INDRODUCTION TO ARTIFICIAL INTELLIGENCE Homework #1

1.

使用Readlines方式讀出文件中所要帶入方程式的x,y值，其各自的range用逗號分隔找出。再用兩個for迴圈將x,y帶入，得到的值存入list中再從中找出最小值。

2.

|  |  |  |
| --- | --- | --- |
|  | call func次數 | 初始點 |
| 暴力解 | 12221 | X |
| Hill climbing(step size-1) | 266  230  256  194  202  122  90  198  86  118 | (33,63)  (-58,-9)  (-53,41)  (-60,18)  (-24,56)  (-31,29)  (19,-11)  (25,54)  (38,13)  (-49,10) |

3.a

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| call func次數 | step size-1 | step size-2 | step size-4 | step size-8 | step size-16 |
| (33,63) | 266 | 138 | 66 | 34 | 18 |
| (-58,-9) | 230 | 118 | 62 | 30 | 14 |
| (-53,41) | 256 | 130 | 66 | 34 | 18 |
| (-60,18) | 194 | 98 | 50 | 26 | 14 |
| (-24,56) | 202 | 102 | 54 | 26 | 14 |
| (-31,29) | 122 | 62 | 34 | 14 | 10 |
| (19,-11) | 90 | 46 | 20 | 14 | 6 |
| (25,54) | 198 | 102 | 50 | 26 | 20 |
| (38,13) | 86 | 42 | 26 | 10 | 6 |
| (-49,10) | 118 | 62 | 30 | 16 | 10 |

3.b

隨著step size步伐上升，call func的次數會明顯減少，但得到的最終結果卻不一定是最小值；此外，步伐上升容易因跨太大步而跨過了最小值，導致在谷點邊緣徘徊而停不下來；若是剛好四周值都相近，就可能卡在平原的地方。

我們並不知道方程式有幾個谷點，因此可能只是找到區域最佳解而非全域，和初始點有很重要的關係。

我覺得step size侷限了尋找最小值的範圍，我覺得可以先針對大範圍的搜索在慢慢收斂某個特定區域，而不是一開始就被初始點綁死；至於落到谷底但非最小值的改良方式，我自己本身沒有什麼想法，但有搜尋到前人的改良方法「模擬退火演算法」是以機率的方式使有機會跳脫並非最佳解的區域，且完全改善初始值的影響。

4.

Input.txt抓值--起初使用Readlines方式和for迴圈，共同找出input.txt內的值，發現太耗時間，且有bug（在抓第三行時會同時把第四行也抓出來），故改用只用Readlines方式抓取。

Hill climbing – 原本使用座標點與上下左右（共5次call func）去做挑出最小值的動作，但發現其實座標點就是上一次判斷的最佳解。因此改為上下左右（共4次call func）挑出最小值，再和上一次判斷的最佳解比大小，減少call func次數。

暴力破解 VS Hill climbing -- 暴力破解雖然耗能耗時（我的電腦跑完約5分鐘）但可找出全域最佳解，而Hill climbing雖然可大大降低計算時間（我的電腦step size=1，一個點，約2-3秒），但可能如3.b第二段所說，不一定是全域最佳解。

由目前的資料觀測下來，此方程式的最低點可能在（0,30）附近，最小值可能約為-30.010，且此方程式有不只一個谷點（X次方在2次以上）。