西安电子科技大学

智能控制与控制智能 课程实验报告

实验名称 粒子群优化算法综合设计实验

电子工程 学院 190209 班

授课教师 吴宪祥

姓名 郭家豪 学号 19020100298

同作者

成 绩

实验日期 2020 年 4 月 12 - 日

指导教师评语： 指导教师： 年 月 日
实验报告内容基本要求及参考格式 一、实验目的 二、实验所用仪器（或实验环境） 三、实验基本原理及步骤（或方案设计及理论计算） 四、实验数据记录（或仿真及软件设计） 五、实验结果分析及回答问题（或测试环境及测试结果） 六、小组成员分工 七、 课程学习体会与收获

目录

实验内容

实验 1. 绘制单变量正态分布在区间[-4,4]上的波形 p(x)~N(0,1)，并利用粒子群优化算法求解其最大值。

实验 2. 绘制二元正态分布在区间（[-4,4]，[-4,4]）上波形$\boldsymbol{p(x_1,\ x_2)\sim N(\mu,\Sigma)}$，并利用粒子群优化算法求解其最大值。已知条件：$\boldsymbol{\mu=\begin{bmatrix}0\\0\end{bmatrix}}$，$\boldsymbol{\Sigma=\begin{bmatrix}1&0\\0&1\end{bmatrix}}$。

实验 3. 典型二阶欠阻尼控制系统若干结论验证。

(1) 查阅相关文献，利用粒子群优化算法求解 t 在[0,4π]区间二阶欠阻尼系统单位阶跃响应的最大值（假设〖ω〗_n=1rad/s，ζ=0.707），计算此时系统的超调量，讨论粒子群大小 swamSize 和最大迭代次数 maxgen 对寻优结果的影响。

(2) 编程绘制出误差带 Δ=5%时，阻尼比 ζ（在区间 0≤ζ≤1）与调整时间 ts 之间的关系曲线（三条关系曲线，真实调整时间、包络线调整时间、近似公式）；

(3)利用粒子群优化算法，以真实调整时间 ts 作为粒子群优化算法的适应度函数（Fitness）当误差带为 Δ=5%时，优化得到 0≤ζ≤1 区间内的最优 ζ 值，绘制出收敛曲线；

实验目的

1. 加深对粒子群优化算法的理解；
2. 提高动手能力；
3. 提高信息检索利用能力；

实验环境

1. 笔记本电脑一台
   a) 系统 Windows10（非必须），配置应该不重要（我猜）
   b) 联网（为了搜索一些信息）
   c) 使用 IDLE (Python 3.8 64-bit)，Microsoft Word，Chrome 浏览器等软件。
   d) 主要使用的 python 模块：matplotlib 等

实验基本原理和步骤

基本原理

粒子群优化（Particle Swarm Optimization, PSO）算法

（以下部分摘自自维基百科 https://zh.wikipedia.org/wiki/粒子群优化）

- 由 J. Kennedy 和 R. C. Eberhart 等于 1995 年开发。
- 同时维护多个解
- 每次迭代中，有一个目标函数来评估每个解的适应度（fitness）
- 每个解由搜索空间的一个例子表示
- 粒子“飞来飞去”，进而求得目标函数的最优解

$$v(t+1) = w * v(t) + c_1 * rand * (x_{best} - x) + c_2 * rand * (g_{best} - x)$$

$$x(t+1) = x(t) + v(t+1)$$

程序框图如下：

程序框图如下：

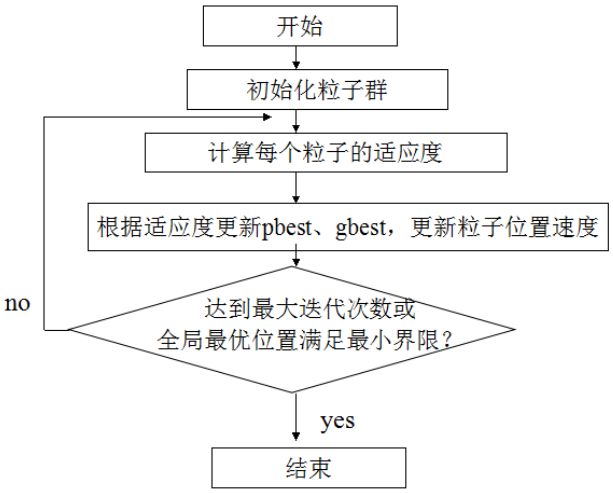


Figure 1 来源 https://www.cnblogs.com/21207-iHome/p/6062535.html

实验一

给定条件下：

$$p(x) = \frac{1}{\sqrt{2\pi}} exp\left(\frac{x^2}{2}\right)$$

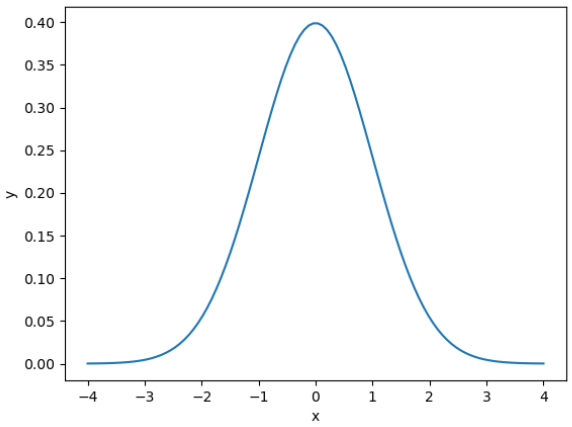
绘制曲线

a) 绘制曲线的 Python 代码如下

```
1. import math
2. import numpy as np
3. import matplotlib.pyplot as plt
4.
```

```
x = np.linspace(-4,4,100)

def func(x):
    c1=1/(math.sqrt(2*math.pi)*1)
    c2=-(x**2)/2
    return c1*math.exp(c2)

y = list(map(func,x))
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x,y)

plt.show()
```

b） 函数图像如下



求解

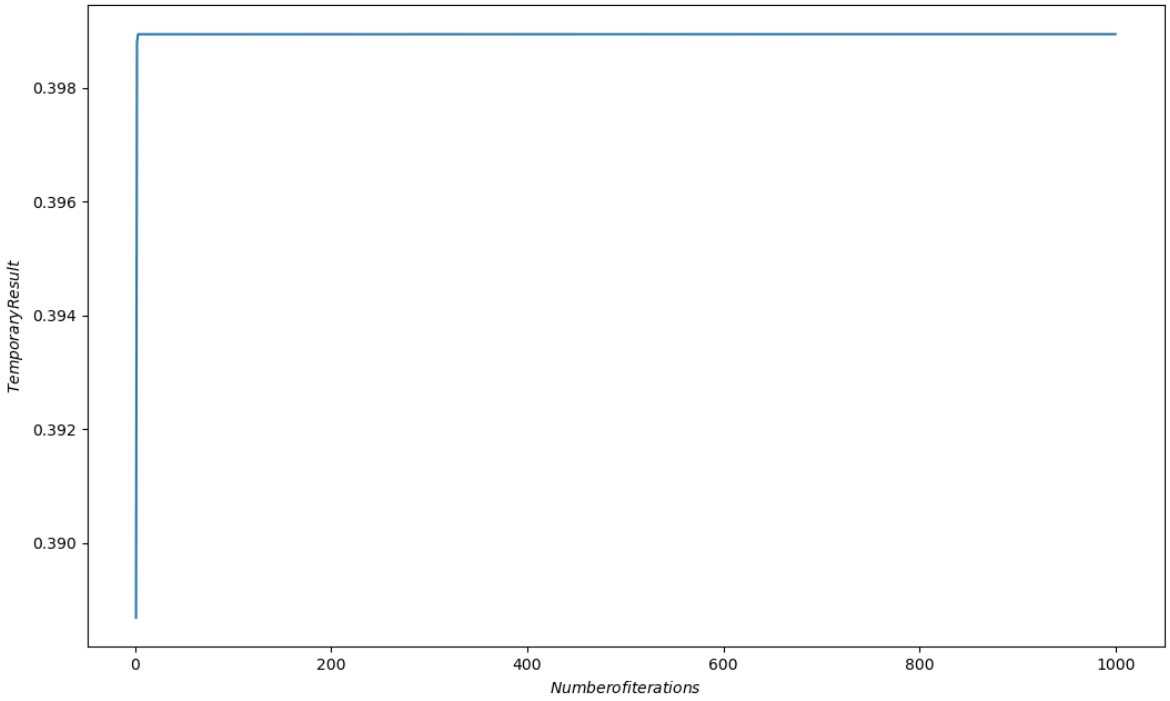
```
import math
from random import random
import numpy as np
class Pso():
    def __init__(self,pN,dim,max_iter,func):
        self.w = 0.5 #惯性因子
        self.c1 = 1.5 #自身认知因子
        self.c2 = 1.5 #社会任质因子
        self.pN = pN  #粒子数量
        self.maxv = 1 #最大速度
        self.dim = dim #维数
        self.max_iter = max_iter #迭代维数
        self.X = np.zeros((self.pN,self.dim)) #各维坐标，初值为全为0
        self.V = np.zeros((self.pN,self.dim)) #各维速度，初值为全为0
        self.pbest = np.zeros((self.pN,self.dim)) #粒子最优位置
        self.gbest = np.zeros((1,self.dim)) #全局最优位置
```

```
        self.p_bestfit = np.zeros(self.pN) #粒子最佳位置对应值
        self.fit = -1e15 #为求最大值，初始设为一足够小的值
        self.func = func #待求解函数

    #待求解函数
    def function(self,x):
        return self.func(x)

    #初始化粒子群
    def init_pop(self,):
        for i in range(self.pN):
            self.X[i] = np.random.uniform(-4,4,[1,self.dim])
            self.V[i] = np.random.uniform(0,self.maxv,[1,self.dim])

            self.pbest[i] = self.X[i] #更新粒子最佳位置
            self.p_bestfit[i] = self.function(self.X[i]) #更新对应值
        for i in range(self.pN):
            if(self.p_bestfit[i] > self.fit):
                self.gbest = self.X[i]
                self.fit = self.p_bestfit[i]

    #开始迭代
    def update(self):
        fitness = []

        for QAQ in range(self.max_iter): #迭代次数
            for i in range(self.pN):
                temp = self.function(self.X[i]) #获得该粒子位置的函数值
                if(temp > self.p_bestfit[i]):
                    self.p_bestfit[i] = temp
                    self.pbest[i] = self.X[i]
                    if(self.p_bestfit[i]>self.fit): #更新全局最优
                        self.fit = self.p_bestfit[i]
                        self.gbest = self.X[i]
            for i in range(self.pN):
                self.V[i] = min(self.w*self.V[i]+\
                            self.c1*random()*(self.pbest[i]-self.X[i])+\
                            self.c2*random()*(self.gbest-self.X[i]),self.max
v)

                self.X[i] = self.X[i] + self.V[i]

            fitness.append(self.fit)

```

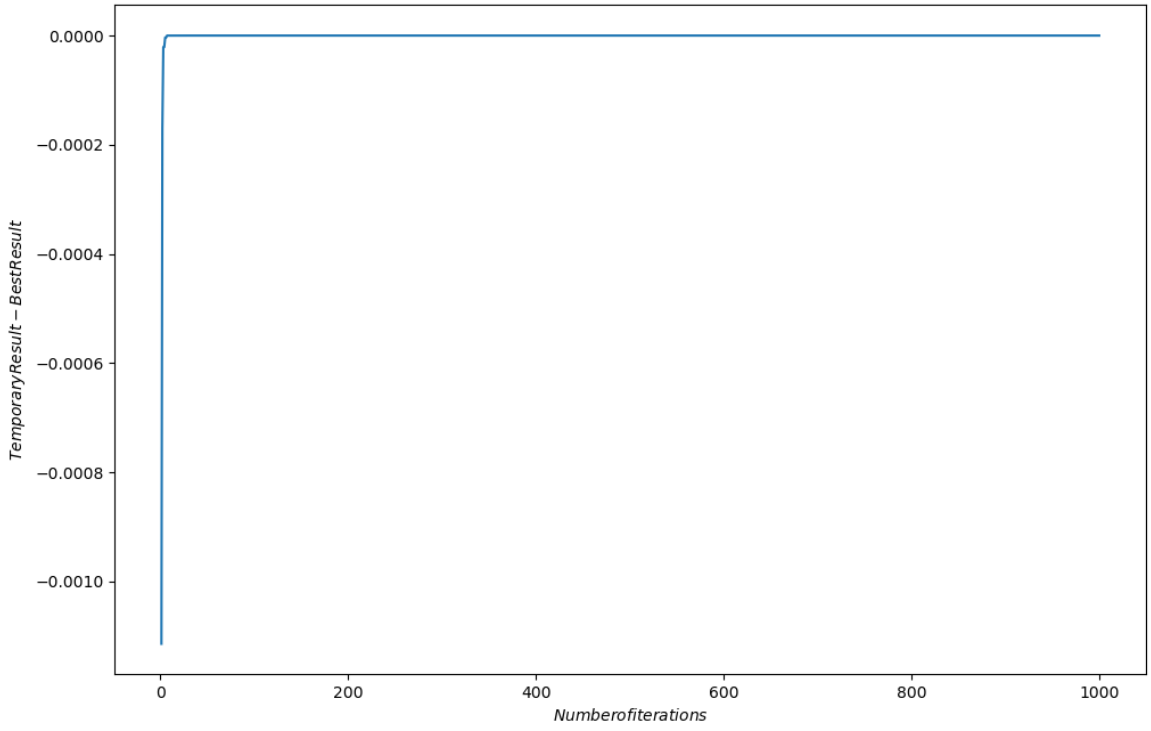
```
        return self.gbest,self.fit,fitness


def count_func(x):
    c1=1/(math.sqrt(2*math.pi)*1)
    c2=-(x**2)/2
    return c1*math.exp(c2)
iternum = 1000 #迭代数
pso = Pso(pN = 50,dim = 1,max_iter = iternum, func = count_func)
pso.init_pop()
x_best,fit_best,fitness= pso.update()

print("{:.6f}  {:.6}".format(x_best[0],fit_best))

print(fitness-fit_best)
import matplotlib.pyplot as plt

##下面是结果分析（画图）的代码
x = range(1,iternum+1)
plt.plot(x,fitness)
plt.xlabel(r'$Number of iterations$')
plt.ylabel(r'$TemporaryResult$')

plt.show()
```

输出结果：0.000000 0.398942 （保留六位小数）
可知最终结果 x = 0.00000 p(x) = 0.398942

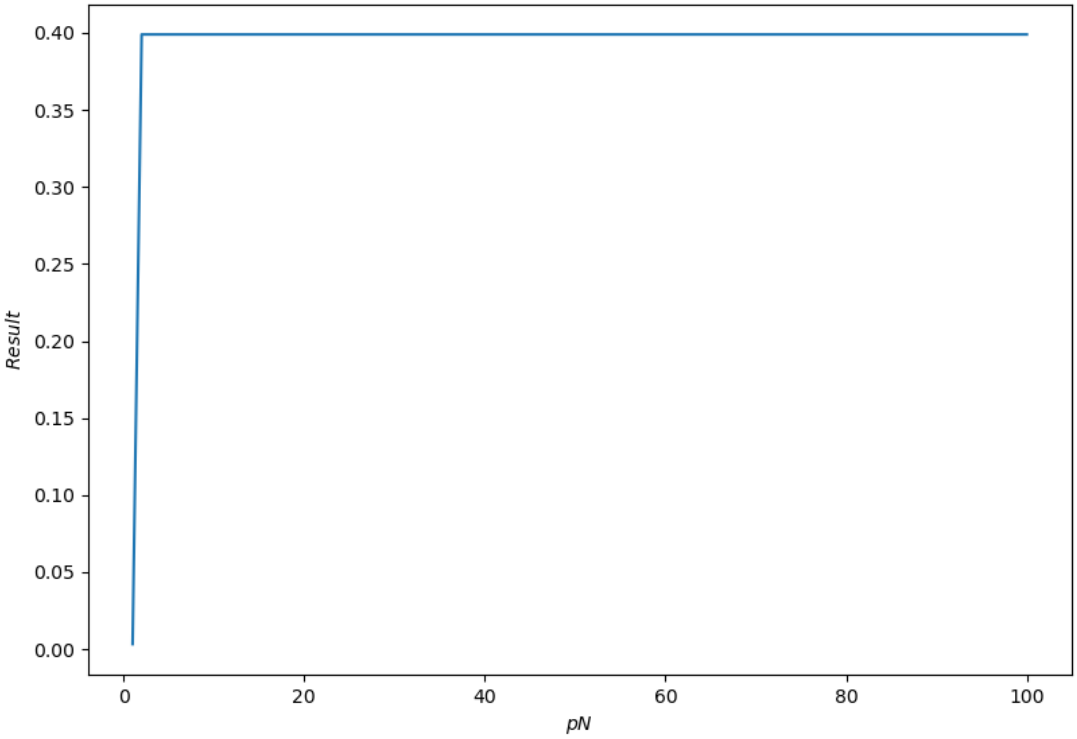
分析



y 轴为所得值，x 轴为迭代数



y 轴为所得值与最终值的差，x 轴为迭代次数



y 轴为所求值，x 轴为群中粒子个数

第一个图象：y(x)=迭代次数为 x 时的所求值，反应了所得值随迭代次数的变化规律
第二个图像：y(x)=迭代次数为 x 时的所求值 减去 迭代次数为 1000 的所求值
第三个图像：y(x)=群粒子数为 x 时的所求值。
以上三个图像反应的规律是：

1. 其他条件一定，迭代次数越大，最终值越趋近一个确定值，越接近最优值。
2. 其他条件一定，群粒子数目越大，最终值越趋近一个确定值，越接近最优值。
3. 本函数较为简单，所以较低的迭代次数和较少的粒子数目就能轻松的求出最终结果。

实验二

在给定的条件下，满足下列公式：

$$P(x_1, x_2) = \frac{1}{2\pi}\exp(-\frac{1}{2}(x_1^2 + x_2^2))$$

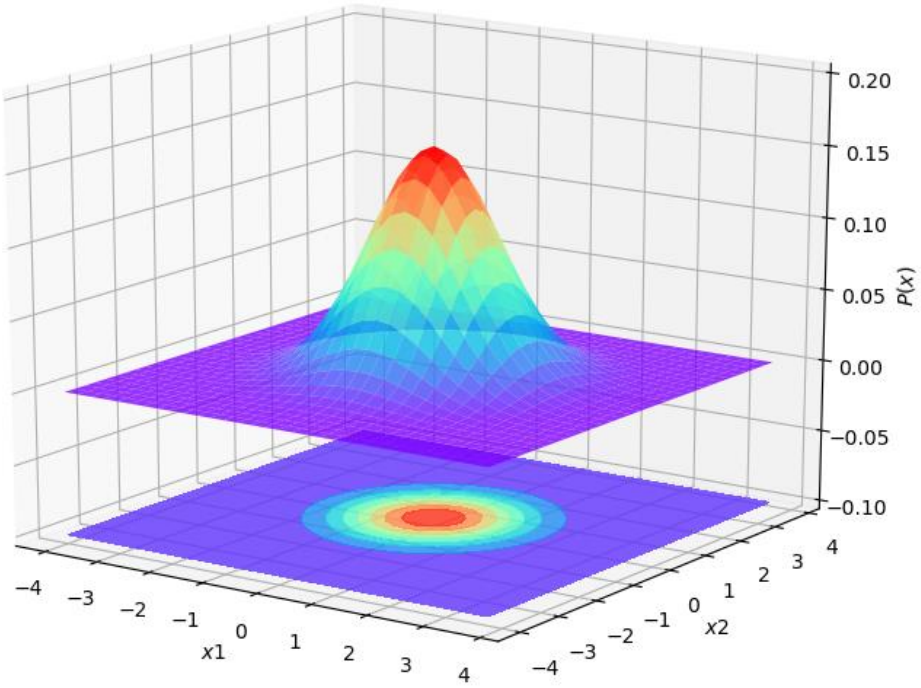
绘制图形

a) python 代码：

```
1. import numpy as np
2. import math
3. import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = Axes3D(fig)
#构建(x,y,z)点集
L = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(L, L)
Z = np.exp((X**2+Y**2)*(-0.5))/(2*np.pi)
#绘制
ax.plot_surface(X, Y, Z, rstride= 1, cstride=1, cmap=plt.get_cmap('rainbow')
,alpha=0.8)
#投影，显示出等高线
ax.contourf(X, Y, Z, zdir='z', offset=-0.1, cmap=plt.get_cmap('rainbow'),alp
ha=0.8)

ax.set_xlabel(r'$x1$')
ax.set_ylabel(r'$x2$')
ax.set_zlabel(r'$P(x)$')
ax.set_zlim(-0.1, 0.20)
plt.show()
```

b）图形如下：



求解

代码与实验一中代码核心部分一致

完整代码请访问：https://github.com/grejioh/PsoReporter

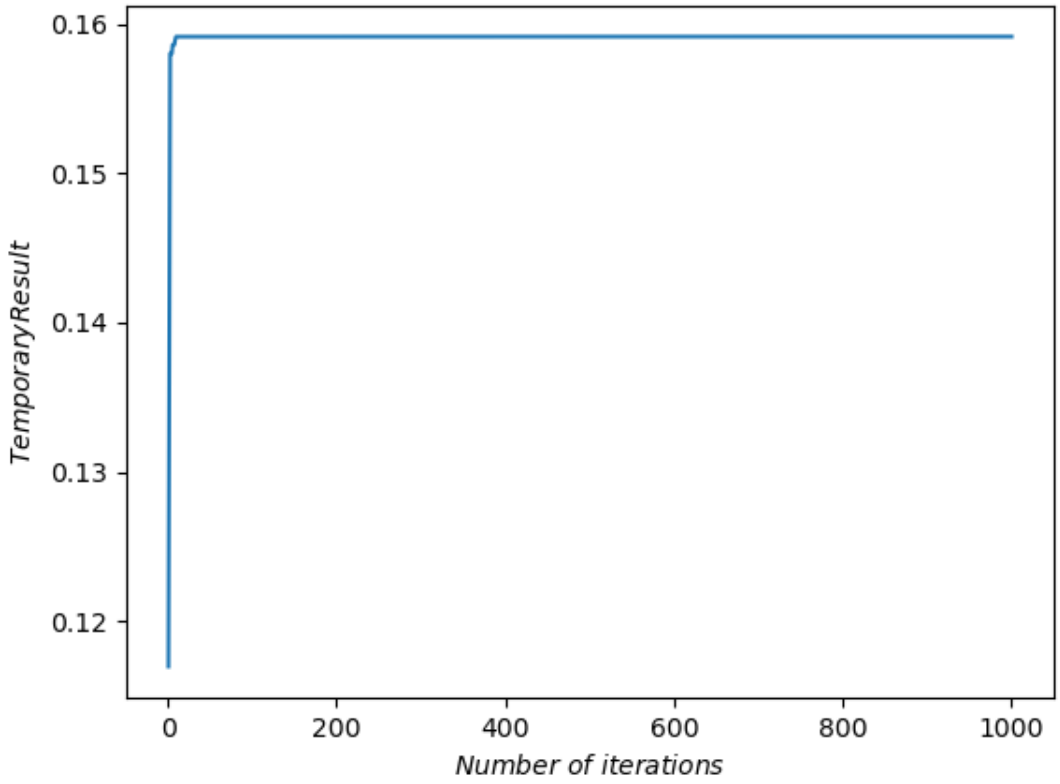
主要修改处如下：

```
###################待求解函数要修改##############################
```

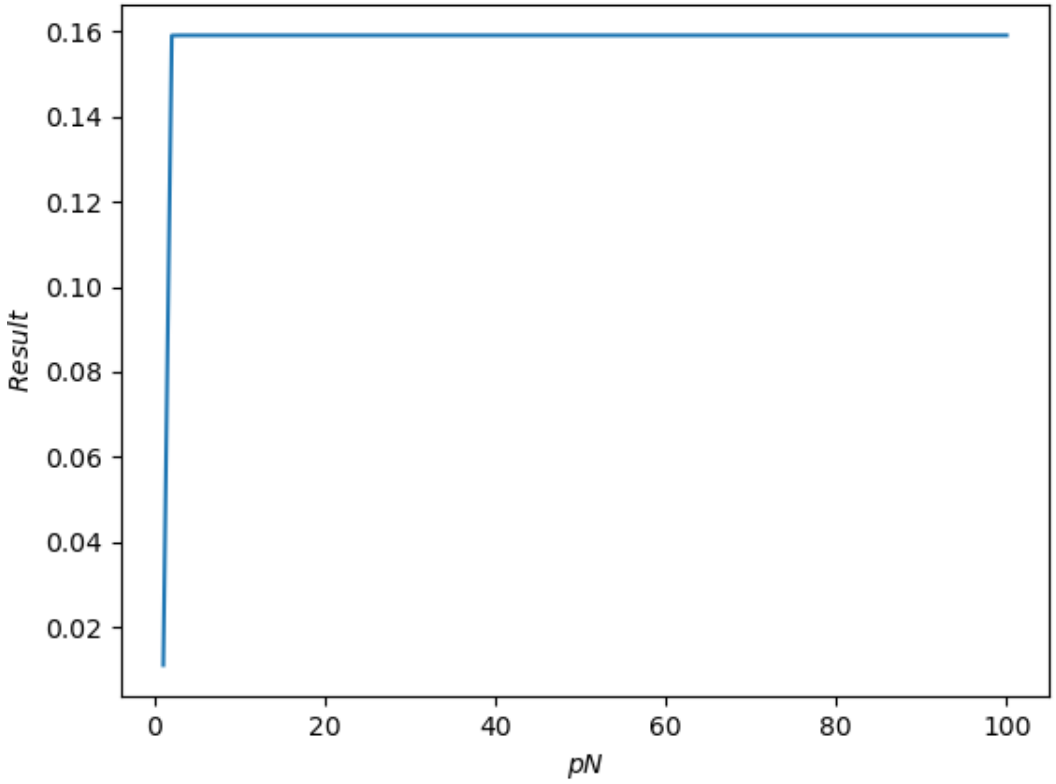
```
def count_func(x):
    return math.exp(-0.5*(x[0]**2+x[1]**2))/(2*math.pi)
####################Pso 中 update 函数要修改#########################
    def update(self):
        fitness = []

        for QAQ in range(self.max_iter): #迭代次数
            for i in range(self.pN):
                temp = self.function(self.X[i]) #获得该粒子位置的函数值
                if(temp > self.p_bestfit[i]):
                    self.p_bestfit[i] = temp
                    self.pbest[i] = self.X[i]
                    if(self.p_bestfit[i]>self.fit): #更新全局最优
                        self.fit = self.p_bestfit[i]
                        self.gbest = self.X[i]
            for i in range(self.pN):
                for j in range(self.dim):
                    self.V[i][j] = min(self.w*self.V[i][j]+\
                            self.c1*random()*(self.pbest[i][j]-self.X[i][j])
+\
                            self.c2*random()*(self.gbest[j]-self.X[i][j]),se
lf.maxv)
                    self.X[i][j] = self.X[i][j] + self.V[i][j]
            fitness.append(self.fit)
        return self.gbest,self.fit
##################print 函数格式要修改############################
```

结果：最优点(0.000000,0.000000) 此处 p(x)=0.159155…

分析：



y 轴为所得值，x 轴为最大迭代数



y 轴为所求值，x 轴为群中粒子个数

第一个图象：y(x)=迭代次数为 x 时的所求值，反应了所得值随迭代次数的变化规律

第二个图像：y(x)=群粒子数为 x 时的所求值。

以上两个图像反应的规律是：

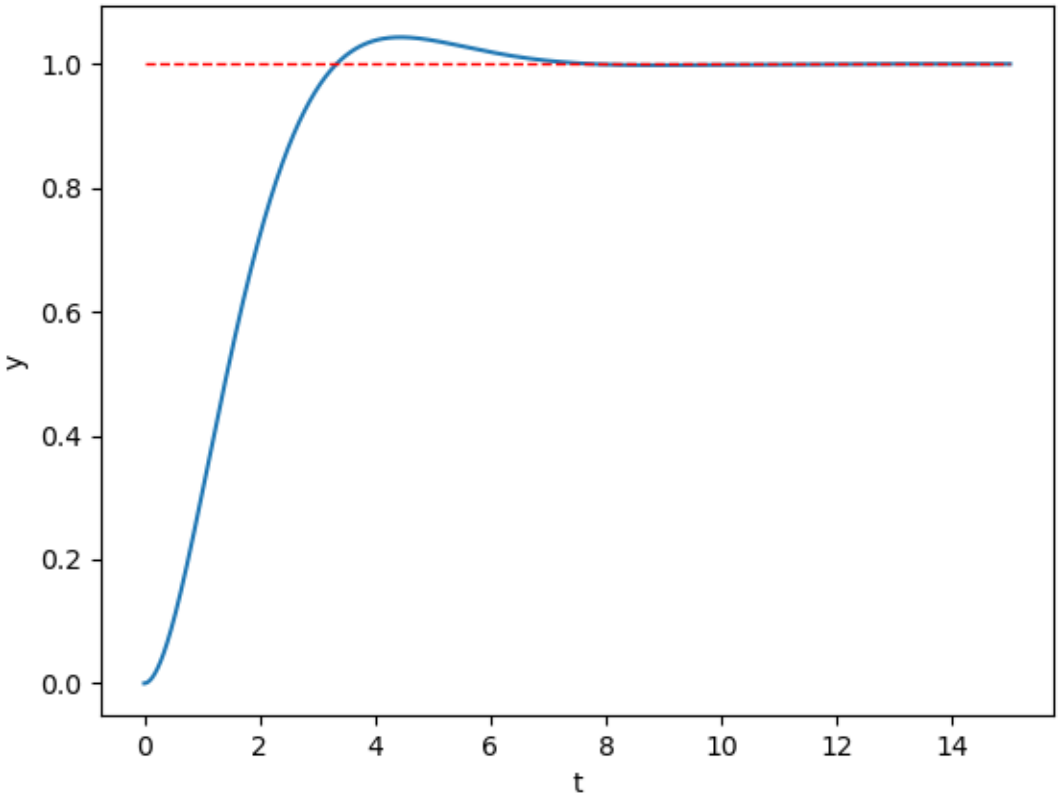
1. 其他条件一定，迭代次数越大，最终值越趋近一个确定值，越接近最优值。（与实验一得出结论相同）
2. 其他条件一定，群粒子数目越大，最终值越趋近一个确定值，越接近最优值。（与实验一得出结论相同）
3. 本函数较为简单，所以较低的迭代次数和较少的粒子数目就能轻松的求出最终结果。

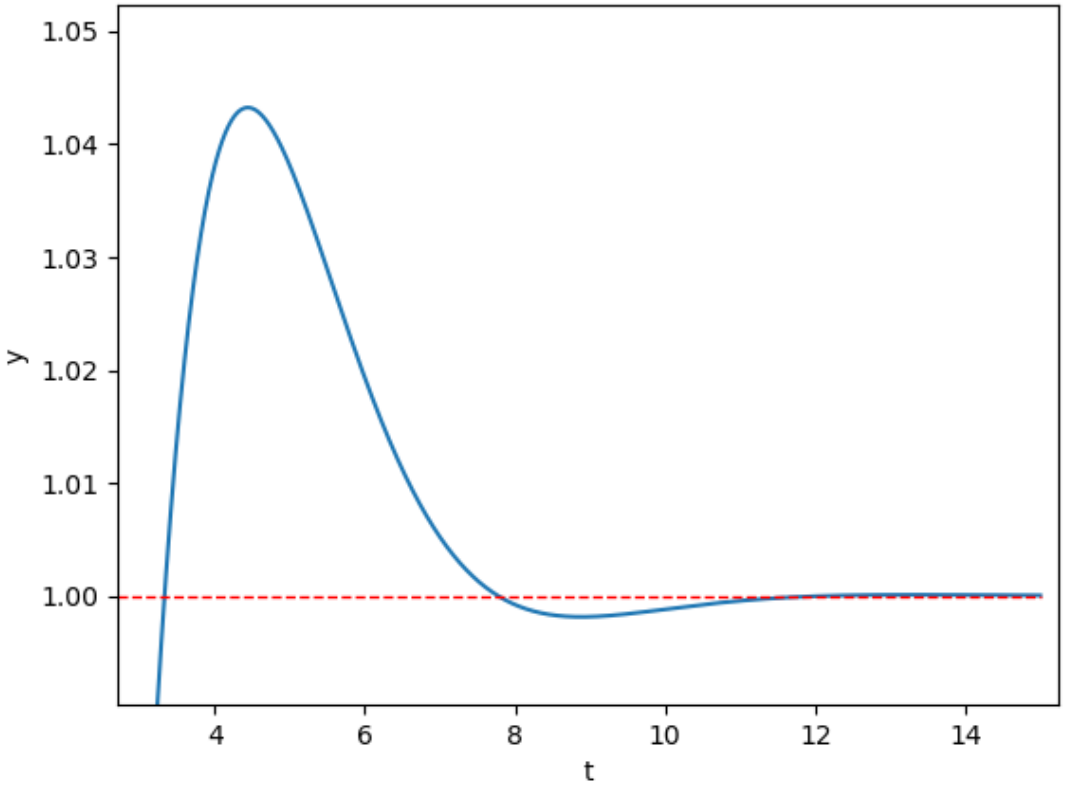
实验一、二都是很简单的函数，都是单峰，且对称性很好，不会出现误导性的第二第三好的点。

实验三

(1)求解 t 在[0,4π]区间二阶欠阻尼系统单位阶跃响应的最大值,假设$\boldsymbol{\omega_n = 1rad/s}$，$\boldsymbol{\zeta = 0.707}$），计算此时系统的超调量，讨论粒子群大小 swamSize 和最大迭代次数 maxgen 对寻优结果的影响。

a) 给定条件下**单位阶跃响应随时间的变化如图所示**





（局部放大）

b) 计算超调量

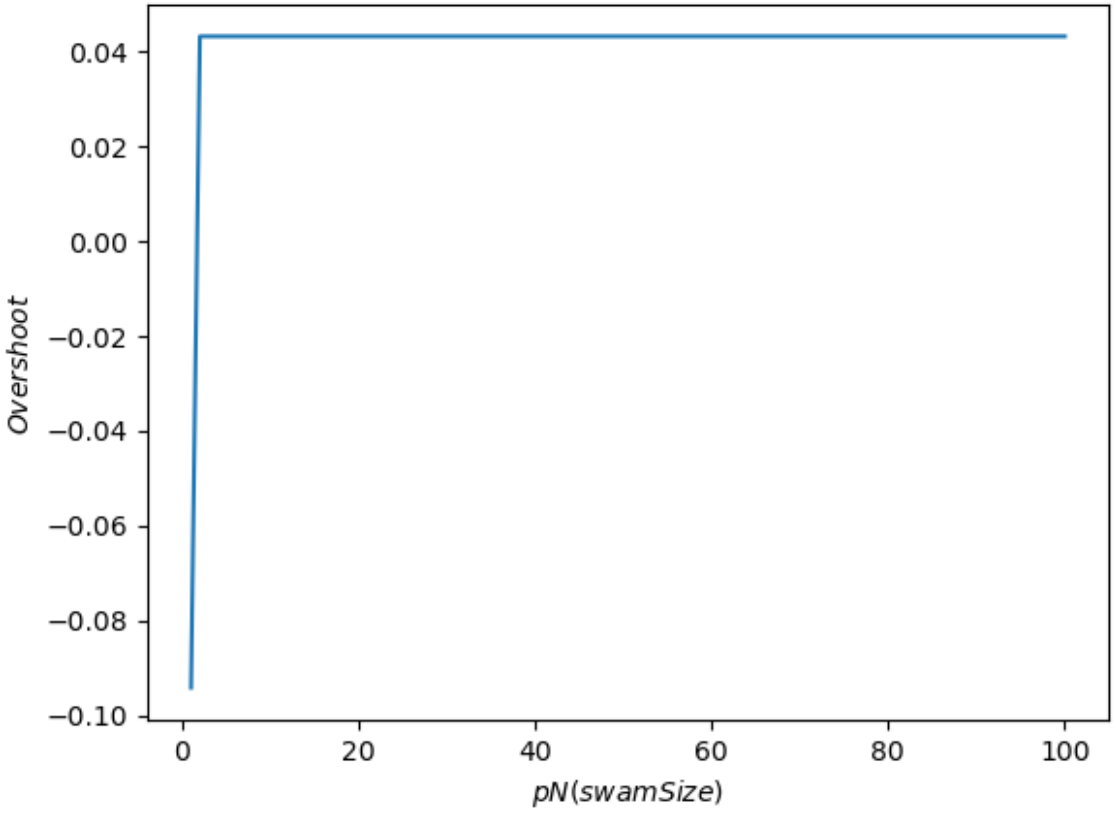
i. 代码，相对于实验一，只需修改函数和搜索范围，详细代码请查看：https://github.com/grejioh/PsoReporter

```
def count_func(x):
 k = 0.707 #阻尼比系数
b = math.acos(k) #阻尼角
wn = 1 #无阻尼振荡角频率
c1 = math.sqrt(1-k**2)
wd = wn*c1 #阻尼振荡角频率
ans = 1-math.exp(-k*wn*x)*math.sin(wd*x+b)/c1
return ans
```

ii.结果 t=4.442212　超调量 0.0432549（迭代数 1000，粒子群大小 100）

c) 分析粒子群大小 swamSize 对寻优结果的影响

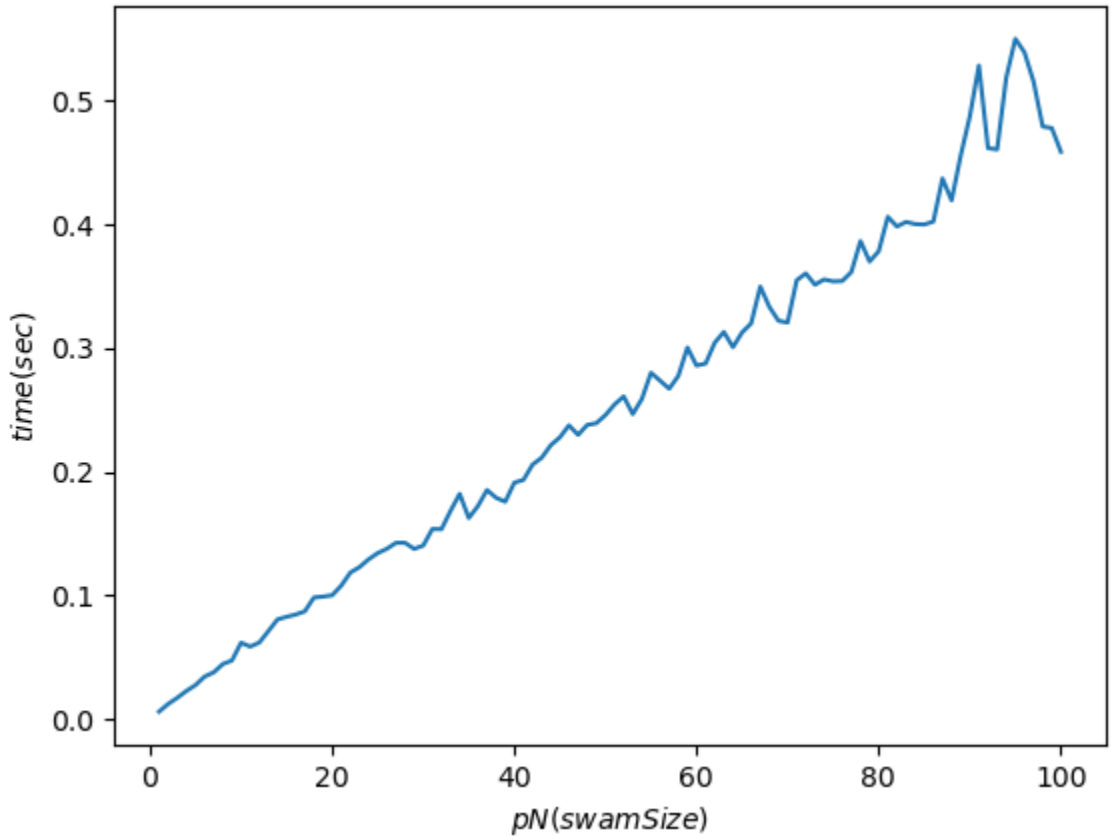
固定迭代数为 300，改变粒子群大小（1 到 100）



超调量随粒子群大小的变化

观察上图不难发现：

1. 其他条件一定时，粒子群越大，所得结果越好
2. 粒子群达到一定大小时，粒子群再增大，寻优结果的变化不大（应该是到达了最优处）



计算时间随粒子群大小的改变

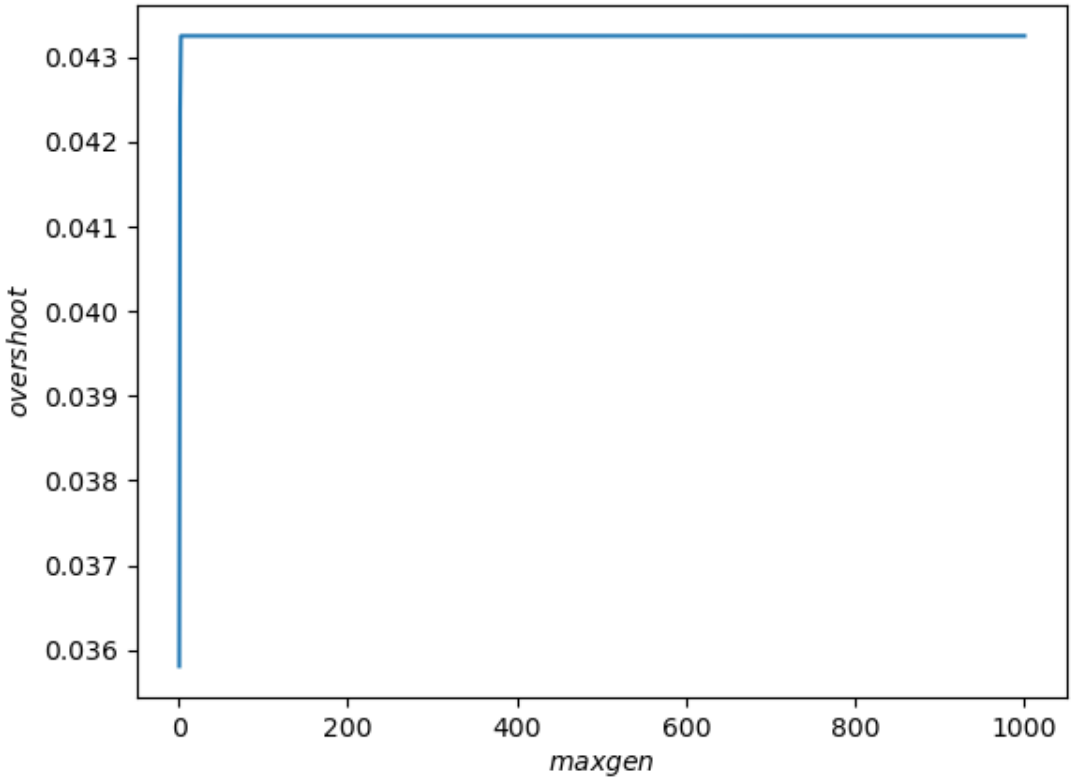
观察上图不难发现:

1. 其他条件不变，计算时间随着粒子群增大而增大
2. 误差允许范围内，计算时间与粒子群有线性关系

结合对两个变量的分析，要兼顾效率和精确性，应选择合适的粒子群大小。

d) 分析最大迭代次数 maxgen 对寻优结果的影响

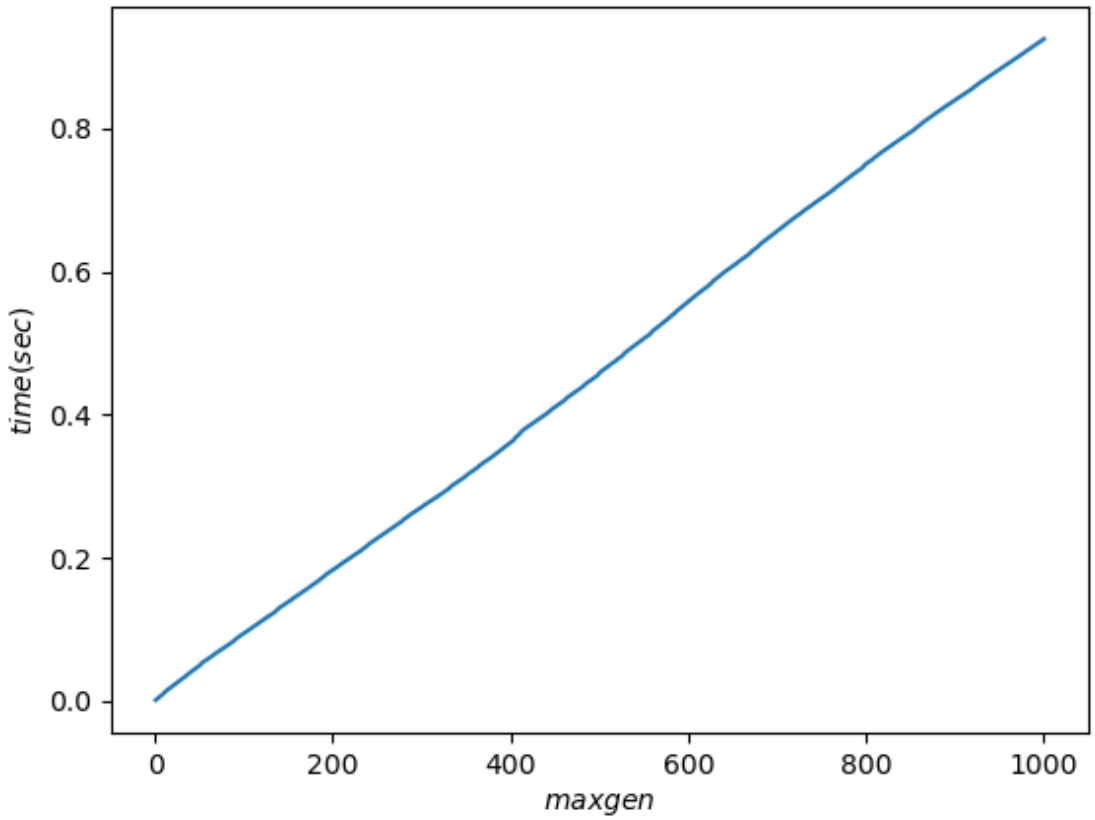
固定粒子群大小为 50，改变迭代次数（1-1000）



观察上图不难发现：

1. 其他条件一定时，迭代数越大，所得结果越好
2. 迭代数达到一定大小时，迭代数再增大，寻优结果的变化不大（应该是到达了最优处）

所求最大超调量随着迭代次数的变化



观察上图不难发现:

1. 其他条件不变，计算时间随着迭代数增大而增大
2. 误差允许范围内，计算时间与迭代数有线性关系

结合两幅图，要兼顾效率和精确性，应选择合适的迭代数。

(2)编程绘制出误差带$\boldsymbol{\Delta = 5\%}$时，阻尼比 ζ（在区间 0≤ζ≤1）与调整时间 ts 之间的关系曲线（三条关系曲线，真实调整时间、包络线调整时间、近似公式）；

真实调整时间 无简单求解公式，我的思路是暴力求解。

包络线调整时间：

$$\omega_n t_s = -\frac{-ln\Delta - ln\sqrt{1-\xi^2}}{\xi}$$

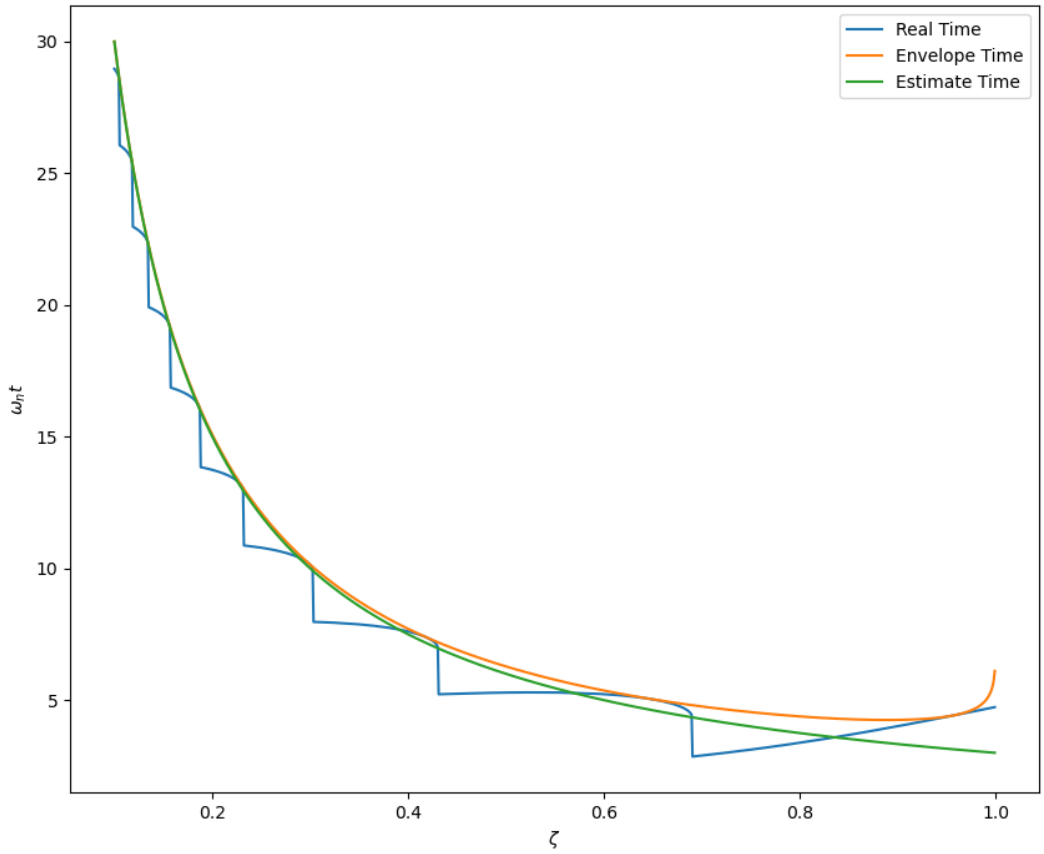
近似公式：

$$t_s \approx \frac{3}{\xi\omega_n}, \Delta = 0.05$$

a) 代码：

```
1. import math
2. import numpy as np
3. import matplotlib.pyplot as plt
4.
5.
```

```
def func(x,k):
    #k = 0.707 阻尼比系数
    b = math.acos(k) #阻尼角
    wn = 1 #无阻尼振荡角频率
    c1 = math.sqrt(1-k**2)
    wd = wn*c1 #阻尼振荡角频率
    ans = 1-math.exp(-k*wn*x)*math.sin(wd*x+b)/c1
    return ans

def realTime(k):
    t = np.linspace(0,30,num=30000) # delta t = 0.001
    seqk = [k]*30000
    y = list(map(func,t,seqk))
    realtime = 0
    ttt = []
    for i in range(1,30000+1):
        if  y[-i]<= 1-0.05 or y[-i] >= 1+0.05:
            realtime=t[-i]
            break
    return realtime

def envelopeTime(k):
    return -(math.log(0.05)+math.log(math.sqrt(1-k**2)))/k
def estimateTime(k):
    return 3/k

def solve():
    k = np.linspace(0.1,0.999,num=1000)
    realtime = list(map(realTime,k))
    envelopetime = list(map(envelopeTime,k))
    estimatetime = list(map(estimateTime,k))
    plt.figure()
    plt.plot(k,realtime,label='Real Time')
    plt.plot(k,envelopetime,label ='Envelope Time')
    plt.plot(k,estimatetime,label = 'Estimate Time')
    plt.legend()
    plt.xlabel(r'$\zeta$')
    plt.ylabel(r'$\omega_nt$')
    plt.show()
solve()
```



b)

(3)粒子群优化算法，以真实调整时间 ts 作为粒子群优化算法的适应度函数（Fitness）当误差带为$\boldsymbol{\Delta = 5\%}$时，优化得到 0≤ ζ ≤1 区间内的最优 ζ 值，绘制出收敛曲线；

a) 代码

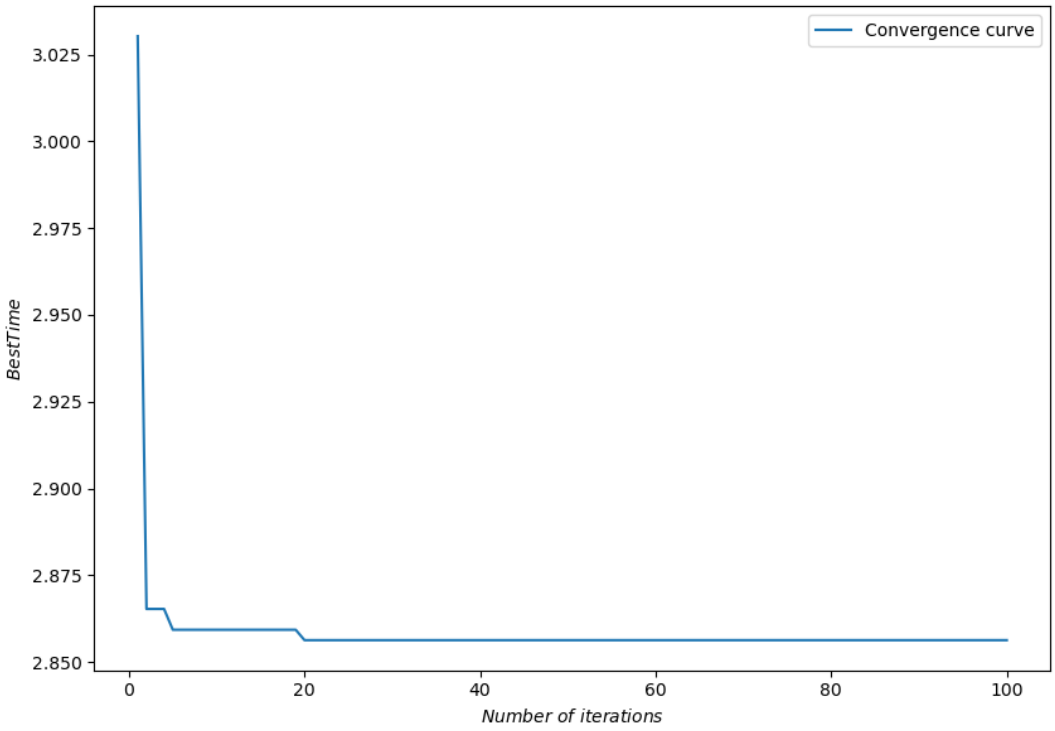
相对于实验一代码，修改如下：（完整代码见：https://github.com/grejioh/PsoReporter）

```
##修改部分如下##
def func(x,k):
    #k = 0.707 阻尼比系数
    b = math.acos(k) #阻尼角
    wn = 1 #无阻尼振荡角频率
    c1 = math.sqrt(1-k**2)
    wd = wn*c1 #阻尼振荡角频率
    ans = 1-math.exp(-k*wn*x)*math.sin(wd*x+b)/c1
    return ans

def realTime(k):
    t = np.linspace(0,30,num=10000) # delta t = 0.001
    seqk = [k]*10000
    y = list(map(func,t,seqk))
    realtime = 0

```

```
    for i in range(1,10000+1):
        if  y[-i]<= 1-0.05 or y[-i] >= 1+0.05:
            realtime=t[-i]
            break
    return realtime

def count_func(x):
    return -realTime(x[0])

def solve():
    iternum = 100
    pso = Pso(pN = 50,dim = 1,max_iter = iternum, func = count_func)
    pso.init_pop()
    x_best,fit_best,fitness= pso.update()
    print("bestZeta={:.6f}  bestTime={:.6}".format(x_best[0],-fit_best))

solve()
```

结果：bestZeta=0.690150 bestTime=2.85629

b) 收敛曲线如下图：



小组分工

实验报告，演示文档制作：郭家豪 19020100298 电子工程学院

课程学习体会与收获

1. 算法很漂亮，是受到自然界真实发生的生命活动的启发而设计出来。即使是由一行行代码、一串串数字组成的学科也是离不开自然界的启发的。
2. 实践能很好地促进学习。这次实验遇到了很多困难，但是最终通过种种办法克服了。看着这个长长的实验报告，很有成就。
3. 绘图和算法实现选择了 python，而没选 matlab。都不太熟悉，python 功能也很丰富，但是 matlab 更加系统，以后可能还要好好学习 matlab。
4. 一些问题：
   a) 对于 word 的排版不太熟悉，操作起来很僵硬，还需学习，或者找到一个更好的工具（也许是 latex）。
   b) 对 python 不够熟悉，很多实现都比较啰嗦。

参考文献和感谢

1. 对粒子群优化算法的主要了解来自（https://en.wikipedia.org/wiki/Particle_swarm_optimization）
2. 算法的具体实现最初参考了**粒子群算法的 python 实现**,虽然发现了一些问题，但是还是表示感谢。(https://uzzz.org/2019/08/02/795485.html)
3. 涉及的二阶欠阻尼控制系统的一些专业知识，感谢吴老师在课堂上的讲解。