta_error)
```

```
x=np.linspace(0, 2.5, 100)
y=1.1*np.exp(-1.912*x**2)*x
plt.plot(x, y)
plt.plot(min_x,-fmin_x,'r*')
print('%f 是函数执行了%d 后的最优点'%(min_x,stepNum))
print('区间长度为 %f'%(abs(min_x-minx_1)))
plt.show()
```

实验结果：

```
D:\Python\Python36\python.exe C:/Users/lenovo/PycharmProjects/untitled3/ppppp.py
0.511381是函数执行了4后的最优点
区间长度为 0.000402
```

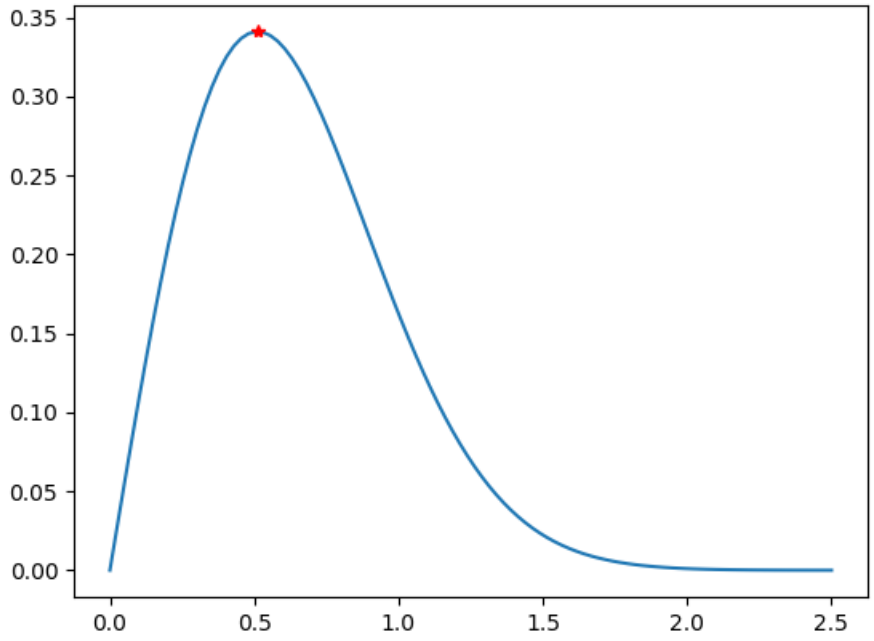


表 1 不同方法的对比

方法	最优值	迭代次数	最后区间长度
黄金分割法	0.512179	14	0.002966
修正的斐波那契数列法	0.511193	15	0.001722
二分法	0.511475	10	0.002441
牛顿法	0.511376	2	0.000826
割线法	0.511381	4	0.000402

从表 1 可以看出，不同的搜索方法最终确定的最优取值差不多，但是迭代次数却有较大的差距。相比其他三种方法来说，牛顿法和割线法的迭代次数较少，但它们对初值的选取较为敏感，甚至会失效。如：割线法的初始值设为 0.1,0.15

时，其最优值为 60.434650，但是其迭代次数为 10056.