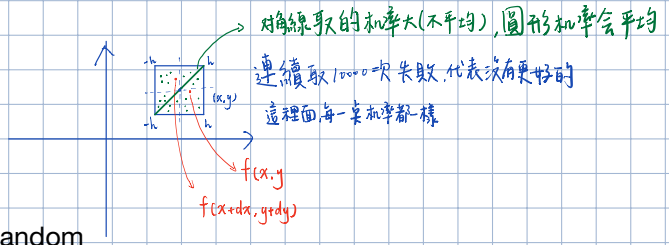
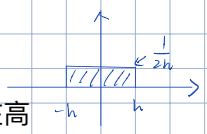
爬山演算法：隨机的做法，隨机在他前後h.左右h之間的方形範圍，去取一個点看有沒有比現在的更好



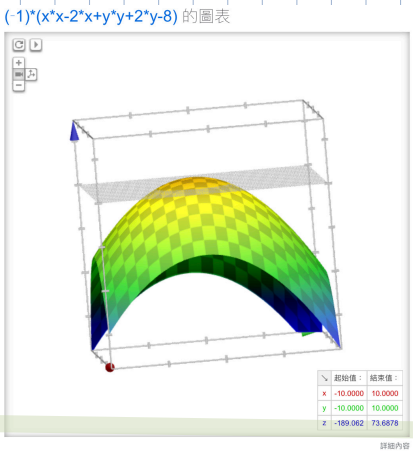
—

```
import random

def hillClimbing(f, x, y, h=0.01):
    failCount = 0                       # 失敗次數歸零
    while (failCount < 10000):          # 如果失敗次數小於一萬次就繼續執行
        fxy = f(x, y)                   # fxy 為目前高度
        dx = random.uniform(-h, h)      # dx 為左右偏移量
        dy = random.uniform(-h, h)      # dy 為前後偏移量
        if f(x+dx, y+dy) >= fxy:        # 如果移動後高度比現在高
            x = x + dx                  # 就移過去
            y = y + dy
            print('x={:.3f} y={:.3f} f(x,y)={:.3f}'.format(x, y, fxy))
            failCount = 0               # 連續失敗次數歸零
        else:                           # 若沒有更高
            failCount = failCount + 1.  # 那就又失敗一次
    return (x,y,fxy)                    # 結束傳回 （已經連續失敗超過一萬次了）

def f(x, y):
    return -1 * ( x*x -2*x + y*y +2*y - 8 )
hillClimbing(f, 0, 0)
```

均等分佈



$= -((x-1)^2 + (y+1)^2 - 10)$



- solution.py

```
class Solution: # 解答的物件模版 (類別)
    def __init__(self, v, step = 0.01):
        self.v = v          # 參數 v 為解答的資料結構
        self.step = step  #  每一小步預設走的距離

    # 以下兩個函數至少需要覆蓋掉一個，否則會無窮遞迴
    def height(self):  #爬山演算法的高度函數
        return -1*self.energy()                    # 高度 = -1 * 能量
```

其他函数

```
    def energy(self):  # 尋找最低點的能量函數
        return -1*self.height()                    #  能量 = -1 * 高度
```

• hillClimbing.py

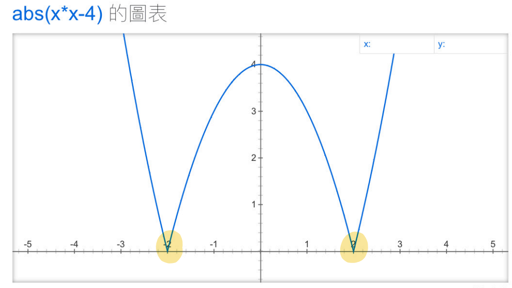
hillClimbing 通用函數

初始解　最大次數　連續失敗次數結束

```
def hillClimbing(s, maxGens, maxFails):   # 爬山演算法的主體函數
    print("start: ", s.str())             # 印出初始解
    fails = 0                             # 失敗次數設為 0
    # 當代數 gen<maxGen，且連續失敗次數 fails < maxFails 時，就持續嘗試尋找更好的解。
    for gens in range(maxGens):
        snew = s.neighbor()               #  取得鄰近的解
        sheight = s.height()              #  sheight=目前解的高度
        nheight = snew.height()           #  nheight=鄰近解的高度
        if  (nheight >= sheight):         #  如果鄰近解比目前解更好
            print(gens, ':', snew.str())  #  印出新的解
            s = snew                      #  就移動過去
            fails = 0                     #  移動成功，將連續失敗次數歸零
        else:                             #  否則
            fails = fails + 1             #  將連續失敗次數加一
        if (fails >= maxFails):
            break
    print("solution: ", s.str())          #  印出最後找到的那個解
    return s                              #  然後傳回。
```

• solutionNumber.py

abs(x*x-4) 的圖表



```
from hillClimbing import hillClimbing  # 引入解答類別
from solution import Solution
import random

class SolutionNumber(Solution):
    def neighbor(self):  # 單變數解答的鄰居函數。
        x = self.v          物件
        dx= self.step  = 0.01               # x:解答 , dx : 移動步伐大小
        xnew = x+dx if random.random() > 0.5  else x-dx  # 用亂數決定向左或向右移動
        return SolutionNumber(xnew)   # 建立新解答並傳回

    def energy(self):                      # 能量函數
        x = self.v                         # x:解答
        return abs(x*x-4)    有2種解        # 能量函數為 |x^2-4|

    def str(self): # 將解答轉為字串，以供印出觀察。
        return "energy({:.3f})={:.3f}".format(self.v, self.energy())
```

- hillClimbingNumber.py

```
from hillClimbing import hillClimbing # 引入爬山演算法類別
from solutionNumber import SolutionNumber # 引入平方根解答類別

# 執行爬山演算法 (從「解答=0.0」開始尋找, 最多十萬代、失敗一千次就跳出)
hillClimbing(SolutionNumber(0.0), 100000, 1000)
```

因亂數決定向左or向右移動，所以解可能会是2 or -2

- solutionArray.py

```
from solution import Solution
from random import random, randint

class SolutionArray(Solution):
    def neighbor(self):                             #  多變數解答的鄰居函數
        nv = self.v.copy()                          #  nv=v.clone()=目前解答的複製品
        i = randint(0, len(nv)-1)                   #  隨機選取一個變數
        if (random() > 0.5):                        #  擲骰子決定要往左或往右移
            nv[i] += self.step
        else:
            nv[i] -= self.step
        return SolutionArray(nv)  #  傳回新建的鄰居解答

    def energy(self):  #  能量函數
        x, y, z =self.v
        return x*x+3*y*y+z*z-4*x-3*y-5*z+8 #  (x^2+3y^2+z^2-4x-3y-5z+8)

    def str(self):         #  將解答轉為字串的函數，以供列印用
        return "energy({:s})={:f}".format(str(self.v), self.energy())
```

- hillClimbingArray.py

```
from hillClimbing import hillClimbing # 引入爬山演算法類別
from solutionArray import SolutionArray # 引入平方根解答類別

# 執行爬山演算法 (最多十萬代、失敗一千次就跳出)。
hillClimbing(SolutionArray([1,1,1]), 100000, 1000)
```

- solutionEquation.py 解聯立方程式

```
"""
A X = B ，求 X 是多少？

範例：題目來源: http://mail.im.tku.edu.tw/~idliaw/LinTup/99ie/99IEntu.pdf

4a+3b+6c=1
```

```
1a+1b+2c=2
2a+1b+3c=-1
"""

from random import random, randint
import numpy as np
from numpy import linalg as LA
from solution import Solution

A = np.array([[4,3,6],[1,1,2],[2,1,3]])
B = np.array([[1,2,-1]]).transpose()

class SolutionEquation(Solution):
    def neighbor(self):                 #  多變數解答的鄰居函數
        nx = self.v.copy()              #  複製目前解的矩陣
        rows = nx.shape[0]
        #  修改了這裡：最多改變 n 個維度(只是某些 n 維的例子可以，無法確定一定可以，除非能證明能量函數只有一個低點)
        for _ in range(rows):           #  原本只改一維，會找不到！
            i = randint(0, rows-1)  #  隨機選取一個變數
            if (random() > 0.5):        #  擲骰子決定要往左或往右移
                nx[i][0] += self.step * random()  #  原本是 nx.m[i][0] += self.step
            else:
                nx[i][0] -= self.step * random()   #  原本是 nx.m[i][0] -= self.step

        return SolutionEquation(nx)             #  傳回新建的鄰居解答。

    def energy(self):    #  能量函數:計算 ||AX-B||，也就是 ||Y-B||
        X = self.v
        Y = A.dot(X)  內積
        return LA.norm(Y-B, 2) 長度 √(a²+b²)

    def str(self):     #  將解答轉為字串的函數，以供列印用。
        return "energy({:s})={:f}".format(str(self.v.transpose()), self.energy())

    @classmethod
    def zero(cls):
        return SolutionEquation(np.zeros((3,1)))
```

[A]x⃗ = B⃗　||Y-B||
[A]x⃗ = Y　[A]x⃗

- hillClimbingEquation.py

```
from hillClimbing import hillClimbing # 引入爬山演算法類別
from solutionEquation import SolutionEquation # 引入平方根解答類別

# 執行爬山演算法 (最多十萬代、失敗一千次就跳出)
hillClimbing(SolutionEquation.zero(), 100000, 、)
```

- solutionScheduling.py

```
from random import random, randint, choice
```

```
from solution import Solution
import numpy as np

courses = [                                    上課時長
{'teacher': '  ', 'name':'       ', 'hours': -1},
{'teacher': '甲', 'name':'機率', 'hours': 2},
{'teacher': '甲', 'name':'線代', 'hours': 3},
{'teacher': '甲', 'name':'離散', 'hours': 3},
{'teacher': '乙', 'name':'視窗', 'hours': 3},
{'teacher': '乙', 'name':'科學', 'hours': 3},
{'teacher': '乙', 'name':'系統', 'hours': 3},
{'teacher': '乙', 'name':'計概', 'hours': 3},
{'teacher': '丙', 'name':'軟工', 'hours': 3},
{'teacher': '丙', 'name':'行動', 'hours': 3},
{'teacher': '丙', 'name':'網路', 'hours': 3},
{'teacher': '丁', 'name':'媒體', 'hours': 3},
{'teacher': '丁', 'name':'工數', 'hours': 3},
{'teacher': '丁', 'name':'動畫', 'hours': 3},
{'teacher': '丁', 'name':'電子', 'hours': 4},
{'teacher': '丁', 'name':'嵌入', 'hours': 3},
{'teacher': '戊', 'name':'網站', 'hours': 3},
{'teacher': '戊', 'name':'網頁', 'hours': 3},
{'teacher': '戊', 'name':'演算', 'hours': 3},
{'teacher': '戊', 'name':'結構', 'hours': 3},
{'teacher': '戊', 'name':'智慧', 'hours': 3}
]

teachers = ['甲', '乙', '丙', '丁', '戊']

rooms = ['A', 'B']

slots = [                                          A教室
'A11', 'A12', 'A13', 'A14', 'A15', 'A16', 'A17',   禮拜一的第七節
'A21', 'A22', 'A23', 'A24', 'A25', 'A26', 'A27',
'A31', 'A32', 'A33', 'A34', 'A35', 'A36', 'A37',
'A41', 'A42', 'A43', 'A44', 'A45', 'A46', 'A47',
'A51', 'A52', 'A53', 'A54', 'A55', 'A56', 'A57',
'B11', 'B12', 'B13', 'B14', 'B15', 'B16', 'B17',
'B21', 'B22', 'B23', 'B24', 'B25', 'B26', 'B27',
'B31', 'B32', 'B33', 'B34', 'B35', 'B36', 'B37',
'B41', 'B42', 'B43', 'B44', 'B45', 'B46', 'B47',
'B51', 'B52', 'B53', 'B54', 'B55', 'B56', 'B57',
] 禮拜五的第一節
  B教室
cols = 7  一天7堂課

def randSlot() :
    return randint(0, len(slots)-1)
```

```
def randCourse() :
    return randint(0, len(courses)-1)


class SolutionScheduling(Solution) :
    def neighbor(self):    # 單變數解答的鄰居函數。
        fills = self.v.copy()
        choose = randint(0, 1)
        if choose == 0: # 任選一個改變
            i = randSlot()
            fills[i] = randCourse()
        elif choose == 1: # 任選兩個交換
            i = randSlot()
            j = randSlot()
            t = fills[i]
            fills[i] = fills[j]
            fills[j] = t
        return SolutionScheduling(fills)                # 建立新解答並傳回。

    def height(self) :      # 高度函數
        courseCounts = [0] * len(courses)
        fills = self.v
        score = 0
        # courseCounts.fill(0, 0, courses.length)
        for si in range(len(slots)):
            courseCounts[fills[si]] += 1
            #                                連續上課:好                隔天:不好       跨越中午:不好
            if si < len(slots)-1 and fills[si] == fills[si+1] and si%7 != 6 and si%7 != 3:0~6 ⇒ 7堂課 0-3⇒4堂課
                score += 0.1
            if si % 7 == 0 and fills[si] != 0: # 早上 8:00: 不好
                score -= 0.12

        for ci in range(len(courses)):
            if (courses[ci]['hours'] >= 0):
                score -= abs(courseCounts[ci] - courses[ci]['hours']) # 課程總時數不對: 不好
        return score

    def str(self) :    # 將解答轉為字串，以供印出觀察。
        outs = []
        fills = self.v
        for i in range(len(slots)):
            c = courses[fills[i]]
            if i%7 == 0:
                outs.append('\n')
            outs.append(slots[i] + ':' + c['name'])
        return 'score={:f} {:s}\n\n'.format(self.energy(), ' '.join(outs))

    @classmethod
    def init(cls):
        fills = [0] * len(slots)
        for i in range(len(slots)):
```

```
            fills[i] = randCourse()
        return SolutionScheduling(fills)
```

- hillClimbingScheduling.py

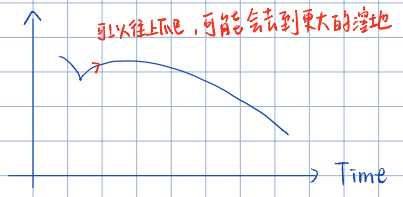
```
from hillClimbing import hillClimbing # 引入爬山演算法類別
from solutionScheduling import SolutionScheduling # 引入平方根解答類別

# 執行爬山演算法 (最多3萬代、失敗一千次就跳出)
hillClimbing(SolutionScheduling.init(), 30000, 1000)
```

初始課表

爬山演算法其實可以解任何問題，深度學能做的爬山也能但速度会慢上一百萬倍

模擬退火法與爬山的差別是能量函数，可以克服局部的陷阱找到最低T

模擬退火法控制不是用失敗次数而是溫度控制




```
import math
import random

def P(e, enew, T): # 模擬退火法的機率函數
    if (enew < e):
        return 1
    else:
        return math.exp((e-enew)/T)

def annealing(s, maxGens) :                 # 模擬退火法的主要函數
    sbest = s                               # sbest:到目前為止的最佳解
    ebest = s.energy()                      # ebest:到目前為止的最低能量
    T = 100                                 # 從 100 度開始降溫
    for gens in range(maxGens):             # 迴圈，最多作 maxGens 這麼多代。
        snew = s.neighbor()                 # 取得鄰居解
        e    = s.energy()                   # e : 目前解的能量
        enew = snew.energy()                # enew : 鄰居解的能量
        T  = T * 0.995                      # 每次降低一些溫度
        if P(e, enew, T)>random.random():   # 根據溫度與能量差擲骰子，若通過
            s = snew                        # 則移動到新的鄰居解
            print("{} T={:.5f} {}".format(gens, T, s.str())) # 印出觀察

        if enew < ebest:                    # 如果新解的能量比最佳解好，則更新最佳解。
            sbest = snew
            ebest = enew

    print("solution: {}", sbest.str())      # 印出最佳解
    return sbest                            # 傳回最佳解
```

在溫度低的時候不会接受爛解
在溫度高的時候会接受爛解