

计算物理 A 第十二题

杨旭鹏 PB17000234

2019 年秋季

目录

1 题目描述	1
2 算法	2
3 程序使用方法	2
4 程序结果与讨论	2
4.1 方程参数对于迭代结果影响的可视化结果	2
4.2 Feigenbaum 常数的计算	8
4.3 关于分岔值计算时的方法讨论	9
5 心得与体会	11
6 附录	12
A C 语言源程序	12
B 可视化绘图及数据分析 Python 程序源码	14

1 题目描述

以 $x_{n+1} = \lambda \sin(\pi x_n)$ 为迭代方程:

- (1) 画出系统状态随参数 λ 的变化图, 要求包括定值状态、倍周期分叉和混沌状态;
- (2) 列出各个倍周期分叉处的 λ 值, 求相应的 Feigenbaum 常数。

2 算法

对一定区间内间隔为某一值 $step$ 的每一个 λ , 任意选定 x 的初始值 x_0 , 利用给定的迭代公式计算并输出 N 迭代后的后 n 个 x 的值 $x_{N-n}, x_{N-n+1}, \dots, x_N$ 。然后利用输出的数据画出 $x - \lambda$ 图像即可。

对于找计算 Feigenbaum 常数所需的分岔值, 对不同 λ 取值输出的后 n 个迭代值进行种类计数, 找到不同个数的不动点所对应的参数 λ 取值。然后不断缩小区间, 加大精度来找到比较精确的分岔值 λ_m 。

3 程序使用方法

此程序设计为参数在程序代码中直接赋值形式, 每次需在程序源码中进行参数调整, 进而编译运行, 以简化每次输入的过程 (重复运行时不用在此输入)。可供调整的参数包括参数 λ 的最大最小值, 其每一步的步长, 迭代次数 N , 输出迭代结果个数 n , 调整完参数后, 编译运行, 程序会自动输出不同参数 λ 对应的后 n 个迭代结果至文件 (在同一文件中, 顺序按 λ 从小到大排列)。

4 程序结果与讨论

4.1 方程参数对于迭代结果影响的可视化结果

最开始对方程的特性不了解, 故先设定 $step = 0.5$, $\lambda \in [-100, 100]$ 得到:

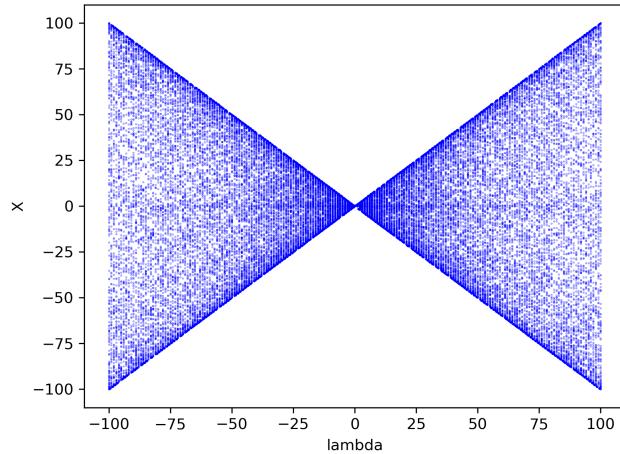


图 1: $step = 0.5$, $\lambda \in [-100, 100]$

发现此迭代方程只有在 0 附近收敛，在趋向负无穷和正无穷的过程中逐渐趋于混沌。

我们减小 λ 的范围至 $[-10, 10]$, 并调小 $step$ 至 0.01 得到:

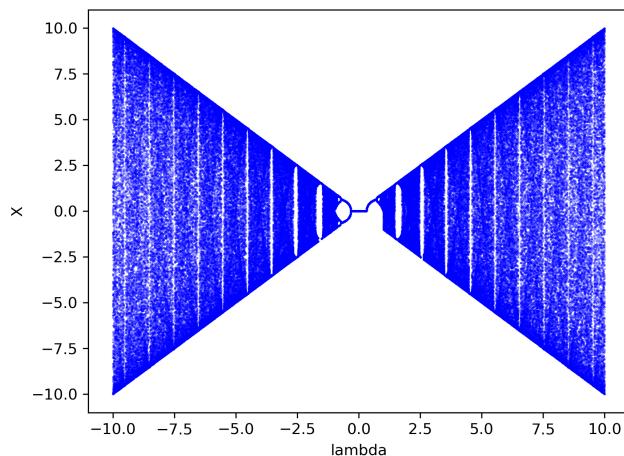


图 2: $step = 0.01$, $\lambda \in [-10, 10]$

我们减小 λ 的范围至 $[-1, 1]$, 并调小 $step$ 至 0.001 得到:

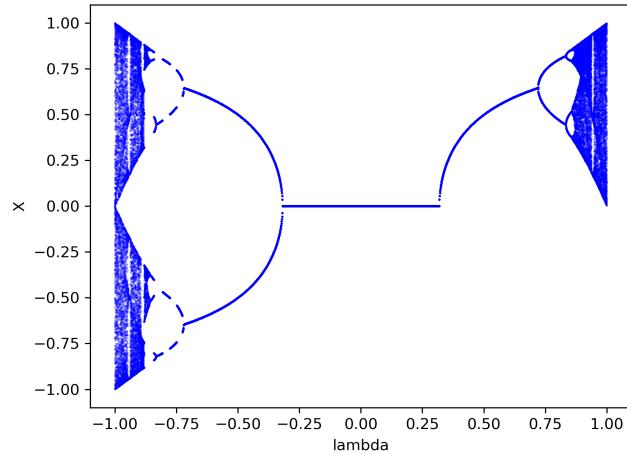


图 3: $step = 0.001, \lambda \in [-1, 1]$

可以看到在 0 附近, 迭代方程是收敛到 0 的, 即所谓的“绝灭”。随着参数 λ 从 0 开始向无穷方向移动, 方程从“绝灭”到一个不动点(定态)到 2 个不动点到 4 个不动点以此类推。我们将参数 λ 的范围一分为 2 来看有:

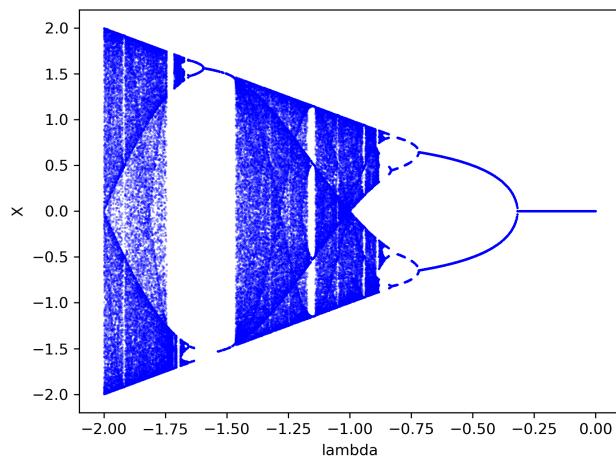


图 4: $step = 0.001, \lambda \in [-2, 0]$

可以看到在靠近 0 处，系统处于绝灭态，随着 λ 逐渐变小系统依次变成 2 分岔，4 分岔，直到变为混沌态，之后又有周期窗口等现象。

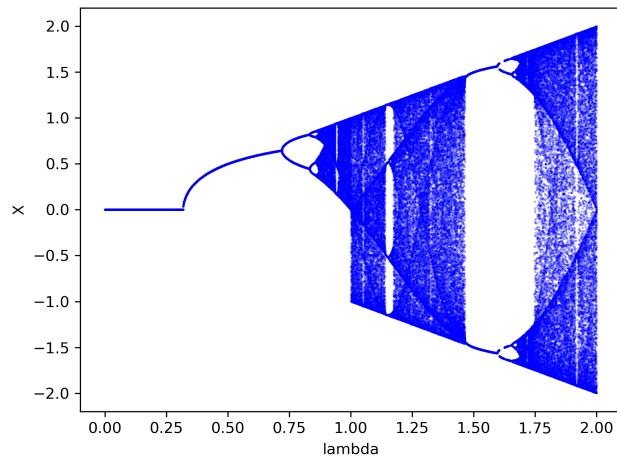


图 5: $step = 0.001, \lambda \in [0, 2]$

可以看到在靠近 0 处，系统处于绝灭态，随着 λ 逐渐变大系统依次变成定值态，2 分岔，4 分岔，直到变为混沌态，之后又有周期窗口等现象。

若将图形的一部分放大，我们期待能有分形的结构。下面展示对 $\lambda < 0$ 的情况下部分图像的放大：

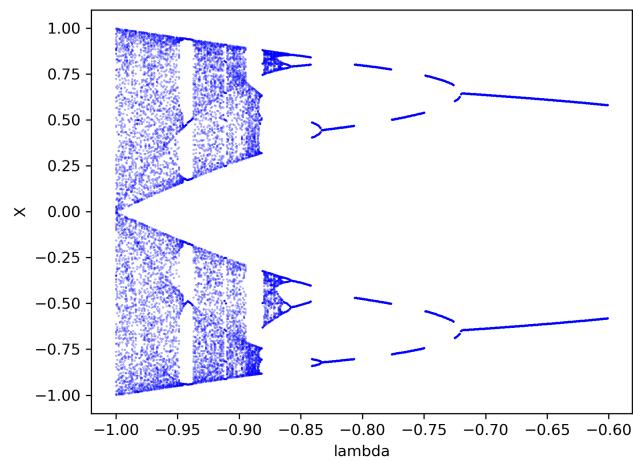


图 6: $step = 0.001, \lambda \in [-1, -0.6]$

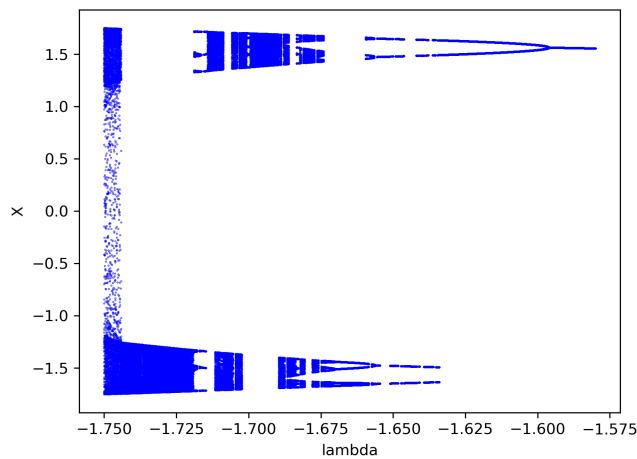


图 7: $step = 0.0001, \lambda \in [-1.75, -1.58]$

可以大致看出此图形的确有分形的结构。

我们还将 λ 在 $[-2, 0], [0, 2]$ 两个区间内以间隔为 0.001 做出迭代输出的后 100 个结果中的不同结果个数与参数 λ 的关系曲线：

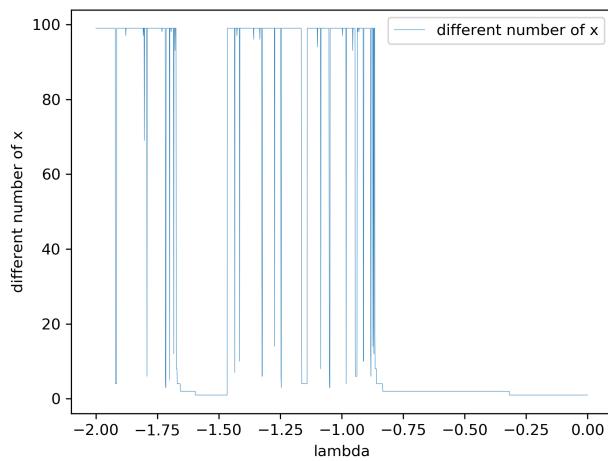


图 8：输出迭代后 100 个结果， $step = 0.001$ ， $\lambda \in [-2, 0]$ 的迭代不同结果个数- λ 曲线

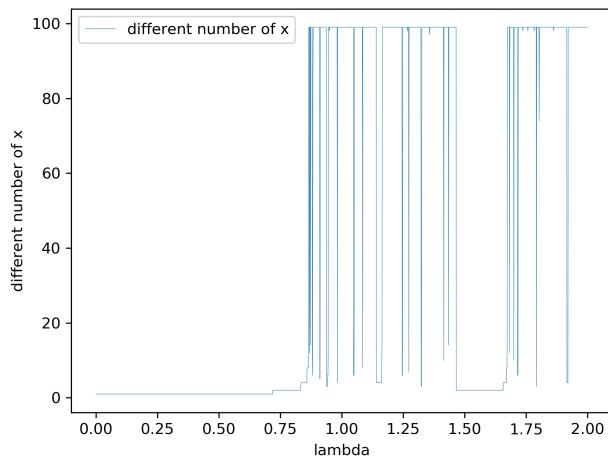


图 9：输出迭代后 100 个结果， $step = 0.001$ ， $\lambda \in [0, 2]$ 的迭代不同结果个数- λ 曲线

4.2 Feigenbaum 常数的计算

为了计算 Feigenbaum 常数，我们利用 Python 脚本¹对不同 λ 取值输出的后 100 个迭代值进行分析。对于常数 δ ，对结果的不同值进行个数统计，找到不同个数的不动点所对应的参数 λ 取值。然后不断缩小区间，加大精度来找到比较精确的分岔值 λ_m 。这里省略具体的过程，只给出最后的结果：

m	分叉情况	分岔值 λ_m	Feigenbaum 常数 δ
1	$1 \rightarrow 2$	-0.3183122	
2	$2 \rightarrow 4$	-0.8332658	20.31936
3	$4 \rightarrow 8$	-0.858608	4.62852
4	$8 \rightarrow 16$	-0.8640842	4.66095
5	$16 \rightarrow 32$	-0.86525894	4.66611
6	$32 \rightarrow 64$	-0.8655107	

表 1: $\lambda < 0$ 时的 Feigenbaum 常数 δ 计算结果

m	分叉情况	分岔值 λ_m	Feigenbaum 常数 δ
1	$1 \rightarrow 2$	0.7199599	
2	$2 \rightarrow 4$	0.8332658	4.47090
3	$4 \rightarrow 8$	0.8586088	4.62852
4	$8 \rightarrow 16$	0.8640842	4.66071
5	$16 \rightarrow 32$	0.865259	4.66746
6	$32 \rightarrow 64$	0.8655107	

表 2: $\lambda > 0$ 时的 Feigenbaum 常数 δ 计算结果

¹具体代码见附录

与 Feigenbaum 常数理论值 $\delta = 4.669201609 \dots$ 比较，看出除 $\lambda < 0$ 情况下的 $\frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_2}$ 以外，计算模拟值与理论值误差比较小。具体为何 $\lambda < 0$ 情况下的 $\frac{\lambda_2 - \lambda_1}{\lambda_3 - \lambda_2}$ 与理论值相差较大，目前本人并不清楚。另外注意到，参数 λ 大于 0 和小于 0 的时候，分岔值除了 λ_1 其他的基本一致，而小于 0 的情形下，与 λ_1 有关的 Feigenbaum 常数 δ 计算值与理论值偏差很大。

由于 $\lambda < 0$ 时的情况有些复杂²，但是与 > 0 方法差不多，故下面只讨论 $\lambda > 0$ 的情况对于 Feigenbaum 常数 α 的值，我们也如法炮制，不断缩小区间，加大精度，看输出结果与 0.5 的差值小于某个值时，认为此时与 0.5 相交，认为此值为 λ_m

m	与 0.5 相交值 λ_m	d_m	Feigenbaum 常数 α
0	0.500000		
1	0.7777338	0.27773377	
2	0.84638217	0.10720426	2.590697142072526
3	0.86145035	0.04251802	2.521384109608114
4	0.864694181	0.01696186	2.506683818873638

表 3: $\lambda > 0$ 时的 Feigenbaum 常数 α 计算结果

与理论值 $\alpha = 2.502907875 \dots$ 比较可看出随着 λ_m 的精度提高，其计算值与理论值偏差越小，基本验证此常数值。

4.3 关于分岔值计算时的方法讨论

一般情况下，我们理所当然的认为将输出结果输入到分析脚本中，判断结果和之前的结果均不同时，结果种类数 +1。但实际操作中，发现这样处理数据会出现一些问题。例如我们改变输出结果的精度时，统计结果会出现偏差：

²本人发现对于一些分岔值并不会与 0.5 相交，几乎相交线只与混沌区中的某些值相交

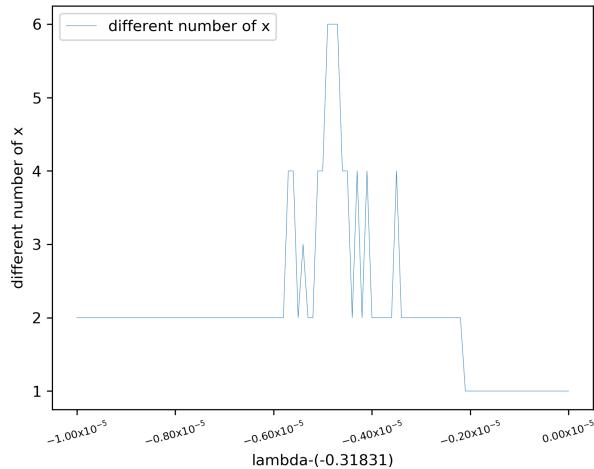


图 10: 输出结果保留 6 位小数时, $step = 10^{-7}$, $\lambda \in [-0.31832, -0.31831]$ 的迭代不同结果个数- λ 曲线

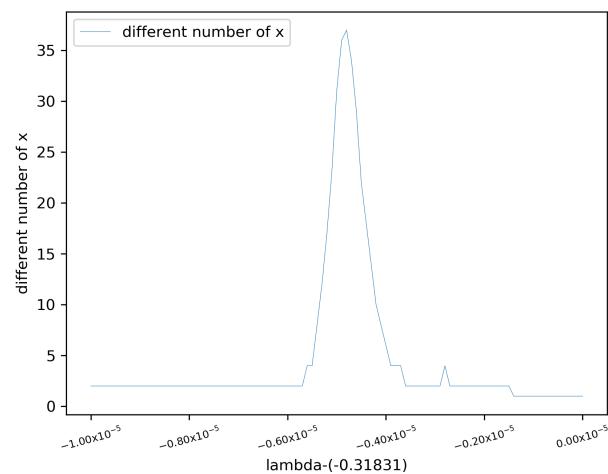


图 11: 输出结果保留 7 位小数时, $step = 10^{-7}$, $\lambda \in [-0.31832, -0.31831]$ 的迭代不同结果个数- λ 曲线

但实际上，对此区间的迭代结果进行可视化绘图得到

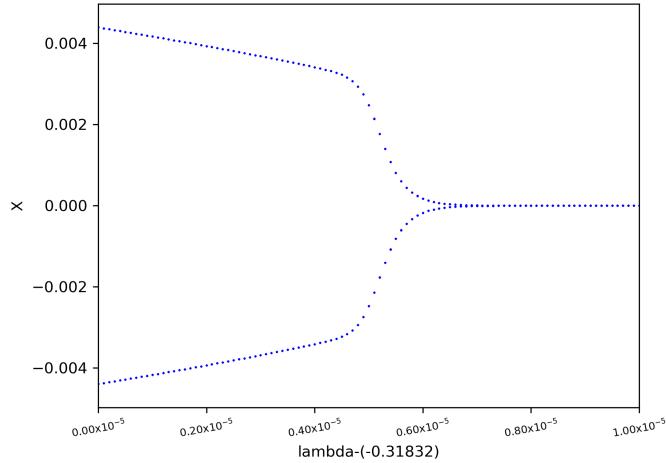


图 12: 输出结果保留 7 位小数时, $step = 10^{-7}$, $\lambda \in [-0.31832, -0.31831]$ 的迭代结果可视化

明显看出此区间内为绝灭到 2 分岔值的转变，并无更高分岔值的转变。利用脚本分析分岔值出现这种问题的原因可能为程序迭代时不可避免的出现了一些很小的舍入误差，故造成本该相同的值输出值不同。所以对于分岔很多的情况下，必须考虑减小计算机迭代过程中出现的各种误差，从而得到正确的分岔值。

5 心得与体会

通过此次作业，对混沌和 Feigenbaum 常数有了更深的认识。另外注意到在程序中比如类似（以 C 语言的 for 循环示意）`for(y = min;y < max+step;y += step)` 的循环，当 step 很小的时候，由于计算机的误差，可能会导致此循环进行 $y = max+step$ 的步骤。所以循环应改为 `for(y = min;y < max+step/2;y += step)`，这样才能保证程序进行我们希望计算的循环。寻找分岔值时，也体会到数值计算误差带来的一些事情需要特别小心注意。

6 附录

A C 语言源程序

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5 #define Pi 3.1415926535
6
7 int my_filewriter_double(char str1[],char str2[],double
8     num[],double y,int n){
9     FILE * fp;
10    char str[20];
11    strcpy(str,str1);
12    strcat(str,str2);
13    fp = fopen(str,"a+");
14
15    for(int i=0;i<(n);i++){
16        fprintf(fp,"%lf,%lf\n",y,num[i]);
17
18    }
19    fclose(fp);
20    return 0;
21 }
22
23
24
25 int my_fiter(double x[],int N,int n,double y,double x0){
26     double temp = x0; //迭代初值
27     for(int i =0;i<N-n;i++){
28         temp = y*sin(Pi*temp);
29     }
30
31     for(int i =0;i<n;i++){ //要输出的结果进行幅值
32         temp = y*sin(Pi*temp);
33         x[i] = temp;
```

```
34     }
35
36     return 0;
37 }
38
39
40
41 int main(int argc, const char * argv[]) {
42     double min = 0.86551; //lambda的最小值
43     double max = 0.86552; //lambda的最大值
44     int N = 1000000; //迭代次数
45     int n = 100; //输出迭代结果的后多少个结果
46     double step = 0.0000001; //不同lambda之间的间隔
47
48     double *x = malloc(sizeof(double)*n);
49
50     for(double y = min;y < max + (step/(double)2);y += step){
51         //y为迭代参数lambda
52         my_fiter(x, N, n, y, 15); //迭代初值选为15
53         my_filewriter_double("", "x.dat", x, y, n);
54     }
55
56     return 0;
57 }
```

B 可视化绘图及数据分析 Python 程序源码

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import math
4 import csv
5 #from IPython.core.pylabtools import figsize # import figsize
6 plt.rcParams['savefig.dpi'] = 300 #图片像素
7 plt.rcParams['figure.dpi'] = 300 #分辨率
8 # 默认的像素: [6.0,4.0], 分辨率为100, 图片尺寸为 600&400
9 fig1 = plt.figure()
10 fig2 = plt.figure()
11
12 ax1 = fig1.add_subplot(111)
13 ax2 = fig2.add_subplot(111)
14
15 step = 0.0000001 # 每个lambda取值之间的间隔
16 max = 0.86552 # lambda的取值区间
17 min = 0.86551
18
19 X = []
20 Y = []
21 X1 = []
22 Y1 = []
23 flag = 0
24 i = 0
25 temp1 = []
26 temp = 0
27
28 csv_file = csv.reader(open('problem 12/x.dat', 'r'))
29 for line in csv_file:
30     [tempX, tempY] = line
31     i += 1
32     if i % 100 == 0 and i != 0: #
            每读100行数据, 进行统计 (因为C程序输出后100个迭代值)
33         temp2 = set(temp1) #
            因为此步骤考虑不同lambda值对应的输出x值中的不同结果数,
34         # 故变为集合形式, 使其中只有不同的元素, 然后其长度即代表所求
```

```

35     temp = len(temp2)
36     Y1.append(len(temp2)) #
37         记录不同lambda值对应的输出x值中的不同结果数
38     temp1 = []
39 else:
40     temp1.append(float(tempY))
41
42 X.append(float(tempX)) # lambda 的值
43 Y.append(float(tempY)) # 与相应lambda对应的输出结果
44
45 X = np.array(X)
46 Y = np.array(Y)
47
48
49 ax1.scatter(X, Y, c='b', label='', s=0.1, alpha=0.5, marker='o') #
50         x-lambda散点图
51 ax1.set_xlabel('lambda')
52 ax1.set_ylabel('X')
53 fig1.savefig("1.png")
54
55 X1 = np.arange(min, max + step/2, step)
56
57 ax2.plot(X1, Y1, lw=0.3, label='different number of x') #
58         x不同结果数-lambda折线图
59 ax2.legend(loc=2)
60 ax2.set_xlabel('lambda')
61 ax2.set_ylabel('different number of x')
62 fig2.savefig("2.png")
63
64
65 if max < 0:
66     Y1.reverse() # 当参数lambda<0时，需对Y1 List
67         进行排序上的反转，>0时此步骤不运行
68 for i in Y1: # 输出

```

```
lambda值及其对应的输出结果中的x的种类数 (用来找分岔值)
69   if flag < 1001:
70     print(min + flag*step, "\t", i) #
    前面为lambda值，后面为其对应的输出结果中的x的结果数
71   flag += 1
```