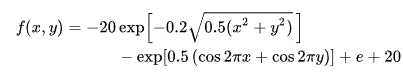
## 演化計算導論 期末書面報告

1. 實作目標：
   1. 利用Simple GA，其中使用雙點交配及使用兩個點取範圍變異，找出Ackley Function函數在二維的全域最小值。
   2. 評估Simple GA在解決問題時的表現，包括收斂速度及解的品質。
2. 問題描述：

Ackley Function是一個數學最佳化的函數，用於優化算法的性能測試問題。公式為：

x及y為函數的變數，設定其他變數為建議值：

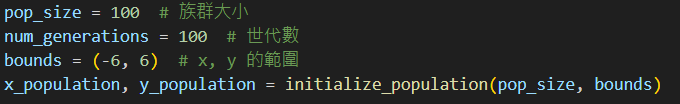
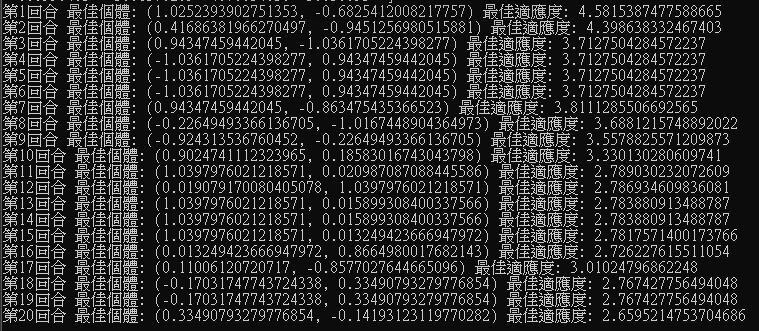
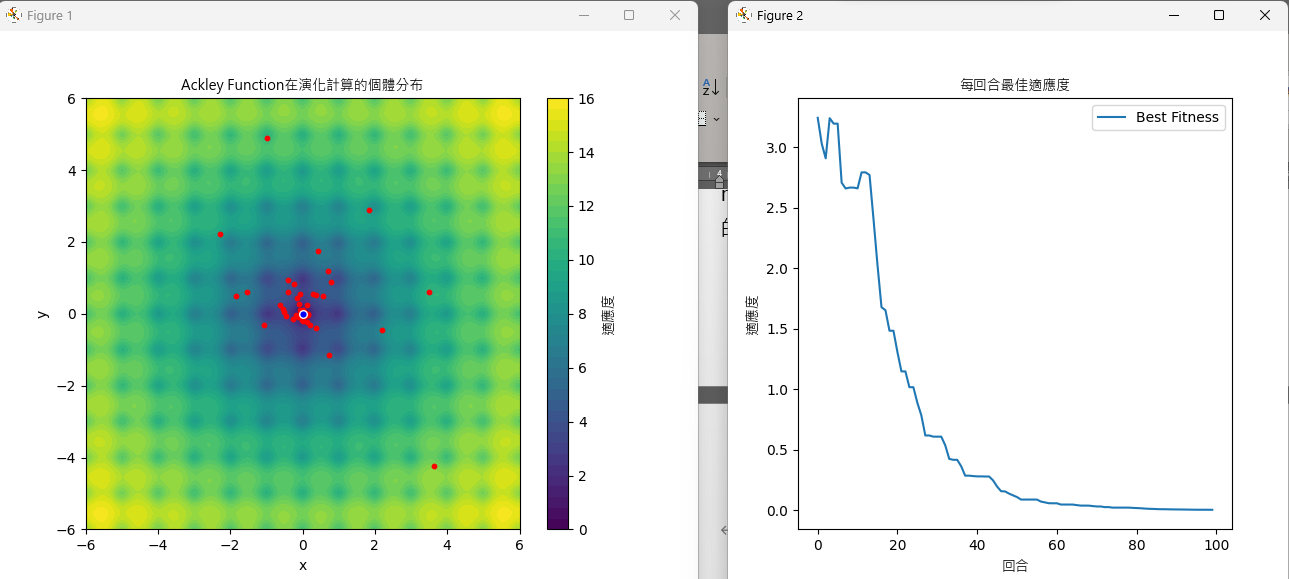
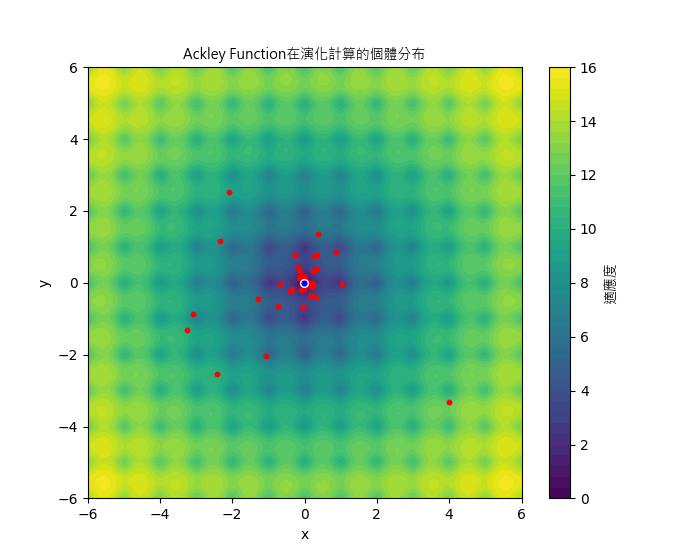
a = 20, b= 0.2, c=2π

1. 演化計算設計：使用Simple GA
   1. 選擇(Selection)：應當評估適應度以做選擇，但考慮到可能有的值可能無法進入交配及變異環節，故使所有的個體都被選擇。
   2. 交配(crossover)：使用雙點交配(two-point crossover method)，挑選隨機二點，範圍內的x及y個體做交換。
   3. 變異(Mutation)：在x, y列表中，各隨機挑選兩點，範圍內的x除以1.1，範圍內的y除以1.2。
   4. 替換完後計算出子代得出的最小數，並且保留該數。
   5. 繼續下一輪的演化計算
2. 實作流程
   1. 設定初始個體：生成x, y的值。
   2. 評估父代：按照演化計算設計的內容評估適應度。
   3. 做演化計算，並迭代，不斷更新最優解，直到停止計算。
   4. 輸出最優解。
3. 使用語言：

使用Python做迭代更新以、計算適應度以及圖形化更新過程。

1. 回合數或停止條件：

設定固定的回合數，做完即停止。

1. 實際操作：
   1. 設定初始值：提供可調參數，可以隨時更改情況。
   2. 演化計算流程：設計出交配以及變異的過程。
2. 定義 Ackley 函數
3. **def** **ackley\_function**(x, y):
4. a = 20
5. b = 0.2
6. c = 2\*np.pi
7. sum\_sq\_term = -a \* np.exp(-b \* np.sqrt(0.5 \* (x\*\*2 + y\*\*2)))
8. cos\_term = -np.exp(0.5 \* (np.cos(c \* x) + np.cos(c \* y)))
9. **return** sum\_sq\_term + cos\_term + a + np.exp(1)
10. # 初始化族群
11. **def** **initialize\_population**(pop\_size, bounds):
12. x\_population = np.random.uniform(bounds[0], bounds[1], pop\_size)
13. y\_population = np.random.uniform(bounds[0], bounds[1], pop\_size)
14. **print**(x\_population)
15. **print**(y\_population)
16. **return** x\_population, y\_population
17. # 計算適應度
18. **def** **calculate\_fitness**(x\_population, y\_population):
19. fitness = np.array([ackley\_function(x, y) **for** x, y **in** zip(x\_population, y\_population)])
20. #print(f'適應度為：{fitness}')
21. **return** fitness
22. # 交配(使用雙點交配)
23. **def** **crossover**(x\_population, y\_population):
24. offspring\_x = np.copy(x\_population)
25. offspring\_y = np.copy(y\_population)
26. # 隨機選擇交配範圍
27. idx1, idx2 = np.sort(np.random.randint(0, x\_population.size, 2))
28. #print(idx1,idx2)
29. # 在範圍內進行交配
30. tmp\_x = np.copy(offspring\_x)
31. offspring\_x[idx1:idx2] = offspring\_y[idx1:idx2]
32. offspring\_y[idx1:idx2] =tmp\_x[idx1:idx2]
33. **return** offspring\_x, offspring\_y
34. # 變異
35. **def** **inverse\_mutation**(offspring\_x, offspring\_y):
36. mutation\_x = np.copy(offspring\_x)
37. mutation\_y = np.copy(offspring\_y)
38. # 隨機選擇變異範圍
39. idx1, idx2 = np.sort(np.random.randint(0, offspring\_y.size, 2))
40. idx3, idx4 = np.sort(np.random.randint(0, offspring\_y.size, 2))
41. #print(idx1,idx2)
42. # 在範圍內進行變異
43. mutation\_x[idx1:idx2] = [x/1.1 **for** x **in** mutation\_x[idx1:idx2]]
44. mutation\_y[idx3:idx4] = [y/1.2 **for** y **in** mutation\_y[idx3:idx4]]
45. #為防止值被交換回原位
46. **return** mutation\_x,mutation\_y
    1. 迭代：每回合更新最佳個體及最佳適應度。
    2. 圖形化：為了觀察每一代的變化及最佳適應度，使用matplotlib製作每一代個體可視化圖形以及每代最佳適應度的折線圖。
47. 成果：
    1. 程式碼：<https://hackmd.io/@i1sXxH_zTBS11LYfqJBZUA/Hyr_i0Dva>
    2. 實際結果：由於在變異的時候我讓個體的值都稍微變小，所以在散點圖可以看出最佳個體的分布大概都在(x, y) = (0, 0)附近的位置（藍色點）。最佳適應度（Ackley Function最小值）也會愈來愈趨近於0，以達到觀察求最小值的目的。將(x, y)的範圍縮小，則可以觀察到趨近於0的速度愈快；將回合數提高則可以看到更精準的收斂過程。
48. 參考資料：

<https://www.sfu.ca/~ssurjano/ackley.html>

<https://ithelp.ithome.com.tw/articles/10218678>

<https://docs.sciml.ai/Surrogates/stable/ackley/>