高压油管的压力控制——传统优化控制论与强化学习之战

摘要

高压油管是许多燃油发动机的重要零部件，燃油通过进入和喷出高压油管，使得燃油的压强发生改变，从而影响燃油发动机的供油量，从而影响发动机的工作效率。然而，燃油进入和喷出过程的间隔可以导致油管内的压强发生变化，因此高压油管是依赖于油管内压强的变化来工作的。合理地控制油管内的压强，使得高压油管可以保持在一定的恒定值或者合理地改变压强值，是本文研究的主要内容。

对于整个题目来说，对于一个有关于高压油管的模型，我们首先需要对整个物理系统进行动力学建模，首先对于题中不同的高压油管和控制部件采用隔离法建模分析其状态、整体法合成整个物理模型的实时状态，利用编程进行仿真油管的状态。然后再设计控制算法对几个控制部件进行控制已达到我们想要达到的状态与目的：本质上仍然是一个优化问题。

对于问题一来说，我们可进行控制的部件比较少：只有 A 口一个单向阀。但是对于 B 口的喷油嘴，他的运行状态是未知的，因此我们需要对其的状态进行分类讨论：根据工作的间隔进行分割。在分类讨论之后，我们需要将我们的两个控制目的：保持油管体系气压稳定在$100\ \ Mpa$与在时间内调整油管体系的气压，转化成我们的已知控制目标，转化成目标函数，再选择优化问题的求解算法。最终我们几次求解优化问题中发现了规律，提出了一种周期性控制策略完成了在给定时间内转换成任意压强值之内的任务，同时也解决了题目的问题。

对于问题二来说，相较于问题一最大的区别就是我们在第一问使用的控制部件：A 口单向阀的控制方式发生了改变：从单纯的控制 A 口的开关变成了通过角速度来控制油泵的气压，通过油管和油泵的气压差控制开关。我们仿照第一问的周期性策略，重新对整个策略进行了更新，给出了保持压强在 100Mpa 内的周期角速度。

对于第三问来说，我们可进行控制的部件增加了，并且这个时候问题要求我们给出喷油与供油两种策略，于是这个时候我们就要具体考虑到喷油嘴的实际模型。对于喷油嘴的物理模型来说，很多参数尚未知晓，所以先利用喷油嘴已知的工作曲线和数据进行模型参数的求解，然后再进行规划问题的求解。但是问题过于复杂，参数太多，于是我们引入入强化学习（Reinforcement Learning, RL）对整个环境进行建模优化，意图更好的控制结果，并对新兴方法与传统方法进行一个对比。

关键词：高压油管、物理方程建模、遗传算法、强化学习、DQN

目录

一、问题重述

高压油管是内燃机中的一个重要部件，通过控制燃油进出高压油管的间歇性过程，我们能够控制高压油管内压力的变化，使得影响喷油量出现偏差，进而影响发动机的工作效率。

下图给出了某高压燃油系统的工作原理，燃油经过高压油泵从 A 处进入高压油管，再由喷口 B 喷出。燃油进入和喷出的间歇性工作过程会导致高压油管内压力的变化，使得所喷出的燃油量出现偏差，从而影响发动机的工作效率。

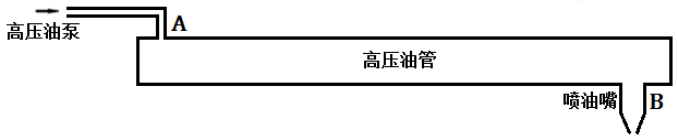


图 1 高压油管示意图

待研究的问题

问题一：

某型号高压油管的体积参数已知，外部以一恒定压力 160MPa 通过单向阀开关 A 向油管内供油，而喷油器每秒工作 10 次，工作曲线如图 2 已知，若将高压油管内的压力尽可能控制在 100MPa，应当如何设置单向阀的开启时长？若在 2s，5s 和 10s 内将油管内的压力从 100MPa 调整后稳定在 150MPa，应如何调整单向阀开启时长？

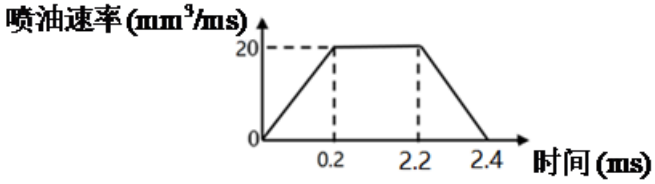


图 2 喷油速率示意图

问题二：

在问题一的基础上，外部供油由恒定压强变为由高压油泵控制压强，而高压油泵通过凸轮驱动柱塞压油，如图 3 所示。当柱塞腔内的压力大于高压油管内的压力时，单向阀 A 开启，燃油进入油管。当柱塞在最高点时，柱塞腔残余容积已知。当柱塞运动至最低点时，低压燃油充满柱塞腔，压力已知。同时喷油嘴结构已知，自此条件下控制角速度，使高压油管内的压力尽量稳定在 100MPa 左右。

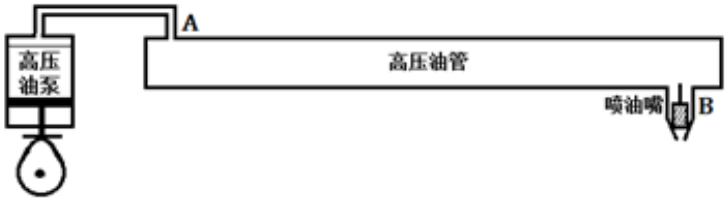


图3 高压油管实际工作过程示意图

问题三：

如图 4 所示，在问题二的基础上，再加一个喷油规律相同的喷油嘴，这是喷油与供油策略应如何调整？若再安装一单向减压阀，使高压油管内的燃油可以回流到外部低压环境中，此时应如何调整喷油供油策略？

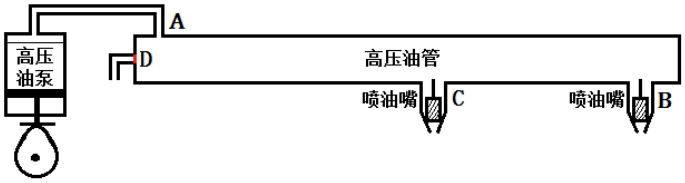


图5 具有减压阀和两个喷油嘴时高压油管示意图

二、问题分析

对于整个题目来说，对于一个有关于高压油管的模型，我们首先需要对整个物理系统进行动力学建模，首先对于题中不同的高压油管和控制部件采用隔离法建模分析其状态、整体法合成整个物理模型的实时状态，利用编程进行仿真油管的状态。然后再设计控制算法对几个控制部件进行控制已达到我们想要达到的状态与目的：本质上仍然是一个优化问题。

对于问题一来说，我们可进行控制的部件比较少：只有 A 口一个单向阀。但是对于 B 口的喷油嘴，他的运行状态是未知的，因此我们需要对其的状态进行分类讨论：根据工作的间隔进行分割。在分类讨论之后，我们需要将我们的两个控制目的：保持油管体系气压稳定在$100\ \ Mpa$与在时间内调整油管体系的气压，转化成我们的已知控制目标，转化成目标函数，这个时候该优化问题的求解算法以及目标函数的设计也值得研究与讨论。

对于问题二来说，相较于问题一最大的区别就是我们在第一问使用的控制部件：A 口单向阀的控制方式发生了改变：从单纯的控制 A 口的开关变成了通过角速度来控制油泵的气压，通过油管和油泵的气压差控制开关。注意到这里虽然提出了喷油嘴的实际情况，但是我们并不能控制它，为了模型简化我们仍然使用喷油嘴已知的工作曲线（图 2）来代替具体模型。

对于第三问来说，我们可进行控制的部件增加了，并且这个时候问题要求我们给出喷油与供油两种策略，于是这个时候我们就要具体考虑到喷油嘴的实际模型。对于喷油嘴的物理模型来说，很多参数尚未知晓，所以先利用喷油嘴已知的工作曲线和数据进行模型参数的求解，然后再进行规划问题的求解。

对于整个问题来说，该问题的是一个经典的控制问题（PID）,但同时来说对于现在存在的策略和我们所讨论的传统优化来说，效果不好：尤其是前两问对于 A 口工作状态

的分类讨论中。本文在将最后试图引入强化学习（Reinforcement Learning, RL）对整个环境进行建模优化，意图更好的控制结果，并对新兴方法与传统方法进行一个对比。

三、模型假设

1. 假设进油口打开后油气就可以迅速进入管中，出油口打开后油气就可以迅速被喷出，不考虑油口开启和关闭时的时间对流量的影响。
2. 假设油气在进入高压油管后迅速气化且以充满整个高压油管的方式流动。
3. 假设油气在进入高压管后，可以迅速与高压管内的气体混合均匀，在每一个时刻下，高压管内的气体都处于平衡状态。
4. 所有过程都不计摩擦。

四、符号与约定

序号	参数	意义
1	P_{pip}	油管内的压强（MPa）
2	ρ_{pip}	油管内燃油的密度（mg/mm^3）
3	V_{pip}	油管的体积（mm^3）
4	$m_{pip}^{t} / m_{pip}^{t-1}$	T 或 t-1 时刻油管内油气的质量（mg）
5	P_{pu}	油泵内的压强（MPa）
6	m_{pu}	油泵内的油气质量（mg）
7	Q_A / Q_B	A 或 B 口高压油管的流量（mm^3 / ms）
8	E	弹性模量
9	d	直径（mm）
10	A	小孔的面积（mm^2）

五、模型的建立与求解

5.1 问题一：高压油管的物理模型

问题一中使用的油管的简化模型如图 5.1-1 所示。

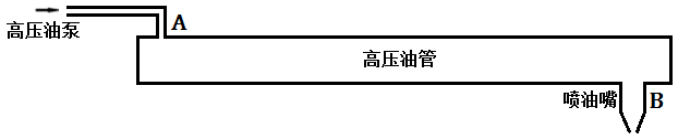


图 5.1-1 高压油管简化模型

其中，高压油管的内腔长度为 500mm，内直径为 10mm，供油入口 A 处小孔的直径为 1.4mm，通过单向阀开关控制供油时间的长短，单向阀每打开一次后就要关闭 10ms。喷油器每秒工作 10 次，每次工作时喷油时间为 2.4ms。

对于高压油管进行隔离法分析，我们可以得到其状态满足如下微分方程：

$$\frac{dP_{pip}}{dt}=\frac{E_{pip}}{\rho_{pip}}\frac{d\rho_{pip}}{dt}$$

其中，P_{pip} 是指高压油管的实时压强（Mpa），ρ_{pip} 是指油管内燃油的实时密度（mg/mm3），E_{pip} 则是在高压油管实时压强下燃油的弹性模量。

燃油的实时密度可以由以下差分方程进行计算：

$$\rho_{pip}{}^{t}=\frac{m_{pip}{}^{t-1}+sign(A)\cdot Q_A\triangle t\rho^{t-1}{}_{pu}-sign(B)\cdot V^{\triangle t}{}_B\rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q_A=CA_A\sqrt{\frac{2\Delta P_A}{\rho_{pu}{}^{t-1}}}$$

$$V_B=vt=S_{cruve}$$

其中，Q_A 是 A 高压油管的流量（mm^3/ms），$\triangle t$ 是差分方程递推时的步长，$V^{\triangle t}$ 是在 $\triangle t$ 时间段中 B 口流出的燃油体积，其由 B 口喷油器的工作曲线图（图 2）的时间段积分决定. $sign(x)$ 则是一个指示函数，用来判断在该时间段内，x 部件（A 口油泵或者 B 口喷油嘴）是否工作，其公式如下：

$$sign(x)=\begin{cases}0 & x\text{部件不工作}\\ 1 & x\text{部件工作}\end{cases}$$

5.2 问题一：燃油密度和弹性模量与压强的关系

我们发现如果要进行 5.1 中物理模型的仿真，我们需要煤油在 160 Mpa 的密度和与弹性模量和压强的关系。注意到由于高压油管内的气体已经经过压缩，不再满足理想气体假设下的弹性模量与压强的线性关系，此时气体应该遵从昂内斯方程：

$$P = RT(\frac{1}{V_m} + \frac{B}{V^2{}_m} + \frac{C}{V^3{}_m} + \cdots)$$

该方程为一组无穷级数的求和，考虑到计算量与近似问题，我们决定采用多项式近似的方法进行附件三中数据（如图 5.2-1）的拟合，选择多项式为四次方多项式。

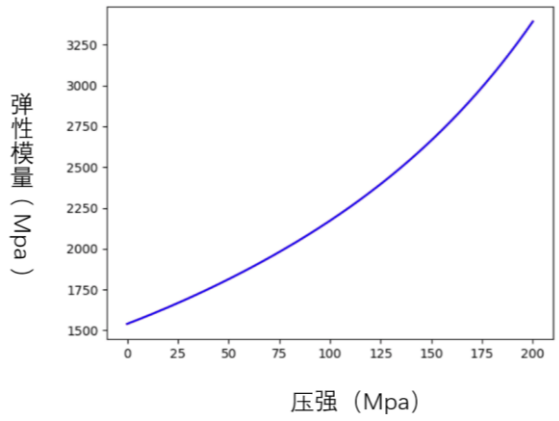


图 5.2-1 煤油弹性模量与压强的关系（附件三）

使用 Python 的 Scipy 库拟合结果的方程为 $3.454177032715454\times10^{-7}p^4-3.8129329087309995\times10^{-5}p^3+0.016667115687673203\,p^2+4.687866549677347p+1539.645780763744$

此次拟合的可决系数 $R^2=0.999$，图 5.2-1 显示了该方程的拟合偏差，可以看见方程值和预测值的差别在 ±1 之间。

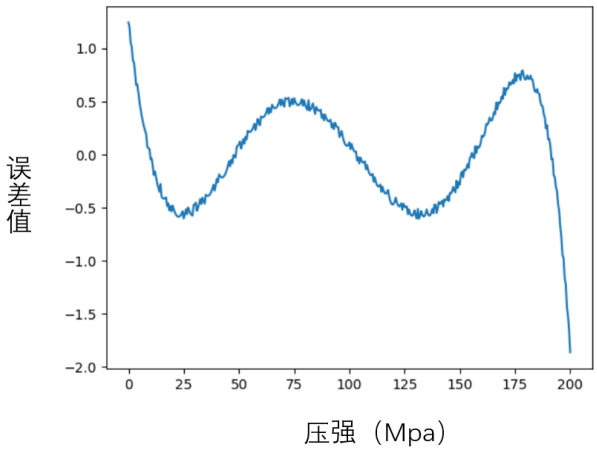


图 5.2-2 拟合方程的预测值与真实值的误差分析

对于煤油密度与压强的关系，我们采用在 5.1 中得到的微分方程的另一个形式：

$$dP_{pip} = \frac{E_{pip}}{\rho_{pip}} d\rho_{pip}$$

利用分离变量法变形为:

$$\frac{1}{E_{pip}} dP_{pip} = \frac{1}{\rho_{pip}} d\rho_{pip}$$

两边求不定积分有:

$$\int \frac{1}{E_{pip}} dP_{pip} = \ln(\rho_{pip}) + C$$

在上面的操作中我们已经拟合求得弹性模量与气压的关系，接近于 1 的可决系数说明气压的变化可以解释 99.9%的弹性模量的变化：我们认为弹性模量的变化置于压强的变化有关。利用 Matlab 的符号积分系统，我们可以求得原函数以及关系，太过于复杂这里不进行罗列，会在附录中给出。图 5.2-3 展示了煤油的密度随着气压变化的关系。

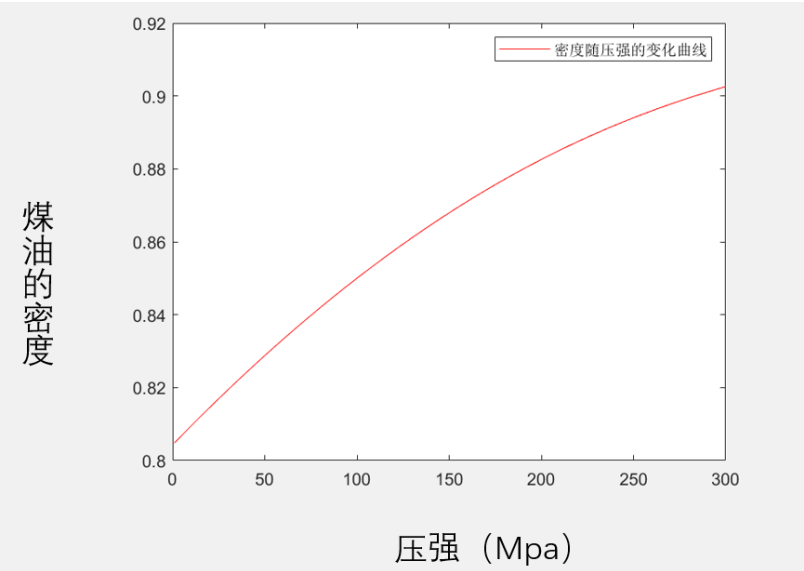


图 5.2-3 煤油密度与压强的关系

从图中我们可以获得：160 Mpa 下煤油的密度为 0.8711 mg/mm^3.

5.3 问题一：物理模型的离散化与仿真

在前两节的铺垫下，我们将对 5.1 建立的物理模型进行离散化，为模拟打下基础。离散化后的模型如下。

$$\begin{cases} P^{t}{}_{pip} - P^{t-1}{}_{pip} = \dfrac{E^{t-1}{}_{pip}}{\rho^{t-1}{}_{pip}}(\rho^{t}{}_{pip} - \rho^{t-1}{}_{pip}) \\ \rho_{pip}{}^{t} = \dfrac{m_{pip}{}^{t-1} + sign(A)\cdot Q_A \triangle t \rho^{t-1}{}_{pu} - sign(B)\cdot V^{\triangle t}{}_{B}\rho^{t-1}{}_{pip}}{V_{pip}} \\ Q^{\triangle t}{}_{A} = CA_A\sqrt{\dfrac{2\Delta P_A}{\rho_{pu}{}^{t-1}}} = CA_A\sqrt{\dfrac{2(P^{t}{}_{A} - P^{t-1}{}_{A})}{\rho_{pu}{}^{t-1}}} \\ V^{\triangle t}{}_{B} = vt = S_{cruve} = \dfrac{1}{2}\triangle t(S(t) + S(t-1)) \end{cases}$$

于是对该物理系统的递推仿真就成了不断求解非线性方程组的一个过程，为了仿真的精准性，递推补步长取 $\triangle t = 0.1ms$ 。 该仿真系统的初值如下：

$$\begin{cases} P^{0}{}_{pip} = 100 \\ m_{pip}{}^{0} = 33362.5 \\ \rho^{0}{}_{pip} = 0.85 \end{cases}$$

5.4 问题一：如何保持高压油管的压强和改变油管气压

现在我们来讨论关于高压油管气压控制的管控模型，我们首先来分析一下该高压油管的动态：高压油管的内部状态就像一个水池：A 管进水，B 管出水。在上面几节中我们已经讨论了 A 口的状态，现在我们对 B 口进行一个分析。

题目中现在对于 B 口的信息严重不足，我们并不知道它的运行模式，我们只知道它一秒钟能够运行 10 次，除此之外再无信息。于是我们需要对其的运行模式进行分类。我们假设这样一种情况：如果两次喷油嘴的运行时间间隔足够的长，长到大于 A 口单向阀运行的休息时间：10ms，也就是说我们可以通过控制 A 口周期性运转来调节整个高压油管的气压。于是我们的分类就变成如下两种：

5.4.1 喷油嘴的运行模式为时间分布均匀地运行

首先我们讨论怎么在**这个 100ms 的周期里面稳定整个高压油管的体系压强保持在 100Mpa**。在这种模式下，喷油嘴的运行模式就是每间隔 100ms 运行一次，我们则可以通过周期性打开 A 口单向阀来调节整个高压油管内的气压。

在 5.1 ~ 5.3 中我们讨论了整个物理体系的递推模式，接下来我们将构造一个优化

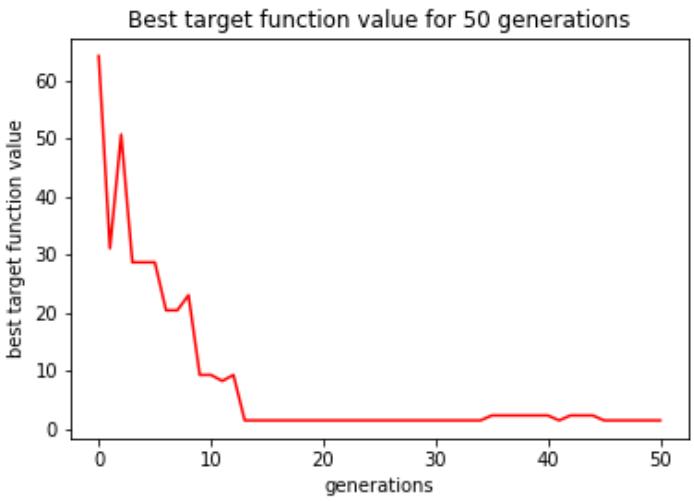
问题来求解。

我们需要求解的参数时 A 口的开关状态，我们用以下两个参数来描述这个状态：t_{open}、t_{last}；t_{open} 表示 A 口打开的时间点，t_{last} 表示 A 口打开状态持续保持的时间。对于这种长周期的分类情况下，我们认为一次的打开就足以平稳整个体系的气压。

由 t_{open} 和 t_{last} 我们可以推断出物理模型中的 $sign(A)$ 函数，由喷油嘴时间均匀分布的运行模式我们可以 tuiduan 物理模型中的 $sign(B)$ 函数。于是需要把气压稳定在 100 Mpa 的目标我们可以描述成如下函数：

$$\min \sum (P_{pip}^{t} - 100)^2$$

我们选择遗传算法（GA）来求解此优化问题的解，传递 50 代，每一代的群体书为 10 个。通过多次求解，我们得到 $t_{open} \approx 0$，$t_{last} \approx 2.7ms$，说明这是一个稳定解。算法的收敛过程如图 5.4.1-1 所示。



5.4.1-1 算法收敛时间

整个高压油管体系的气压变化如图 5.4.1-2 所示。

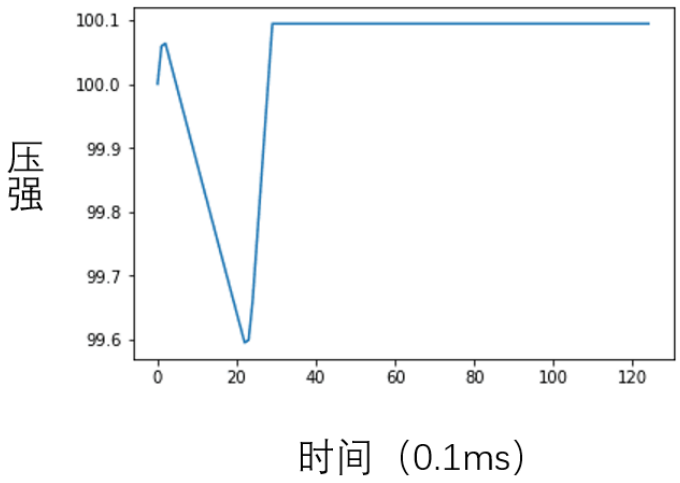


图 5.4.1-2 高压油管压强变化图

从上图中我们可以看见，体系的压强可以很快保持在 100 Mpa 之间，说明我们的算法确实有效。

接下来我们讨论**怎么在固定时间内将体系内的压强进行调整**。题目要求我们在 2s、5s、10s 内通过调整 A 口单向阀的开关将高压油管体系从 100Mpa 提高到 150Mpa，这个时候我们通过将优化目标调整到如下形式：

$$\min \sum (P_{pip}^{t} - 150)^2$$

对于这个问题，我们认为在规定规定时间内 A 口的进气阀可以多次开启。这个时候求解的参数仍然是 t_{open}、t_{last}，但是是多个开启时刻的 t_{open}、t_{last}，总体采用冗余编码制度，求解的参数空间为 $\frac{2000t}{\triangle t}$ 个参数。仍然使用遗传(GA)算法进行求解，求解出三个不同要求时刻的调节策略下体系压强的变化值如图 5.4.1-3 所示。

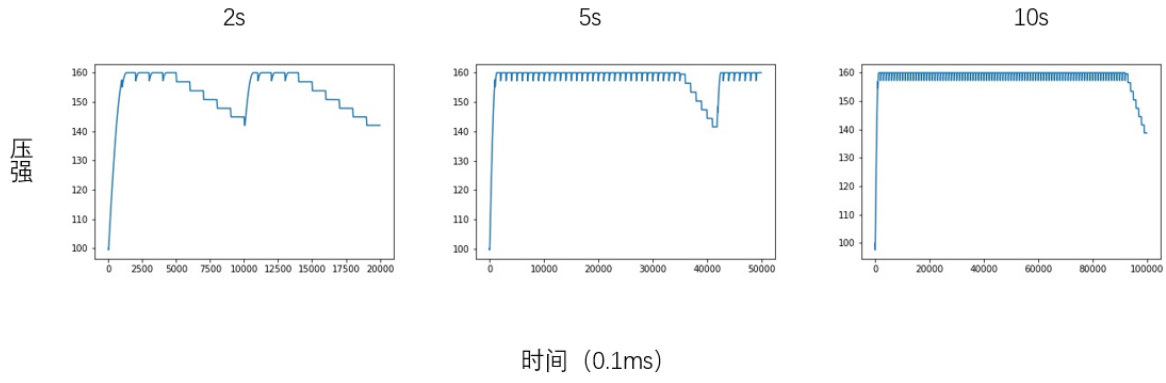


图 5.4.1-3 体系压强的变化值

我们现在分析一下上面模型生成的几个结果：所有的体系都在初期的一段时间内提升到了 160 Mpa，于是我们推断，之前的周期（100ms）其实可能也是足够的：对于长周期内的气压调整，如果想只通过一次达到固定的气压值，我们可以通过先通过一次打开气压阀来达到。于是通过上面的三个结果我们发现，160Mpa 大概在 160ms 时达到，于是我们调整优化周期的长度为 200ms，仍然使用只打开一次的 A 口单向阀门来实现。于是我们获得以下结果：

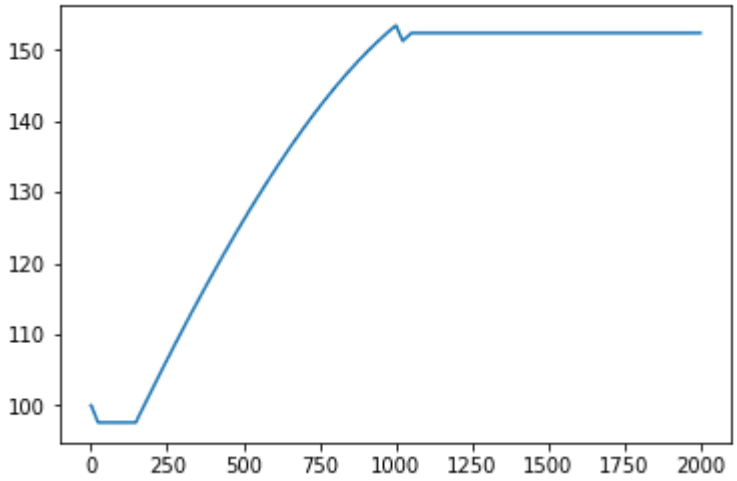


图 5.4.1-3 体系压强的变化值

上图结果是在 0.3ms 的时候打开 A 口气阀，并且持续打开 89ms 的结果。

因此我们最终得出如下策略：

1. 在前 200ms 内我们使用一次 A 口单向阀开关提高到我们想要的任意压强（如 150Mpa）：我们之前的结果已经证明说我们可以在 200ms 内提升到 160Mpa。
2. 后面剩下的时间内我们稳定气压值(这个在前一个稳定气压的问题中已经证明)：在每一次喷嘴口打开的时候，我们同时打开 A 口单向阀固定毫秒稳定整体气压。

最后我们检验一下在短周期内，根据图 5.4.1-2 的结果应该是 12.5ms 之内，通过只打开一次阀门可以稳定任何水平下的气压，结果如图 5.4.1-4 所示，该图描述了在上述策略中的第二步，面对不同目标调整气压时，我们所需要打开 A 单向阀门的时间。

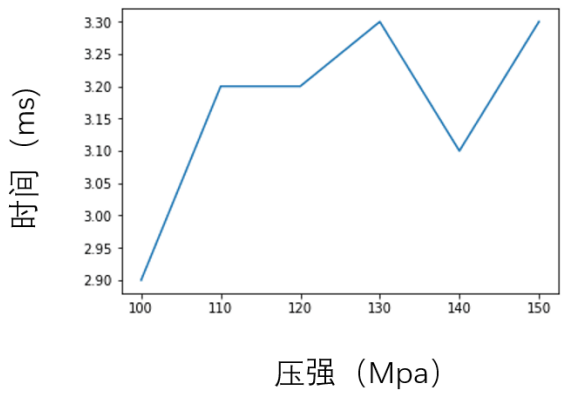


图 5.4.1-4 步骤 2 中不同气压打开 A 阀门的时间

这个图也很好理解，后面的需要打开的时间趋于稳定是由于高压油管的气压接近160Mpa，两边的差值已经略小，不会有太大的变动了由于阀门的打开。

5.4.2 喷油嘴的运行模式为非均匀的时间分布运行

上一段我们对运行时间为均匀时间分布的探索让我们对非均匀的时间分布运行的喷油嘴的条件下，做出一下集合的划分来达到我们的要求。

我们根据考量喷油嘴运行模式的分布问题来进行分类：

1. 如果在整个运行过程中由出现一下情况在三个喷油嘴启动的时间间隔大于我们之前设定的 200ms 的话，那么我们一定能通过之前提出的策略来达到我们的目的气压值。剩下的时间内如果出现喷油嘴运行则利用第二步骤的曲线进行气压稳定的操作，周期性控制住油管的气压。

2. 如果他们之间的间隔小于 200ms 的话，我们就通过划分每一个部分气压的升高至来进行气压的提升：我们首先根据时间间隔从大到小进行喷油嘴之间的排序，然后选择前面的几个的时间间隔之和大于 200ms 的进行气压的上升操作，而剩下的时间点则是选择稳定气压的模式进行气压阀的打开于关闭。

总体来说，在分均匀分布的情况下，我们相当于做了如图 5.4.2 -1 的操作，把 图 5.4.1-3 中的曲线按照时序行分开了。

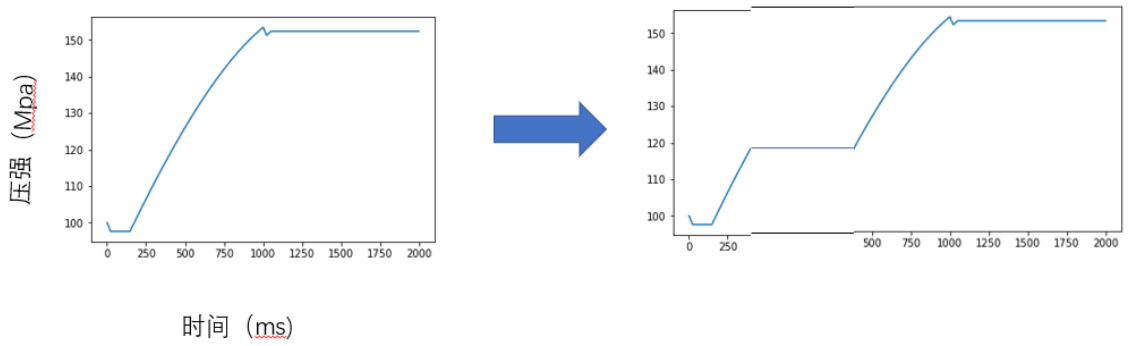


图 5.4.2 -1 时序上分割的算法

优化的算法则变成以下过程：

1. 首先按照排序选择好周期，并且将各个选择的位置按照时间前后拼接起来，构成如图 5.4.2-1 左侧的连续结构，进行优化，优化函数和之前一致。

2. 根据不同的时间分割点，求出在连续状态下应该达到压强值，然后分开进行各自的优化处理。也即从图 5.4.2-1 左侧的图转化为右侧的图。由于

总而言之，这个是个从总体到部分的算法，由于条件总优化的条件比分布优化的条件要强，所以只要总优化有解，那么分布优化必然有解。

综上所描述，我们讨论了两种情况，并且解决了转化的问题。根据以上的分析，我们今后只讨论喷油嘴的运行模式为平均时间分布的时候。

5.5 问题 2：单向阀物理模型的更新

对比第一个问题，我们在这个一个部分需要更新的是 A 口单向阀的模型，B 口由于不变的情况所以我们仍然不需要考虑。

凸轮的模型更新如下，如图 5.5-1。

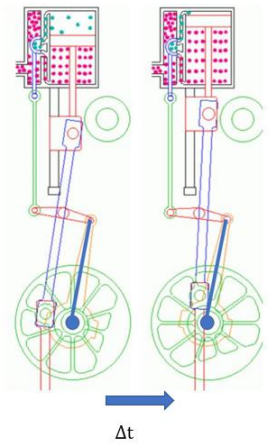


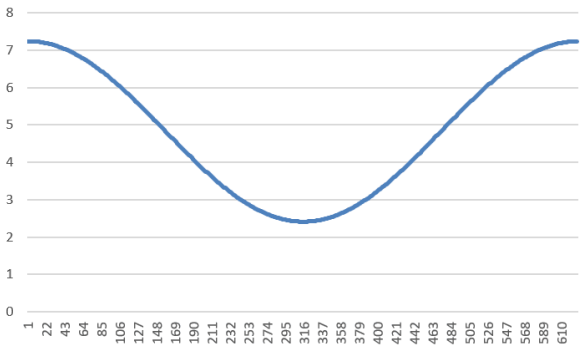
图 5.5-1 凸轮的转动模型

由于凸轮的转动，我们可以明显地看到整个油泵被压缩了，因此改变了油泵的体积，简单点来说，我们在这里关注的是$\triangle h$，也即凸轮的周期性转动导致的$\triangle h$的周期性变化。从上图我们可以得到几个递推的式子：

$$\triangle V_{pu} = \pi\left(\frac{d_0}{2}\right)^2\left(r_t\cos\theta_t - r_{t-1}\cos\theta_{t-1}\right)$$

$$\theta_t = \theta_{t-1} + \omega\triangle t$$

其中r_t 和 θ_t 是附件一中的极角和极径，我们通过 Python Scipy 库拟合得到一下结果：



方程为 $r=-2.413000878090353*\sin(0.9999999820919380\theta+1.5692047768000164)+4.826000802224875$

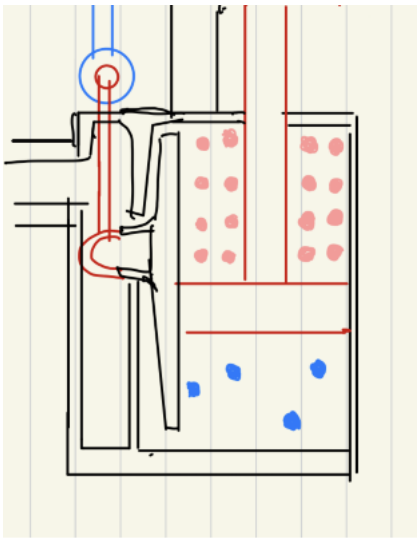


图 5.5-2 活塞装置

接下来考虑压缩油泵的的模型，如图 5.5-2 所示。我们可以用以下物理模型描述这个压缩泵：

$$P_{pu}^{t}-P_{pu}^{t-1}=\frac{E_{pu}^{t-1}}{\rho_{pu}^{t-1}}(\rho_{pu}^{t}-\rho_{pu}^{t-1})$$

$$V_{pu}^{t}=V_{pu}^{t-1}+\Delta V_{pu}$$

$$\Delta V_{pu}=\Delta h\left(\frac{d}{2}\right)^{2}\pi=g_{(\theta_t+\omega\Delta t)}-g_{(\theta_{t-1})}$$

$$r=g_{(\theta)}=a\sin\left(b\theta+c\right)+d$$

$$\rho_{pu}^{t}=\frac{m_{pu}^{t-1}-sign\left(P_{pu}^{t-1},P_{pip}^{t-1}\right)Q_{pu}^{t-1}\Delta t\rho_{pu}^{t-1}-sign(B)\cdot V^{\triangle t}{}_{B}\rho^{t-1}{}_{pip}}{V_{pu}^{t}}$$

$$Q_{pu}^{t-1}=CA_{pu}\sqrt{\frac{2P_{pu}^{t-1}-P_{pip}^{t-1}}{\rho_{pu}^{t-1}}}$$

$$A_{pu}=\left(\frac{d}{2}\right)^{2}\pi$$

而高压油管的模型于之前的一样，B 口我们不能控制暂时不考虑，将提过的所有方程离散化为以下递推公式：

$$P^{t}{}_{pip} - P^{t-1}{}_{pip} = \frac{E^{t-1}{}_{pip}}{\rho^{t-1}{}_{pip}}(\rho^{t}{}_{pip} - \rho^{t-1}{}_{pip})$$

$$\rho_{pip}{}^{t} = \frac{m_{pip}{}^{t-1} + sign(A)\cdot Q_A \triangle t \rho^{t-1}{}_{pu} - sign(B)\cdot V^{\triangle t}{}_B \rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q^{\triangle t}{}_A = CA_A\sqrt{\frac{2\Delta P_A}{\rho_{pu}{}^{t-1}}} = CA_A\sqrt{\frac{2(P^{t}{}_A - P^{t-1}{}_A)}{\rho_{pu}{}^{t-1}}}$$

$$V^{\triangle t}{}_B = vt = S_{cruve} = \frac{1}{2}\triangle t(S(t) + S(t-1))$$

$$P_{pu}^{t} - P_{pu}^{t-1} = \frac{E_{pu}^{t-1}}{\rho_{pu}^{t-1}}(\rho_{pu}^{t} - \rho_{pu}^{t-1})$$

$$V_{pu}^{t} = V_{pu}^{t-1} + \Delta V_{pu}$$

$$\triangle V_{pu} = \pi\left(\frac{d_0}{2}\right)^2\left(r_t\cos\theta_t - r_{t-1}\cos\theta_{t-1}\right)$$

$$r = g_{(\theta)} = a\sin\left(b\theta + c\right) + d$$

$$\rho_{pu}^{t} = \frac{m_{pu}^{t-1} - sign\left(P_{pu}^{t-1}, P_{pip}^{t-1}\right)Q_{pu}^{t-1}\Delta t\rho_{pu}^{t-1} + + sign(\Delta h)m_0}{V_{pu}^{t}}$$

$$Q_{pu}^{t-1} = CA_{pu}\sqrt{\frac{2P_{pu}^{t-1} - P_{pip}^{t-1}}{\rho_{pu}^{t-1}}}$$

$$A_{pu} = \left(\frac{d}{2}\right)^2\pi$$

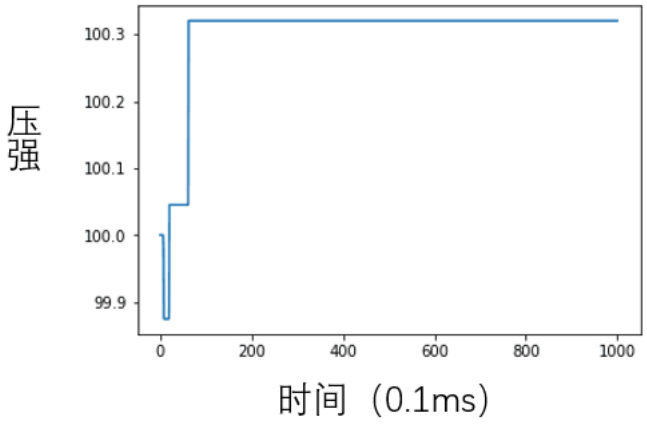
至此，整个问题二的系统可以得到仿真。

5.6 问题 2：如何通过角速度控制

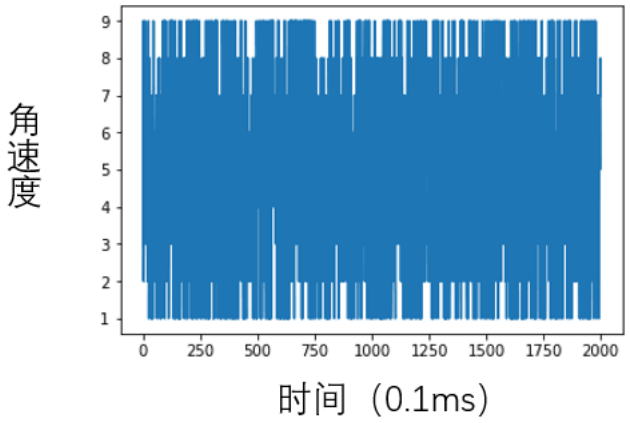
根据问题一的引导，我们仍然是先考虑喷嘴分布工作的情况。我们首先猜测我们仍是寻找一个油泵凸轮的角速度ω的一个周期，使得我们能够在一个阶段稳定住这个高压油管体系的压强。我保持优化目标的不变：

$$\min \ \sum(P_{pip}^{t} - 100)^2$$

在一个周期（100ms）里面，我们取求解一个稳定方式使得整个体系能够稳定，在这里我们要求取 1000ω的值，这里属于冗余编码。这 1000 个角速度描述了这个周期体系里面每一 0.1ms 的状态。然后我们仍然使用遗传算法来求解最优化的结果。求取的高压油管状态如下图所示：



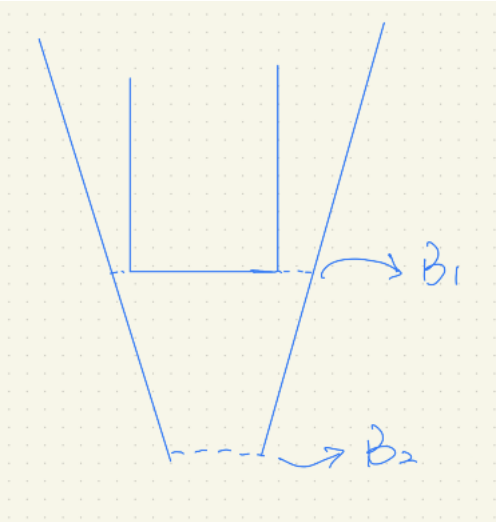
角速度周期如下图所示：



5.7 问题三：控制结构物理的模型更新

第四问中我们添加了两个新的控制原件，一个相同的喷油嘴和一个的单向出油阀。

对于喷油嘴，我们肯定是要考虑到更为具体的情况，我们对喷油嘴进行物理模型的更新，把喷油分成三层：高压油管、B1、B2，如下图所示。



可用以下物理方程进行描述：

$$P_{pip}^{t}-P_{pip}^{t-1}=\frac{E_{pip}^{t-1}}{\rho_{pip}^{t-1}}\left(\rho_{pip}^{t}-\rho_{pip}^{t-1}\right)$$

$$\rho_{pip}{}^{t}=\frac{m_{pip}{}^{t-1}+sign(A)\cdot Q_{A}\triangle t\rho^{t-1}{}_{pu}-sign(B)\cdot Q_{B}\triangle t\rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q^{\triangle t}{}_{A}=CA_{A}\sqrt{\frac{2\Delta P_{A}}{\rho_{pu}^{t-1}}}=CA_{A}\sqrt{\frac{2(P^{t}{}_{A}-P^{t-1}{}_{A})}{\rho_{pu}^{t-1}}}$$

$$Q^{\triangle t}{}_{B1}=CA_{B1}\sqrt{\frac{2\Delta P_{B1}}{\rho_{pip}^{t-1}}}=CA_{B1}\sqrt{\frac{2(P^{t}{}_{B1}-P^{t-1}{}_{B1})}{\rho_{pip}^{t-1}}}$$

$$Q^{\triangle t}{}_{B2}=CA_{B2}\sqrt{\frac{2\Delta P_{B2}}{\rho_{pip}^{t-1}}}=CA_{B2}\sqrt{\frac{2(P^{t}{}_{B2}-P^{t-1}{}_{B2})}{\rho_{pip}^{t-1}}}$$

$$A_{B1}=\pi\left[\left(h\tan 9^{\circ}+(2.5/2)\right)^{2}-(2.5/2)^{2}\right](mm^{2})$$

$$A_{B1}=\pi\left[1.4/2\right]^{2}(mm^{2})$$

我们默认喷油嘴的工作曲线仍然有效，根据附件的数据，我们可以获得外界的高压为 57.23453Mpa，至此，喷油嘴的模型已经完成。

对于 D 口，如上图所示，我们建立如下模型：

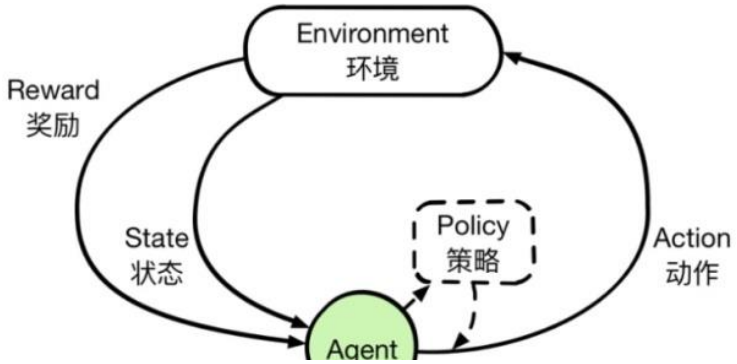
$$\rho_{pip}{}^{t}=\frac{m_{pip}{}^{t-1}\text{-}sign(D)\cdot Q_{D}\triangle t\rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q^{\triangle t}{}_{D}=CA_{D}\sqrt{\frac{2\Delta P_{D}}{\rho_{pip}^{t-1}}}=CA_{D}\sqrt{\frac{2(P^{t}{}_{D}-P^{t-1}{}_{D})}{\rho_{pip}^{t-1}}}$$

综上，我们已经对所有控制部件进行了建模。

5.8 问题三：强化学习与 DQN

强化学习主要包括五个部分——智能体(Agent)、环境(Environment)、状态(State)、动作（Action）、奖励（Reward）。它们之间的关系如图所示：



所谓强化学习，是指在智能体在随机执行了某个动作后，环境将会相应地转换至一个新的状态，对于这个新的状态，环境会反馈给智能体一个奖励信号（分为正奖励和负奖励）。然后智能体根据所得到的反馈与所处的新的状态，改变自己的行为，来执行新的动作。通过不断执行这个过程，智能体可以知道自己子啊什么样的状态下执行什么样的动作来世的自身获得最大的奖励。

我们可以利用小孩子走路来作为一个例子：当小孩子想要走路时，他就相当于一个 agent（智能体）。他试图通过行走这个动作来操纵地面，并且通过走路来改变自己的状态，当他完成了“走路”这个任务时，小孩子就可以得到相应的奖励，而若他没有完成这项任务，他就不能得到奖励。

在我们这道题目当中，加入了多个控制元件，需要选择的参数已经超过一千个，所以我们肯定不能选择传统的优化控制论进行优化，于是我们将目光转向强化学习，转向 DQN，一种神经网络与强化学习相结合的机器学习算法。

整个 DQN 算法框架如下：

第一步：用一个深度神经网络来作为 Q 值的网络，参数为 ω

$$Q(s,a,\omega) \approx Q^{\pi}(s,a)$$

第二部：在 Q 值中使用均方差 mean-square error 来定义目标函数 objective function 也就是 loss function

$$L(\omega) = E[\underbrace{r + \gamma \cdot \max_{a'} Q(s',a',\omega)}_{Target} - Q(s,a,\omega)^2]$$

第三步：计算参数 ω 关于 loss function 的梯度

$$\frac{\partial L(\omega)}{\partial \omega}=E[(\underbrace{r+\gamma\cdot\max_{a'}Q(s',a',\omega)}_{Target}-Q(s,a,\omega))\frac{\partial Q(s,a,\omega)}{\partial \omega}]$$

上面公式是s',a'即下一个状态和动作，这里用了 David Silver 的表示方式，看起来比较清晰。可以看到，这里就是使用了 Q-Learning 要更新的 Q 值作为目标值。有了目标值，又有当前值，那么偏差就能通过均方差来进行计算。

第四步：使用 SGD 实现 End-to-end 的优化目标 ，有了上面的梯度，而$\frac{\partial Q(s,a,\omega)}{\partial \omega}$可以从深度神经网络中进行计算，因此，就可以使用 SGD 随机梯度下降来更新参数，从而得到最优的 Q 值。

5.9 环境与 Reward 函数的设置

对于只添加一个喷油嘴的情况，环境用一下方程组描述：

$$P_{pip}^{t}-P_{pip}^{t-1}=\frac{E_{pip}^{t-1}}{\rho_{pip}^{t-1}}\left(\rho_{pip}^{t}-\rho_{pip}^{t-1}\right)$$

$$\rho_{pip}{}^{t}=\frac{m_{pip}{}^{t-1}+sign(A)\cdot Q_A\triangle t\rho^{t-1}{}_{pu}-sign(B)\cdot Q_B\triangle t\rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q^{\triangle t}{}_{A}=CA_A\sqrt{\frac{2\Delta P_A}{\rho_{pu}{}^{t-1}}}=CA_A\sqrt{\frac{2(P^{t}{}_{A}-P^{t-1}{}_{A})}{\rho_{pu}{}^{t-1}}}$$

$$Q^{\triangle t}{}_{B}=CA_{B1}\sqrt{\frac{2\Delta P_{B1}}{\rho_{pip}{}^{t-1}}}=CA_{B1}\sqrt{\frac{2(P^{t}{}_{B1}-P^{t-1}{}_{B1})}{\rho_{pip}{}^{t-1}}}=CA_{B2}\sqrt{\frac{2\Delta P_{B2}}{\rho_{pip}{}^{t-1}}}=CA_{B2}\sqrt{\frac{2(P^{t}{}_{B2}-P^{t-1}{}_{B2})}{\rho_{pip}{}^{t-1}}}$$

$$A_{B1}=\pi\left[\left(h\tan 9^{\circ}+2.5\right)^2-2.5^2\right](mm^2)$$

对于添加了所有控制设备的情况，环境用以下方程组描述：

$$P_{pip}^{t}-P_{pip}^{t-1}=\frac{E_{pip}^{t-1}}{\rho_{pip}^{t-1}}\left(\rho_{pip}^{t}-\rho_{pip}^{t-1}\right)$$

$$\rho_{pip}{}^{t}=\frac{m_{pip}{}^{t-1}+sign(A)\cdot Q_A\triangle t\rho^{t-1}{}_{pu}-sign(B)\cdot Q_A\triangle t\rho^{t-1}{}_{pip}-sign(C)\cdot Q_C\triangle t\rho^{t-1}{}_{pip}-sign(D)\cdot Q_D\triangle t\rho^{t-1}{}_{pip}}{V_{pip}}$$

$$Q^{\triangle t}{}_{A}=CA_A\sqrt{\frac{2\Delta P_A}{\rho_{pu}{}^{t-1}}}=CA_A\sqrt{\frac{2(P^{t}{}_{A}-P^{t-1}{}_{A})}{\rho_{pu}{}^{t-1}}}$$

$$Q^{\triangle t}{}_{B}=CA_{B1}\sqrt{\frac{2\Delta P_{B1}}{\rho_{pip}{}^{t-1}}}=CA_{B1}\sqrt{\frac{2(P^{t}{}_{B1}-P^{t-1}{}_{B1})}{\rho_{pip}{}^{t-1}}}=CA_{B2}\sqrt{\frac{2\Delta P_{B2}}{\rho_{pip}{}^{t-1}}}=CA_{B2}\sqrt{\frac{2(P^{t}{}_{B2}-P^{t-1}{}_{B2})}{\rho_{pip}{}^{t-1}}}$$

$$Q^{\triangle t}{}_{C}=CA_{C1}\sqrt{\frac{2\Delta P_{C1}}{\rho_{pip}{}^{t-1}}}=CA_{C1}\sqrt{\frac{2(P^{t}{}_{C1}-P^{t-1}{}_{C1})}{\rho_{pip}{}^{t-1}}}=CA_{C2}\sqrt{\frac{2\Delta P_{C2}}{\rho_{pip}{}^{t-1}}}=CA_{C2}\sqrt{\frac{2(P^{t}{}_{C2}-P^{t-1}{}_{C2})}{\rho_{pip}{}^{t-1}}}$$

$$Q^{\triangle t}{}_{D}=CA_{D}\sqrt{\frac{2\Delta P_{D}}{\rho_{pip}{}^{t-1}}}=CA_{D}\sqrt{\frac{2(P^{t}{}_{D}-P^{t-1}{}_{D})}{\rho_{pip}{}^{t-1}}}$$

$$A_{B}=\pi\left[\left(h\tan 9^{\circ}+2.5\right)^{2}-2.5^{2}\right](mm^{2})$$

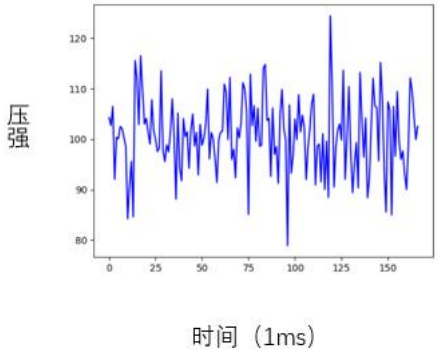
$$A_{C}=\pi\left[\left(h\tan 9^{\circ}+2.5\right)^{2}-2.5^{2}\right](mm^{2})$$

对于 Reward 函数，我们仍然使用：

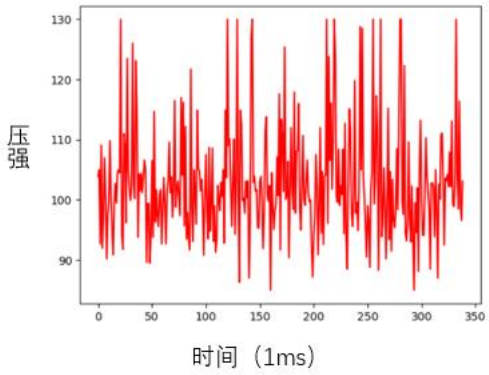
$$(P_{pip}^{t}-100)^{2}$$

5.10 DQN 训练结果

我们对于对于多添加一个喷油嘴的情况，训练 DQN 结果如下图所示：



我们对于添加了所有控制部件的情况，训练 DQN 结果如下：



六、模型的评价

6.1 模型的优点

（1）清楚扎实地尽力模型，并且论证自己的结论。

（2）模型具有一般性，不具有太多的特殊假设。

（3）引入强化学习，进行实时控制。

6.2 模型的缺点

（1）模型过于复杂，程序编程复杂。

参考文献

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.

[2] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013

附录

问题 1：
1）

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""
import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
    if a >= b:
        return 1
    elif a < b:
        return 0


def pip_state(x):

    p_pip_t = x[0]
    m_pip_t = x[1]
    row_pip_t = x[2]

    row_160_oil = 0.8711

    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t - row_pip_t_mins_one),
            row_pip_t -( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one - a[j]*(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(160,p_pip_t_mins_one)*(160-p_pip_t_mins_one))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) + A(c[j])))*(0.5)))]


def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e

def row_oil(p):
```

```
    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 + c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
        return 10*t
    elif 2< t <= 22:
        return 20
    elif 22< t <=24:
        return -10*t + 240

def pip(x):


    t = 20000

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip


    p_pip = [100]       #MPA
    m_pip = [0.850*500*5*5*math.pi]
    row_pip = [0.850]

    #print(x)

    for i in range(len(x)):
        x[i] = int(math.floor(x[i]))

    #open_time = int(x[0])
    #print(open_time)

    #start_time = math.floor(x[0])
    #last_time = math.floor(x[1])



    global a
    global b
    global c

    a = np.zeros(20001)
    b = np.zeros(20001)
```

```
c = np.zeros(20001)

for i in range(20):
    for k in range(1,25):
        b[1000*i + k] = 1
        c[1000*i + k] = k


counter = 0


for i in range(len(x)):
    try:

        for p in range(counter + int(x[2*i+1]),counter + int(x[2*i+1]+x[2*i+2])):
            a[p] = 1
        counter = counter + int(x[2*i+1]+x[2*i+2]) + 100

    except:

        break
        break



global j
#print(t)

for j in range(1,t+1):
    #print(j)
    #row_160_oil = 0.8711

    #V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)

    result = optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one])

    p_pip.append(result[0])
    m_pip.append(result[1])
    row_pip.append(result[2])

on_goal = 0

for i in p_pip:
    if 149 < i < 151:
        on_goal += 1
print(on_goal)
if on_goal ==0:
    return ((((np.array(p_pip) - 150)**2).sum())/20001)
else:
    return (1/on_goal)
```

```
        #print((((np.array(p_pip) - 150)**2).sum())/20001)

        #return (((np.array(p_pip) - 150)**2).sum()/20001)


ppp=[]
high=[]
low=[]

for i in range(200):
        low.append(1)
        low.append(1)
        high.append(20000)
        high.append(20000)



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *


class TestGA:
        def __init__(self):
                self.func = pip
                self.func_type = 'min'
                self.variables_num = len(high)
                self.lower_bound = low
                self.upper_bound = high
                self.cross_rate = 0.8
                self.mutation_rate = 0.2
                self.generations = 50
                self.population_size = 10
                self.binary_code_length = 20
                self.cross_rate_exp = 1.005
                self.mutation_rate_exp = 1.005
                self.code_type = code_type.binary
                self.cross_code = False
                self.select_method = select_method.keep_best
                self.rank_select_probs = None
                self.tournament_num = 2
                self.cross_method = cross_method.uniform
                self.arithmetic_cross_alpha = 0.1
                self.arithmetic_cross_exp = 1
                self.mutation_method = mutation_method.uniform
                self.none_uniform_mutation_rate = 1
                #self.complex_constraints = [constraints1,constraints2,constraints3]
                self.complex_constraints = None
                self.complex_constraints_method = complex_constraints_method.penalty
                self.complex_constraints_C = 1e6
                self.M = 1e8
                self.GA = GA(**self.__dict__)

        def test(self):
                start_time = time.time()
```

```
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()
            ppp.append(self.GA.__dict__)



if __name__ == '__main__':
    TestGA().test()

'''
from time import time
from sopt.util.functions import *
from sopt.util.pso_config import *
from sopt.PSO.PSO import PSO
from sopt.util.constraints import *

class TestPSO:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(low)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.c1 = basic_config.c1
        self.c2 = basic_config.c2
        self.generations = 50
        self.population_size = 100
        self.vmax = 1
        self.vmin = -1
        self.w = 1
        self.w_start = 0.9
        self.w_end = 0.4
        self.w_method = pso_w_method.linear_decrease
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints = None
        self.complex_constraints_method = complex_constraints_method.loop
        self.PSO = PSO(**self.__dict__)

    def test(self):
        start_time = time()
        self.PSO.run()
        print("PSO costs %.4f seconds!" %(time()-start_time))
        self.PSO.save_plot()
        self.PSO.show_result()


if __name__ == '__main__':
    TestPSO().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.Optimizers.SA import SA
from sopt.util.sa_config import *
from sopt.util.constraints import *
```

```
class TestSA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(low)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.T_start = 100
        self.T_end = 1e-6
        self.q = 0.9
        self.L = 100
        self.init_pos = None
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints_method = complex_constraints_method.loop
        self.SA = SA(**self.__dict__)


    def test(self):
        start_time = time()
        self.SA.run()
        print("SA costs %.4f seconds!" %(time()-start_time))
        self.SA.save_plot()
        self.SA.show_result()


if __name__ == '__main__':
    TestSA().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.util.constraints import *
from sopt.util.random_walk_config import *
from sopt.Optimizers.RandomWalk import RandomWalk

class TestRandomWalk:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(high)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.generations = 1000
        self.init_step = 100
        self.eps = 1e-4
        self.vectors_num = 100
        self.init_pos = None
        # self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints = None
        self.complex_constraints_method = complex_constraints_method.loop
        self.RandomWalk = RandomWalk(**self.__dict__)

    def test(self):
        start_time = time()
        self.RandomWalk.random_walk()
        print("random walk costs %.4f seconds!" %(time() - start_time))
```

```
            self.RandomWalk.save_plot()
            self.RandomWalk.show_result()


if __name__ == '__main__':
    TestRandomWalk().test()
'''
2）# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""
import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
    if a >= b:
        return 1
    elif a < b:
        return 0


def pip_state(x):

    p_pip_t = x[0]
    m_pip_t = x[1]
    row_pip_t = x[2]

    row_160_oil = 0.8711

    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t - row_pip_t_mins_one),
            row_pip_t -( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one - a[j]*(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(160,p_pip_t_mins_one)*(160-p_pip_t_mins_one))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) + A(c[j])))*(0.5)))]


def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744
```

```
    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e

def row_oil(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 + c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
          return 10*t
    elif 2< t <= 22:
          return 20
    elif 22< t <=24:
          return -10*t + 240

def pip(x):


    t = 50000

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip


    p_pip = [100]        #MPA
    m_pip = [0.850*500*5*5*math.pi]
    row_pip = [0.850]

    #print(x)

    for i in range(len(x)):
          x[i] = int(math.floor(x[i]))

    #open_time = int(x[0])
    #print(open_time)

    #start_time = math.floor(x[0])
    #last_time = math.floor(x[1])



    global a
```

```
global b
global c

a = np.zeros(50001)
b = np.zeros(50001)

c = np.zeros(50001)

for i in range(50):
    for k in range(1,25):
        b[1000*i + k] = 1
        c[1000*i + k] = k


counter = 0


for i in range(len(x)):
    try:

        for p in range(counter + int(x[2*i+1]),counter + int(x[2*i+1]+x[2*i+2])):
            a[p] = 1
        counter = counter + int(x[2*i+1]+x[2*i+2]) + 100

    except:

        break
        break



global j
#print(t)

for j in range(1,t+1):
    #print(j)
    #row_160_oil = 0.8711

    #V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)

    result = optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one])

    p_pip.append(result[0])
    m_pip.append(result[1])
    row_pip.append(result[2])

on_goal = 0

for i in p_pip:
    if 145 < i < 155:
        on_goal += 1
print(on_goal)
```

```
        if on_goal != 0:

            print((1/on_goal)*((((np.array(p_pip) - 150)**2).sum())/50001))

        if on_goal ==0:
            return ((((np.array(p_pip) - 150)**2).sum())/50001)
        else:
            return (1/on_goal)*((((np.array(p_pip) - 150)**2).sum())/50001)



        #print((((np.array(p_pip) - 150)**2).sum())/50001)

        #return 1/on_goal

ppp = []

high=[]
low=[]

for i in range(500):
        low.append(1)
        low.append(1)
        high.append(50000)
        high.append(50000)



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *


class TestGA:
        def __init__(self):
                self.func = pip
                self.func_type = 'min'
                self.variables_num = len(high)
                self.lower_bound = low
                self.upper_bound = high
                self.cross_rate = 0.8
                self.mutation_rate = 0.05
                self.generations = 20
                self.population_size = 10
                self.binary_code_length = 20
                self.cross_rate_exp = 1.005
                self.mutation_rate_exp = 1.005
                self.code_type = code_type.binary
                self.cross_code = False
                self.select_method = select_method.keep_best
                self.rank_select_probs = None
                self.tournament_num = 2
                self.cross_method = cross_method.uniform
                self.arithmetic_cross_alpha = 0.1
                self.arithmetic_cross_exp = 1
```

```
            self.mutation_method = mutation_method.uniform
            self.none_uniform_mutation_rate = 1
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.penalty
            self.complex_constraints_C = 1e6
            self.M = 1e8
            self.GA = GA(**self.__dict__)

        def test(self):
            start_time = time.time()
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()
            ppp.append(self.GA.__dict__)



if __name__ == '__main__':
        TestGA().test()

'''
from time import time
from sopt.util.functions import *
from sopt.util.pso_config import *
from sopt.PSO.PSO import PSO
from sopt.util.constraints import *

class TestPSO:
        def __init__(self):
            self.func = pip
            self.func_type = 'min'
            self.variables_num = len(low)
            self.lower_bound = np.array(low)
            self.upper_bound = np.array(high)
            self.c1 = basic_config.c1
            self.c2 = basic_config.c2
            self.generations = 50
            self.population_size = 100
            self.vmax = 1
            self.vmin = -1
            self.w = 1
            self.w_start = 0.9
            self.w_end = 0.4
            self.w_method = pso_w_method.linear_decrease
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.loop
            self.PSO = PSO(**self.__dict__)

        def test(self):
            start_time = time()
            self.PSO.run()
            print("PSO costs %.4f seconds!" %(time()-start_time))
            self.PSO.save_plot()
            self.PSO.show_result()
```

```
if __name__ == '__main__':
    TestPSO().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.Optimizers.SA import SA
from sopt.util.sa_config import *
from sopt.util.constraints import *

class TestSA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(low)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.T_start = 100
        self.T_end = 1e-6
        self.q = 0.9
        self.L = 100
        self.init_pos = None
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints_method = complex_constraints_method.loop
        self.SA = SA(**self.__dict__)


    def test(self):
        start_time = time()
        self.SA.run()
        print("SA costs %.4f seconds!" %(time()-start_time))
        self.SA.save_plot()
        self.SA.show_result()


if __name__ == '__main__':
    TestSA().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.util.constraints import *
from sopt.util.random_walk_config import *
from sopt.Optimizers.RandomWalk import RandomWalk

class TestRandomWalk:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(high)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.generations = 1000
        self.init_step = 100
        self.eps = 1e-4
```

```
            self.vectors_num = 100
            self.init_pos = None
            # self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.loop
            self.RandomWalk = RandomWalk(**self.__dict__)

        def test(self):
            start_time = time()
            self.RandomWalk.random_walk()
            print("random walk costs %.4f seconds!" %(time() - start_time))
            self.RandomWalk.save_plot()
            self.RandomWalk.show_result()


if __name__ == '__main__':
    TestRandomWalk().test()
"'3）
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""

import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
    if a >= b:
        return 1
    elif a < b:
        return 0


def pip_state(x):

    p_pip_t = x[0]
    m_pip_t = x[1]
    row_pip_t = x[2]

    row_160_oil = 0.8711

    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t -
row_pip_t_mins_one),
            row_pip_t -( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one -
```

```
a[j]*(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(160,p_pip_t_mins_one)*(160-p_pip_t_mins_one
))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) + A(c[j])))*(0.5)))]


def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e

def row_oil(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 +
c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
          return 10*t
    elif 2< t <= 22:
          return 20
    elif 22< t <=24:
          return -10*t + 240

def pip(x):


    t = 100000

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip


    p_pip = [100]       #MPA
    m_pip = [0.850*500*5*5*math.pi]
    row_pip = [0.850]

    #print(x)
```

```
for i in range(len(x)):
    x[i] = int(math.floor(x[i]))

#open_time = int(x[0])
#print(open_time)

#start_time = math.floor(x[0])
#last_time = math.floor(x[1])



global a
global b
global c

a = np.zeros(100001)
b = np.zeros(100001)

c = np.zeros(100001)

for i in range(100):
    for k in range(1,25):
        b[1000*i + k] = 1
        c[1000*i + k] = k


counter = 0


for i in range(len(x)):
    try:

        for p in range(counter + int(x[2*i+1]),counter + int(x[2*i+1]+x[2*i+2])):
            a[p] = 1
        counter = counter + int(x[2*i+1]+x[2*i+2]) + 100

    except:

        break
        break



global j
#print(t)

for j in range(1,t+1):
    #print(j)
    #row_160_oil = 0.8711

    #V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)
```

```
            result = optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one])

            p_pip.append(result[0])
            m_pip.append(result[1])
            row_pip.append(result[2])

        on_goal = 0

        for i in p_pip:
            if 145 < i < 155:
                on_goal += 1

        print(on_goal)

        if on_goal != 0:

            print((1/on_goal)*((((np.array(p_pip) - 150)**2).sum())/100001))

        if on_goal ==0:
            return ((((np.array(p_pip) - 150)**2).sum())/100001)
        else:
            return (1/on_goal)*((((np.array(p_pip) - 150)**2).sum())/100001)



        #print((((np.array(p_pip) - 150)**2).sum())/100001)

        #return (((np.array(p_pip) - 150)**2).sum()/100001)

ppp = []

high=[]
low=[]

for i in range(1000):
    low.append(1)
    low.append(1)
    high.append(100000)
    high.append(100000)



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *


class TestGA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(high)
        self.lower_bound = low
        self.upper_bound = high
        self.cross_rate = 0.8
        self.mutation_rate = 0.05
```

```
            self.generations = 10
            self.population_size = 30
            self.binary_code_length = 20
            self.cross_rate_exp = 1.005
            self.mutation_rate_exp = 1.005
            self.code_type = code_type.binary
            self.cross_code = False
            self.select_method = select_method.keep_best
            self.rank_select_probs = None
            self.tournament_num = 2
            self.cross_method = cross_method.uniform
            self.arithmetic_cross_alpha = 0.1
            self.arithmetic_cross_exp = 1
            self.mutation_method = mutation_method.uniform
            self.none_uniform_mutation_rate = 1
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.penalty
            self.complex_constraints_C = 1e6
            self.M = 1e8
            self.GA = GA(**self.__dict__)

        def test(self):
            start_time = time.time()
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()
            ppp.append(self.GA.__dict__)



if __name__ == '__main__':
        TestGA().test()

'''
from time import time
from sopt.util.functions import *
from sopt.util.pso_config import *
from sopt.PSO.PSO import PSO
from sopt.util.constraints import *

class TestPSO:
        def __init__(self):
            self.func = pip
            self.func_type = 'min'
            self.variables_num = len(low)
            self.lower_bound = np.array(low)
            self.upper_bound = np.array(high)
            self.c1 = basic_config.c1
            self.c2 = basic_config.c2
            self.generations = 50
            self.population_size = 100
            self.vmax = 1
            self.vmin = -1
            self.w = 1
            self.w_start = 0.9
```

```
            self.w_end = 0.4
            self.w_method = pso_w_method.linear_decrease
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.loop
            self.PSO = PSO(**self.__dict__)

        def test(self):
            start_time = time()
            self.PSO.run()
            print("PSO costs %.4f seconds!" %(time()-start_time))
            self.PSO.save_plot()
            self.PSO.show_result()


if __name__ == '__main__':
    TestPSO().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.Optimizers.SA import SA
from sopt.util.sa_config import *
from sopt.util.constraints import *

class TestSA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(low)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.T_start = 100
        self.T_end = 1e-6
        self.q = 0.9
        self.L = 100
        self.init_pos = None
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints_method = complex_constraints_method.loop
        self.SA = SA(**self.__dict__)


    def test(self):
        start_time = time()
        self.SA.run()
        print("SA costs %.4f seconds!" %(time()-start_time))
        self.SA.save_plot()
        self.SA.show_result()


if __name__ == '__main__':
    TestSA().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.util.constraints import *
```

```
from sopt.util.random_walk_config import *
from sopt.Optimizers.RandomWalk import RandomWalk

class TestRandomWalk:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(high)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.generations = 1000
        self.init_step = 100
        self.eps = 1e-4
        self.vectors_num = 100
        self.init_pos = None
        # self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints = None
        self.complex_constraints_method = complex_constraints_method.loop
        self.RandomWalk = RandomWalk(**self.__dict__)

    def test(self):
        start_time = time()
        self.RandomWalk.random_walk()
        print("random walk costs %.4f seconds!" %(time() - start_time))
        self.RandomWalk.save_plot()
        self.RandomWalk.show_result()


if __name__ == '__main__':
    TestRandomWalk().test()
'''
4）
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""
import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
    if a >= b:
        return 1
    elif a < b:
        return 0




def pip_state(x):

    p_pip_t = x[0]
    m_pip_t = x[1]
    row_pip_t = x[2]
```

```
    row_160_oil = 0.8711

    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t - row_pip_t_mins_one),
            row_pip_t -( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one - a[j]*(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(160,p_pip_t_mins_one)*(160-p_pip_t_mins_one))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) + A(c[j])))*(0.5)))]



'''
def pump_state(x):

    p_pump_t = x[0]
    m_pump_t = x[1]
    row_pump_t = x[2]

    return [p_t - p_t_mins_one -(E_mins/row_mins)*(row-row_mins),
            row-( m_t / V_pub),
            m_t - m_mins-a[i]*(deltat*0.85*0.7*0.7*math.pi*((2*(160-p_mins))/(row_160_oil))**(0.5))-b[i]*(row_mins*((deltat*(A(i-1)+A(i)))*(0.5)))]

'''







#result = optimize.fsolve(f,[1,1,1])
#print(result) # x0,x1,x2 的值
#print(f(result)) #  方程组的误差

def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e
```

```
def row_oil(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 + c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
          return 10*t
    elif 2< t <= 22:
          return 20
    elif 22< t <=24:
          return -10*t + 240

def pip(x):


    t = 100 + 24

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip


    p_pip = [100]       #MPA
    m_pip = [0.850*500*5*5*math.pi]
    row_pip = [0.850]

    print(x)

    start_time = math.floor(x[0])
    last_time = math.floor(x[1])



    global a
    global b
    global c

    a = np.zeros(125)
    b = np.zeros(125)

    c = np.zeros(125)

    for i in range(1,25):
```

```
            b[i] = 1
            c[i] = i


        if not(start_time + last_time <= 124):
            return float('inf')
        else:
            for i in range(start_time,start_time+last_time):
                a[i] = 1

            global j
            #print(t)

            for j in range(1,t+1):
                #print(j)
                #row_160_oil = 0.8711

                #V_pip = 500*5*5*math.pi

                p_pip_t_mins_one = p_pip[j-1]
                m_pip_t_mins_one = m_pip[j-1]
                row_pip_t_mins_one = row_pip[j-1]
                E_pip_t_mins_one = E(p_pip_t_mins_one)

                result                                                                                    =
optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one])

                p_pip.append(result[0])
                m_pip.append(result[1])
                row_pip.append(result[2])

            print(((np.array(p_pip) - 100)**2).sum())

            return ((np.array(p_pip) - 100)**2).sum()



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *

class TestGA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = 2
        self.lower_bound = [1,1]
        self.upper_bound = [125,125]
        self.cross_rate = 0.8
        self.mutation_rate = 0.05
        self.generations = 50
        self.population_size = 10
        self.binary_code_length = 20
        self.cross_rate_exp = 1
        self.mutation_rate_exp = 1
        self.code_type = code_type.binary
```

```
            self.cross_code = False
            self.select_method = select_method.proportion
            self.rank_select_probs = None
            self.tournament_num = 2
            self.cross_method = cross_method.uniform
            self.arithmetic_cross_alpha = 0.1
            self.arithmetic_cross_exp = 1
            self.mutation_method = mutation_method.uniform
            self.none_uniform_mutation_rate = 1
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.penalty
            self.complex_constraints_C = 1e6
            self.M = 1e8
            self.GA = GA(**self.__dict__)

        def test(self):
            start_time = time.time()
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()



if __name__ == '__main__':
        TestGA().test()
```

5）

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""
import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
        if a >= b:
                return 1
        elif a < b:
                return 0




def pip_state(x):

        p_pip_t = x[0]
        m_pip_t = x[1]
        row_pip_t = x[2]

        row_160_oil = 0.8711
```

```
    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t - row_pip_t_mins_one),
            row_pip_t -( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one - a[j]*(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(160,p_pip_t_mins_one)*(160-p_pip_t_mins_one))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) + A(c[j])))*(0.5)))]



'''
def pump_state(x):

    p_pump_t = x[0]
    m_pump_t = x[1]
    row_pump_t = x[2]

    return [p_t - p_t_mins_one -(E_mins/row_mins)*(row-row_mins),
            row-( m_t / V_pub),
            m_t - m_mins-a[i]*(deltat*0.85*0.7*0.7*math.pi*((2*(160-p_mins))/(row_160_oil))**(0.5))-b[i]*(row_mins*((deltat*(A(i-1)+A(i)))*(0.5)))]

'''







#result = optimize.fsolve(f,[1,1,1])
#print(result) # x0,x1,x2 的值
#print(f(result)) #  方程组的误差

def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e

def row_oil(p):

    a = 3.454177032715454e-07
```

```
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 + c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
          return 10*t
    elif 2< t <= 22:
          return 20
    elif 22< t <=24:
          return -10*t + 240

def pip(x):


    t = 100 + 24

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip


    p_pip = [100]       #MPA
    m_pip = [0.850*500*5*5*math.pi]
    row_pip = [0.85]

    print(x)

    start_time = math.floor(x[0])
    last_time = math.floor(x[1])



    global a
    global b
    global c

    a = np.zeros(125)
    b = np.zeros(125)

    c = np.zeros(125)

    for i in range(1,25):
          b[i] = 1
          c[i] = i
```

```
    if not(start_time + last_time <= 124):
        return float('inf')
    else:
        for i in range(start_time,start_time+last_time):
            a[i] = 1

        global j
        #print(t)

        for j in range(1,t+1):
            #print(j)
            #row_160_oil = 0.8711

            #V_pip = 500*5*5*math.pi

            p_pip_t_mins_one = p_pip[j-1]
            m_pip_t_mins_one = m_pip[j-1]
            row_pip_t_mins_one = row_pip[j-1]
            E_pip_t_mins_one = E(p_pip_t_mins_one)

            result                                                                                    =
optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one])

            p_pip.append(result[0])
            m_pip.append(result[1])
            row_pip.append(result[2])

        print((((np.array(p_pip) - 100)**2).sum())/125)

        return (((np.array(p_pip) - 100)**2).sum())/125



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *

class TestGA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = 2
        self.lower_bound = [1,1]
        self.upper_bound = [125,125]
        self.cross_rate = 0.8
        self.mutation_rate = 0.05
        self.generations = 50
        self.population_size = 10
        self.binary_code_length = 20
        self.cross_rate_exp = 1
        self.mutation_rate_exp = 1
        self.code_type = code_type.binary
        self.cross_code = False
        self.select_method = select_method.proportion
        self.rank_select_probs = None
```

```
            self.tournament_num = 2
            self.cross_method = cross_method.uniform
            self.arithmetic_cross_alpha = 0.1
            self.arithmetic_cross_exp = 1
            self.mutation_method = mutation_method.uniform
            self.none_uniform_mutation_rate = 1
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.penalty
            self.complex_constraints_C = 1e6
            self.M = 1e8
            self.GA = GA(**self.__dict__)

        def test(self):
            start_time = time.time()
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()



if __name__ == '__main__':
        TestGA().test()
```

问题 2：

```
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 13 01:57:30 2019

@author: Chonpca
"""
import math
from scipy import optimize
import numpy as np
import time

def sign(a,b):
        if a >= b:
                return 1
        elif a < b:
                return 0


def pip_state(x):

        p_pip_t = x[0]
        m_pip_t = x[1]
        row_pip_t = x[2]

        p_pu_t = x[3]
        m_pu_t = x[4]
        row_pu_t = x[5]

        V_pu_t = x[6]
```

```
    row_160_oil = 0.8711

    V_pip = 500*5*5*math.pi

    p_pip_t_mins_one = p_pip[j-1]
    m_pip_t_mins_one = m_pip[j-1]
    row_pip_t_mins_one = row_pip[j-1]
    E_pip_t_mins_one = E(p_pip_t_mins_one)


    #m_pu_t =
    #row_pu_t =

    omega = a[j]
    theta_t = 0
    #theta2
    for i in range(j+1):
        theta_t = theta_t + deltat*a[j]

    theta2 = 0

    for i in range(j):
        theta2 = theta2 + deltat*a[j]


    p_pu_t_mins_one = p_pu[j-1]
    m_pu_t_mins_one = m_pu[j-1]
    row_pu_t_mins_one = row_pu[j-1]
    V_pu_t_mins_one = V_pu[j-1]
    E_pu_t_mins_one = E(p_pu_t_mins_one)


    return [p_pip_t - p_pip_t_mins_one -( E_pip_t_mins_one / row_pip_t_mins_one )*( row_pip_t -
row_pip_t_mins_one),
            row_pip_t - ( m_pip_t / V_pip),
            m_pip_t - m_pip_t_mins_one -
(deltat*row_160_oil*0.85*0.7*0.7*math.pi*((2*sign(V_pu_t_mins_one,p_pip_t_mins_one)*(V_pu_t_min
s_one-p_pip_t_mins_one))/(row_160_oil))**(0.5))+ b[j]*(row_pip_t_mins_one*((deltat*(A(c[j-1]) +
A(c[j])))*(0.5))),
            p_pu_t - p_pu_t_mins_one - (E_pu_t_mins_one / row_pu_t_mins_one )*( row_pu_t -
row_pu_t_mins_one),
            row_pu_t - (m_pu_t / V_pu_t),
            V_pu_t - V_pu_t_mins_one - (((theta_t*g(theta_t + omega*deltat) -
theta2*g(theta2))*2.5*2.5*math.pi)) ,
            m_pu_t - m_pu_t_mins_one -
sign(p_pu_t_mins_one,p_pip_t_mins_one)*deltat*row_pu_t_mins_one*0.85*2.5*2.5*math.pi*((2*(p_pu_t
-p_pu_t_mins_one)/(row_pu_t_mins_one))**(0.5))]


def g(theta):

    j = -2.413000878090353
    k = 0.9999999982091938
```

```
    l = 1.5692047768000164
    m = 4.826000802224875

    return j*np.sin(k*theta+l)+m



def E(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return a*p**4 + b*p**3 + c*p**2 + d*p**1 + e

def row_oil(p):

    a = 3.454177032715454e-07
    b = -3.8129329087309995e-05
    c = 0.016667115687673203
    d = 4.687866549677347
    e = 1539.645780763744


    return math.e**((x/(a*p**4 + b*p**3 + c*p**2 + d*p**1 + e)) - ((100/(a*100**4 + b*100**3 + c*100**2 + d*100**1)) - math.log(0.85)))



def A(t):
    if    0<= t <= 2:
          return 10*t
    elif 2< t <= 22:
          return 20
    elif 22< t <=24:
          return -10*t + 240

def pip(x):


    t = 1000

    global deltat

    deltat = 0.1    # ms

    global p_pip
    global m_pip
    global row_pip
    global p_pu
    global m_pu
    global V_pu
    global row_pu
```

```
p_pu = [0.5]
p_pip = [100]        #MPA
m_pip = [0.850*500*5*5*math.pi]
row_pip = [0.850]
#m_pu = [0.5]
m_pu = [(20+7.5*2.5*2.5*math.pi)*0.8045]
row_pu = [0.8045]
V_pu =[(20+7.5*2.5*2.5*math.pi)]

#print(x)

for i in range(len(x)):
        x[i] = int(math.floor(x[i]))

#open_time = int(x[0])
#print(open_time)

#start_time = math.floor(x[0])
#last_time = math.floor(x[1])



global a
global b
global c

a = np.zeros(2001)
b = np.zeros(2001)

c = np.zeros(2001)

for i in range(1,25):
        b[i] = 1
        c[i] = i
for i in range(1001,1025):
        b[i] = 1
        c[i] = i-1000


#counter = 0


a = x


global j
#print(t)

for j in range(1,t+1):
        #print(j)
        #row_160_oil = 0.8711

        #V_pip = 500*5*5*math.pi

        p_pip_t_mins_one = p_pip[j-1]
        m_pip_t_mins_one = m_pip[j-1]
        row_pip_t_mins_one = row_pip[j-1]
```

```
            E_pip_t_mins_one = E(p_pip_t_mins_one)

            p_pu_t_mins_one = p_pu[j-1]
            m_pu_t_mins_one = m_pu[j-1]
            row_pu_t_mins_one = row_pu[j-1]
            V_pu_t_mins_one = V_pu[j-1]
            E_pu_t_mins_one = E(p_pu_t_mins_one)


            result = optimize.fsolve(pip_state,[p_pip_t_mins_one,m_pip_t_mins_one,row_pip_t_mins_one,p_pu_t_mins_one,
m_pu_t_mins_one,row_pu_t_mins_one,V_pu_t_mins_one])

            p_pip.append(result[0])
            m_pip.append(result[1])
            row_pip.append(result[2])
            p_pu.append(result[3])
            m_pu.append(result[4])
            row_pu.append(result[5])
            V_pu.append(result[6])




        print((((np.array(p_pip) - 100)**2).sum())/2001)

        return (((np.array(p_pip) - 100)**2).sum()/2001)


ppp=[]
high=[]
low=[]

for i in range(1000):
        low.append(1)
        low.append(1)
        high.append(10)
        high.append(10)



from sopt.GA.GA import GA
from sopt.util.functions import *
from sopt.util.ga_config import *
from sopt.util.constraints import *


class TestGA:
        def __init__(self):
            self.func = pip
            self.func_type = 'min'
            self.variables_num = len(high)
            self.lower_bound = low
            self.upper_bound = high
            self.cross_rate = 0.8
            self.mutation_rate = 0.2
            self.generations = 1
```

```
            self.population_size = 1
            self.binary_code_length = 20
            self.cross_rate_exp = 1.005
            self.mutation_rate_exp = 1.005
            self.code_type = code_type.binary
            self.cross_code = False
            self.select_method = select_method.keep_best
            self.rank_select_probs = None
            self.tournament_num = 2
            self.cross_method = cross_method.uniform
            self.arithmetic_cross_alpha = 0.1
            self.arithmetic_cross_exp = 1
            self.mutation_method = mutation_method.uniform
            self.none_uniform_mutation_rate = 1
            #self.complex_constraints = [constraints1,constraints2,constraints3]
            self.complex_constraints = None
            self.complex_constraints_method = complex_constraints_method.penalty
            self.complex_constraints_C = 1e6
            self.M = 1e8
            self.GA = GA(**self.__dict__)

        def test(self):
            start_time = time.time()
            self.GA.run()
            print("GA costs %.4f seconds!" % (time.time()-start_time))
            self.GA.save_plot()
            self.GA.show_result()
            ppp.append(self.GA.__dict__)



if __name__ == '__main__':
        TestGA().test()

'''
from time import time
from sopt.util.functions import *
from sopt.util.pso_config import *
from sopt.PSO.PSO import PSO
from sopt.util.constraints import *

class TestPSO:
        def __init__(self):
            self.func = pip
            self.func_type = 'min'
            self.variables_num = len(low)
            self.lower_bound = np.array(low)
            self.upper_bound = np.array(high)
            self.c1 = basic_config.c1
            self.c2 = basic_config.c2
            self.generations = 50
            self.population_size = 100
            self.vmax = 1
            self.vmin = -1
            self.w = 1
            self.w_start = 0.9
            self.w_end = 0.4
```

```
        self.w_method = pso_w_method.linear_decrease
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints = None
        self.complex_constraints_method = complex_constraints_method.loop
        self.PSO = PSO(**self.__dict__)

    def test(self):
        start_time = time()
        self.PSO.run()
        print("PSO costs %.4f seconds!" %(time()-start_time))
        self.PSO.save_plot()
        self.PSO.show_result()


if __name__ == '__main__':
    TestPSO().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.Optimizers.SA import SA
from sopt.util.sa_config import *
from sopt.util.constraints import *

class TestSA:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(low)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.T_start = 100
        self.T_end = 1e-6
        self.q = 0.9
        self.L = 100
        self.init_pos = None
        #self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints_method = complex_constraints_method.loop
        self.SA = SA(**self.__dict__)


    def test(self):
        start_time = time()
        self.SA.run()
        print("SA costs %.4f seconds!" %(time()-start_time))
        self.SA.save_plot()
        self.SA.show_result()


if __name__ == '__main__':
    TestSA().test()
'''
'''
from time import time
from sopt.util.functions import *
from sopt.util.constraints import *
from sopt.util.random_walk_config import *
```

```
from sopt.Optimizers.RandomWalk import RandomWalk

class TestRandomWalk:
    def __init__(self):
        self.func = pip
        self.func_type = 'min'
        self.variables_num = len(high)
        self.lower_bound = np.array(low)
        self.upper_bound = np.array(high)
        self.generations = 1000
        self.init_step = 100
        self.eps = 1e-4
        self.vectors_num = 100
        self.init_pos = None
        # self.complex_constraints = [constraints1,constraints2,constraints3]
        self.complex_constraints = None
        self.complex_constraints_method = complex_constraints_method.loop
        self.RandomWalk = RandomWalk(**self.__dict__)

    def test(self):
        start_time = time()
        self.RandomWalk.random_walk()
        print("random walk costs %.4f seconds!" %(time() - start_time))
        self.RandomWalk.save_plot()
        self.RandomWalk.show_result()


if __name__ == '__main__':
    TestRandomWalk().test()
'''
```