```
# 用來求下列函數的最小值
# f(x) = 3 x^4 - 8 x^3 - 6 x^2 + 24 x
# 此函數之 global opt soln = (-1, -19), local opt soln = (2, 8)

import numpy as np
import math
MAXIT = 400      # maximal iteration number
K   = 1.0        # Boltzmann rate
DWELL = 20       # 計算平衡狀態時需要的迴圈數目
T_high = 1000.0 # 初始溫度
T_scale = 0.9    # 演算法每階段降溫比率: t0 -->
T_low = 1.0      # 最終冷卻溫度溫度
```

設定參數

```
# 設定目標函式
def SAfunc(x):
    return (((3*x - 8)*x - 6)*x + 24)*x
```

設定目標函數(成本函數)

主程式

```
#  ==== 主程式 ====
np.random.seed(0)   # 若要每次跑得都不一樣的結果，
# 找初始解 x
xbest = x = np.random.uniform(-3.0, 3.0)
ybest = y = SAfunc(x)

num_it = 0
t = T_high

while num_it < MAXIT and t > T_low:

    for i in range(DWELL) :
        # 找鄰居 xnew
        xnew = x + np.random.uniform(-0.1, 0.1)
        ynew = SAfunc(xnew)
        if ynew < y or np.random.uniform(0.0, 1.0) < math.exp( - (ynew - y) / (K * t) ):
            x = xnew
            y = ynew
        if ynew < ybest:     # 若新的成本比較小，取代最佳解
            xbest = xnew
            ybest = ynew

    print('\tf(%f) = %f\n' %(xbest, ybest))

    t *= T_scale
    num_it += 1
```

隨機給產生一組解 x ∈ [-3, 3]
計算其成本值y，並設定xbest, ybest

迴圈數 num_it 小於最大迴圈 MAXIT
且 溫度 t 比最低溫 T_low 高

熱平衡(重複執行DWELL次)

產生鄰居解 xnew 為 x 的左右 0.1 範圍內隨機數
並計算其成本值 ynew

若鄰居解較佳 (ynew < y)
或 r < exp(-(ynew – y) / (K * t))
則接受 xnew 取代 x

若鄰居解比目前最佳解好，則取代