

# Stochastic Search

Howard Hao-Chun Chuang (莊皓鈞)

Professor  
College of Commerce  
National Chengchi University

December, 2024  
Taipei, Taiwan

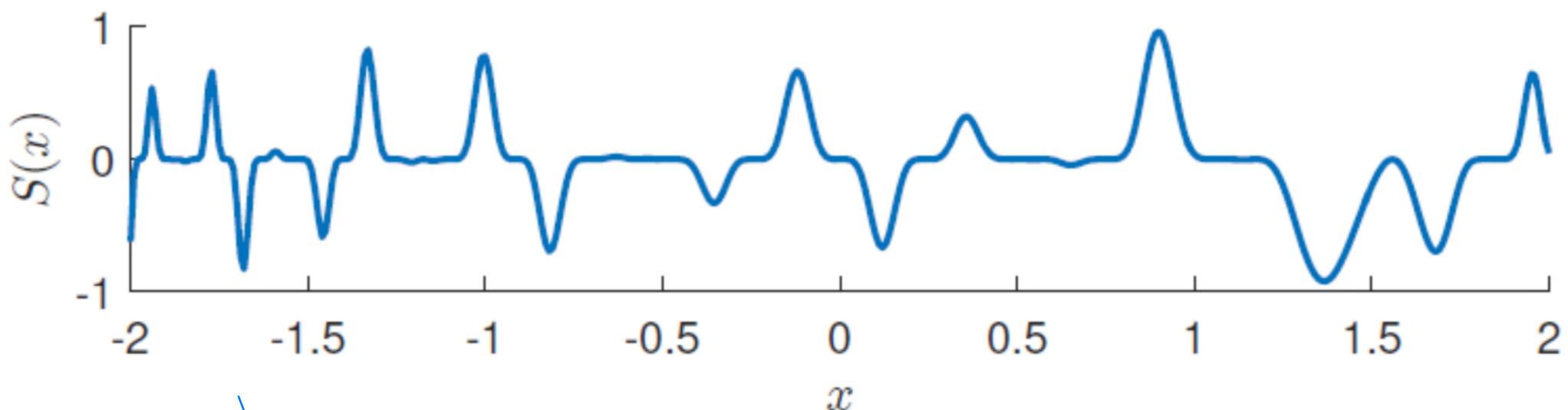


# Difficult Optimization

- We could easily have multiple local minimums/maximums

$$S(x) = \begin{cases} -e^{-x^2/100} \sin(13x - x^4)^5 \sin(1 - 3x^2)^2, & \text{if } -2 \leq x \leq 2, \\ \infty, & \text{otherwise.} \end{cases}$$

*Stochastic Search*  
 $U_{\text{old}} \curvearrowright U_{\text{new}}$



很難找到全域的最大最小



退火 (打鐵燒紅後, 放到水裡)

## Simulated Annealing 適用於多個局部最優解的問題

- $S(x)$  is an arbitrary function to be minimized,  $x$  may take values in continuous or discrete set  $X$

*Simulated annealing* is a Monte Carlo technique for minimization that emulates the physical state of atoms in a metal when the metal is heated up and then slowly cooled down. When the cooling is performed very slowly, the atoms settle down to a minimum-energy state. Denoting the state as  $x$  and the energy of a state as  $S(x)$ , the probability distribution of the (random) states is described by the *Boltzmann pdf*

: if

$$f(x) \propto e^{-\frac{S(x)}{kT}}, \quad x \in X,$$

where  $k$  is Boltzmann's constant and  $T$  is the temperature.

$S(x)$ ：狀態  $x$  的能量 (或目標函數值)

$k$  : Boltzmann 常数 (优化問題中通常設為 1)

$T$  : 當前溫度 (隨時間遞減)

含義：

- 當  $T$  高時，所有狀態  $x$  都有較高的接受概率，因此算法可以在不同狀態之間自由探索。
- 隨著  $T$  降低，分布集中於較低能量的狀態，最終收斂於最小能量狀態。

# 目標 $s(x)$

情況 1.  $\Rightarrow s(Y) < s(X_t)$  比當前狀態  $[s(x_0)]$  低

$$X_{t+1} = Y$$

情況 2.  $\Rightarrow s(Y) > s(X_t)$ .

\* 計算接受概率



接受率 Metropolis - Hastings \*  $\pi(Y)$  是目標分布

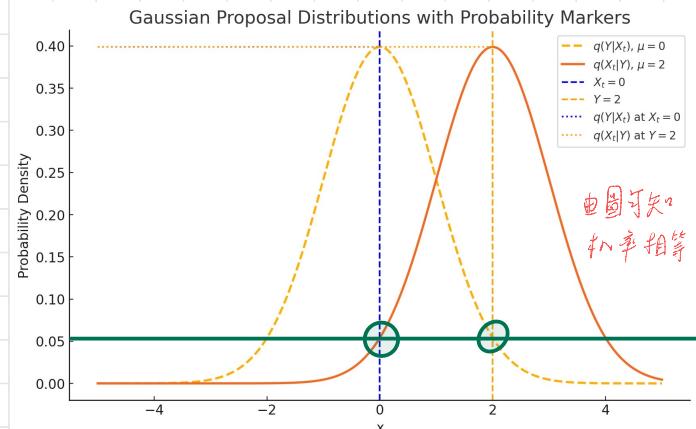
$$\alpha = \min \left( 1, \frac{\pi(Y) \cdot q(X_t | Y)}{\pi(X_t) \cdot q(Y | X_t)} \right)$$

使用對稱提案分布  $q(X_t | Y) = q(Y | X_t)$  → ex: 高斯分布

$$\therefore \alpha = \min \left( 1, \frac{\pi(Y)}{\pi(X_t)} \right)$$

$$\pi(x) \propto s(x) \propto e^{-\frac{s(x)}{T_0}}$$

$$\alpha = \min \left( 1, \frac{e^{-\frac{s(Y)}{T_0}}}{e^{-\frac{s(X_t)}{T_0}}} \right) = \min \left( 1, e^{-\frac{s(Y) - s(X_t)}{T_0}} \right)$$



$$(正態分布) f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$\mu$ : 均值     $\sigma^2$ : 方差

$$q(Y | X_t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(Y-X_t)^2}{2\sigma^2}}$$

⇒ 令  $X_t$  為 均值     $\mu = X_t = 0$

$$q(X_t | Y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(X_t-Y)^2}{2\sigma^2}}$$

⇒ 令  $Y$  為 均值     $\mu = Y = 2$ .

代入  $s(x), s(t)$ , 算出  $\alpha$  與  $v(0,1)$  比較

小於  $\Rightarrow$  接受

大於  $\Rightarrow$  反對

# Simulated Annealing

The idea of simulated annealing is to create a sequence of points  $X_1, X_2, \dots$  that are approximately distributed according to pdfs  $f_{T_1}(\mathbf{x}), f_{T_2}(\mathbf{x}), \dots$ , where  $T_1, T_2, \dots$  is a sequence of “temperatures” that decreases (is “cooled”) to 0 — known as the *annealing schedule*. If each  $X_t$  were sampled *exactly* from  $f_{T_t}$ , then  $X_t$  would converge to a global minimum of  $S(\mathbf{x})$  as  $T_t \rightarrow 0$ . However, in practice sampling is *approximate* and convergence to a global minimum is not assured. A generic simulated annealing algorithm is as follows.

*Temperature* →

---

## Algorithm 3.4.1: Simulated Annealing

---

**input:** Annealing schedule  $T_0, T_1, \dots$ , function  $S$ , initial value  $\mathbf{x}_0$ .

**output:** Approximations to the global minimizer  $\mathbf{x}^*$  and minimum value  $S(\mathbf{x}^*)$ .

- 1 Set  $X_0 \leftarrow \mathbf{x}_0$  and  $t \leftarrow 1$ .
  - 2 **while** not stopping **do**
  - 3     Approximately simulate  $X_t$  from  $f_{T_t}(\mathbf{x})$ .
  - 4      $t \leftarrow t + 1$
  - 5 **return**  $X_t, S(X_t)$
- 



---

### Algorithm 3.4.2: Simulated Annealing with a Random Walk Sampler

---

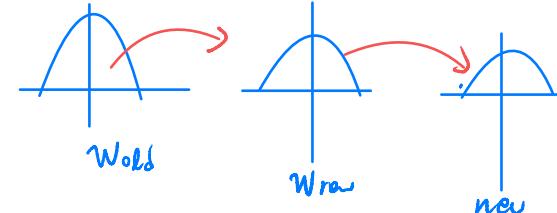
**input:** Objective function  $S$ , starting state  $X_0$ , initial temperature  $T_0$ , number of iterations  $N$ , symmetric proposal density  $q(y|x)$ , constant  $\beta$ .

**output:** Approximate minimizer and minimum value of  $S$ .

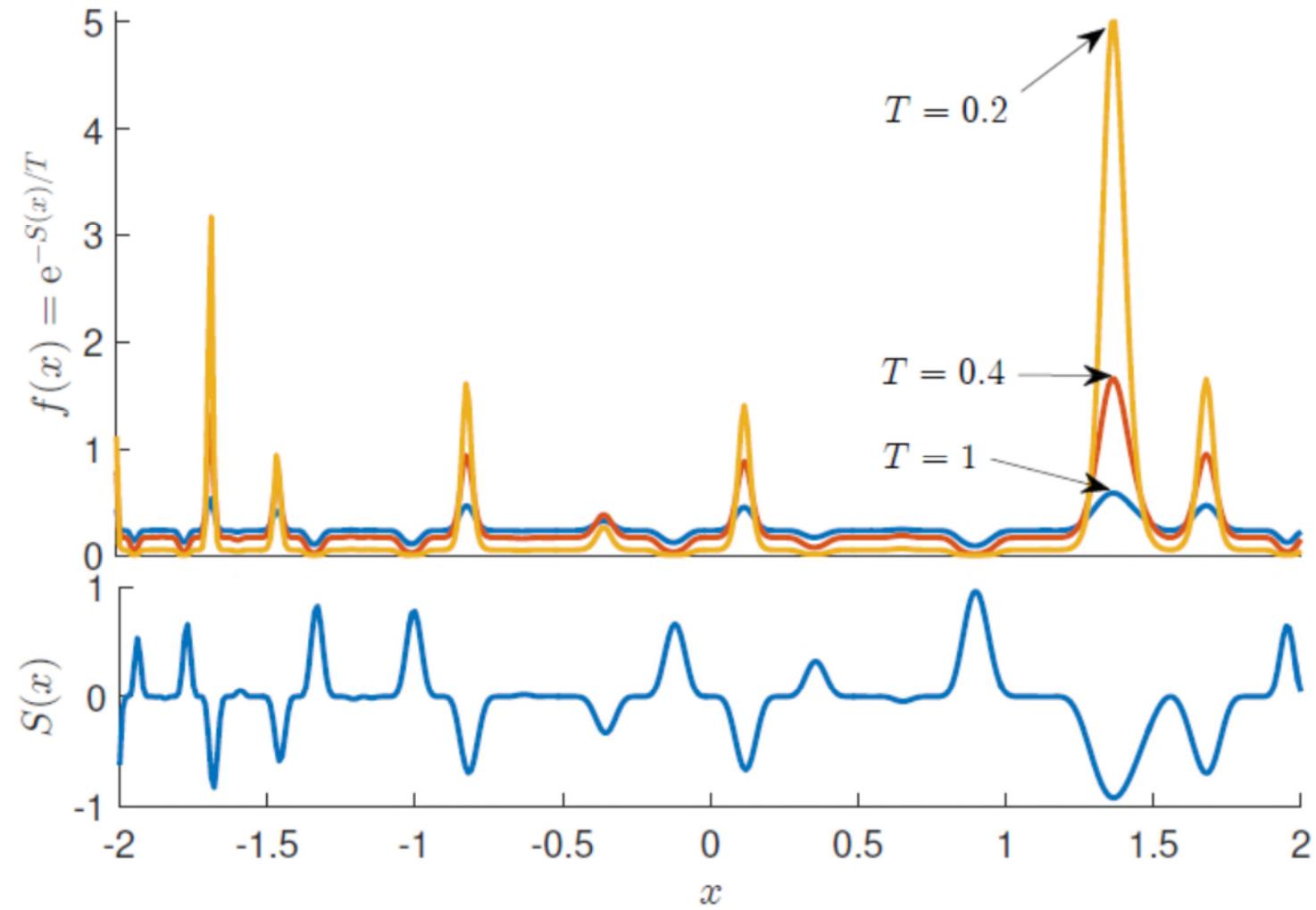
```
1 for  $t = 0$  to  $N - 1$  do
2     Simulate a new state  $Y$  from the symmetric proposal  $q(y|X_t)$ .
3     if  $S(Y) < S(X_t)$  then *:  $s(Y) < s(X_t)$  接受
4          $X_{t+1} \leftarrow Y$            if  $s(Y) \geq s(X_t)$ 
5     else 不如预期时           $e^{- (s(Y) - s(X_t)) / T_t}$ 
6         Draw  $U \sim U(0, 1)$ .    需要不要接受
7         if  $U \leq e^{-(S(Y) - S(X_t)) / T_t}$  then
8              $X_{t+1} \leftarrow Y$ 
9         else                      判断要不要等
10             $X_{t+1} \leftarrow X_t$ 
11
12      $T_{t+1} \leftarrow \beta T_t$        $\beta$  降温速度
13 return  $X_N$  and  $S(X_N)$ 
```

均匀分布  $U(0, 1)$   
中抽取

$$U_{\text{new}} \leftarrow \text{Normal}(\mu = \mu_{\text{old}}, \sigma)$$



# Simulated Annealing



2.

## Multivariable Optimization

- Consider the following assets in 2017/11/10 ~2020/11/10  
APPLE, FACEBOOK, AMAZON, S&P500, NASDAQ, WWE (?)

\$ Budget

+ Allowance

Build a portfolio: Six continuous decision variables  $x$  in  $[0, 1]$

$$\sum x = 1$$

$$x \leq 1 \quad x \geq 0$$

Daily return is  $\{\text{price}(t) - \text{price}(t-1)\}/\text{price}(t-1)$

Maximize the Sharpe ratio =  $E[\text{Daily Return}] / (\text{Var}[\text{Daily Return}])^{0.5}$

see [https://en.wikipedia.org/wiki/Sharpe\\_ratio](https://en.wikipedia.org/wiki/Sharpe_ratio)

- Min objective =  $-1 * \text{Sharpe} + \lambda * (\text{constraint})$

How can we accommodate key constraints?

$$((x_1 + x_2 + x_3 + x_4 + x_5 + x_6) - 1)^2$$

$$\sum \max(0, x[i] - 1)^2, \text{ for } i=1, 2, \dots, 6$$

$$\sum \max(0, -x[i])^2, \text{ for } i=1, 2, \dots, 6$$

Price  $\text{MadCap}$

100 200

105 200

90 250

Return



Asset:

Apple  $\Rightarrow X_1$

Facebook  $\Rightarrow X_2$

Amazon  $\Rightarrow X_3$

S&P 500  $\Rightarrow X_4$

NASDAQ  $\Rightarrow X_5$

WWE  $\Rightarrow X_6$

\*: 建立資產組合，最大化 Sharpe Ratio

$$\sum x_i = 1$$

$$\text{Sharpe Ratio} = \frac{\text{E}[Daily\ Return]}{\sqrt{\text{Var}(Daily\ Return)}}$$

$$\text{Daily Return} = \frac{\text{price}(t) - \text{price}(t-1)}{\text{price}(t-1)}$$

$$\text{Sharpe Ratio} = \frac{\text{E}[R_p - R_f]}{\sigma_p}$$

- $\mathbb{E}[R_p - R_f]$ : 投資組合的超額回報 (Portfolio Excess Return)，這是投資組合的期望回報 ( $R_p$ ) 減去無風險收益率 ( $R_f$ )。

- $R_p$ : 投資組合的期望收益率 (如年化或日收益率)。
- $R_f$ : 無風險利率 (通常以國債利率或銀行存款利率為參考)。
- $\sigma_p$ : 投資組合回報的標準差，表示投資組合的波動性或風險。

$$\text{Min Objective} = -1 * \text{Sharpe Ratio} + \lambda \cdot (\text{penalty})$$

$$\text{Penalty} = \lambda_1 \cdot \left( \sum_{i=1}^6 x_i - 1 \right)^2 + \lambda_2 \cdot \sum_{i=1}^6 \max(0, x_i - 1)^2 + \lambda_3 \cdot \sum_{i=1}^6 \max(0, -x_i)^2$$

#### 1. 資金總和約束：

目標是要求資產的分配總和為 1，即：

$$\text{Constraint: } \sum_{i=1}^6 x_i = 1$$

懲罰項：

$$\text{Penalty} = \left( \sum_{i=1}^6 x_i - 1 \right)^2$$

懲罰機制解釋：

- 如果  $\sum x_i = 1$ ，則 Penalty = 0。

• 如果  $\sum x_i \neq 1$ ，則 Penalty > 0，違反最嚴，懲罰越大。

• 例如：

- $\sum x_i = 1.2$ ，則懲罰  $(1.2 - 1)^2 = 0.04$ 。
- $\sum x_i = 0.5$ ，則懲罰  $(0.5 - 1)^2 = 0.25$ 。

作用：

目標函數會自動傾向於選擇  $\sum x_i = 1$  的解，因為這樣能使懲罰項為 0。

#### 2. 範圍約束 $x_i \in [0, 1]$

懲罰項分為兩部分：

- $x_i$  超過上界時：

$$\text{Penalty (upper)} = \sum_{i=1}^6 \max(0, x_i - 1)^2$$

- 如果  $x_i \leq 1$ ，懲罰為 0。

- 如果  $x_i > 1$ ，懲罰  $(x_i - 1)^2$ 。

- $x_i$  小於下界時：

$$\text{Penalty (lower)} = \sum_{i=1}^6 \max(0, -x_i)^2$$

- 如果  $x_i \geq 0$ ，懲罰為 0。

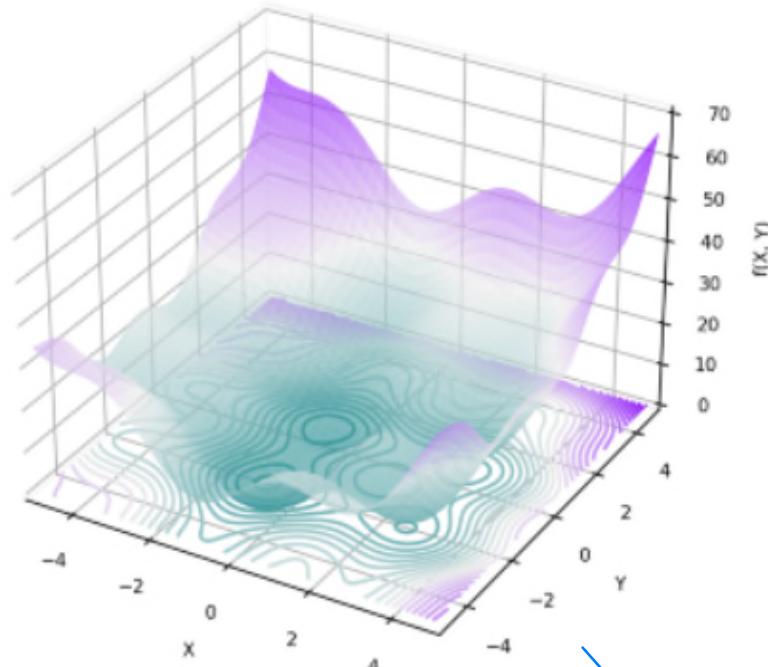
- 如果  $x_i < 0$ ，懲罰  $(-x_i)^2$ 。

# 粒子群 Particle Swarm Optimization, PSO

粒子群最佳化

- Inspired by social behavior 每隻鳥可以想成「粒子 (particle)」  
We make decisions by self-experience & group-experience

A group of birds is searching for food, how to move in a valley?



$f_{best}$

$g_{best}$

1. 自我經驗 (Self-Experience): 1. 自己過去最佳位置

每隻鳥會記住它自己曾經發現的食物位置 (最佳經驗)。

2. 群體經驗 (Group-Experience): 2. 群體中找到的最佳位置

鳥群中的個體會交流資訊，靠近整體找到的最佳位置。

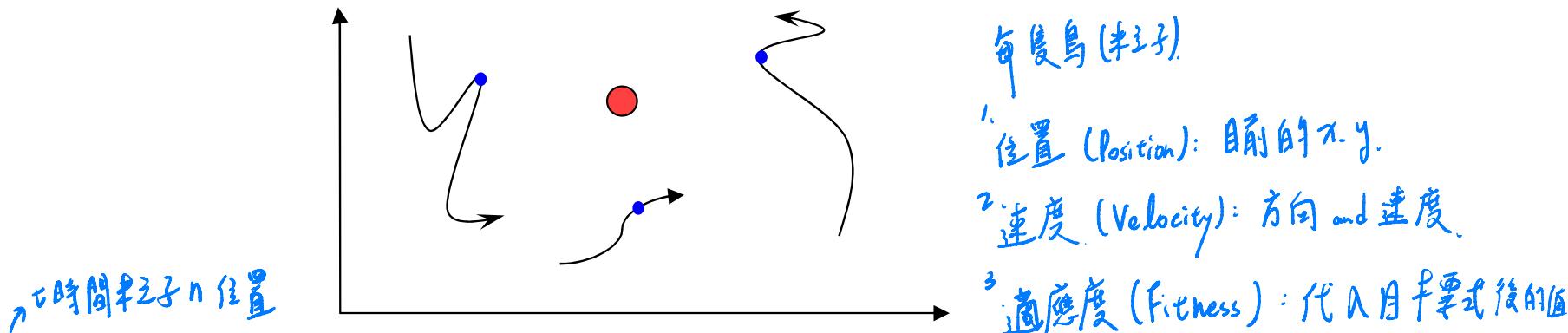
$$f(x,y) = x^2 + (y+1)^2 - 5\cos(3x/2 + 3/2) - 3\cos(2y - 3/2)$$

$$f(x,y) = x^2 + (y+1)^2 - 5 \cos\left(\frac{3x}{2} + \frac{3}{2}\right) - 3 \cos\left(2y - \frac{3}{2}\right)$$

↓  
(x,y) 代表粒子位置

# Particle Swarm Optimization, PSO

- Let's simplify the case into 2 dimensions & only 3 particles



- $X_{n,t}$  : travel route of  $n$  particles in time period  $t$

粒子 aaron  $P(aaron) = [X_{a,1}, X_{a,2}, X_{a,3} \dots X_{a,t}]$ , min at  $X_{a,2}$

粒子 betty  $P(betty) = [X_{b,1}, X_{b,2}, X_{b,3} \dots X_{b,t}]$ , min at  $X_{b,3}$

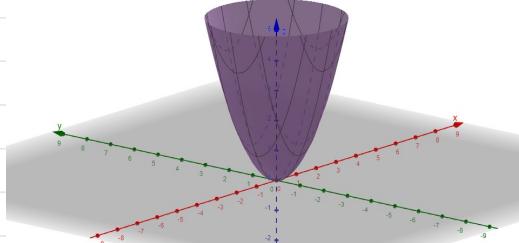
粒子 cathy  $P(cathy) = [X_{c,1}, X_{c,2}, X_{c,3} \dots X_{c,t}]$ , min at  $X_{c,t}$

⇒ 每個粒子會參考過去最佳位置  $p_{best}$   
同時也會參考全局最佳位置  $g_{best}$

- Next position for each particle is calculated by:

$$X_{n,t+1} = X_{n,t} + V_{n,t+1}$$





#### 4. 舉例說明

假設我們在一個簡單的 2 維空間中尋找函數  $f(x, y) = x^2 + y^2$  的最小值 ( 最低點是  $(0, 0)$  ) 。

##### 初始設定：

- Aaron、Betty 和 Cathy 的初始位置分別是：
  - Aaron :  $X_{a,1} = (5, 4)$
  - Betty :  $X_{b,1} = (-3, 7)$
  - Cathy :  $X_{c,1} = (6, -2)$

##### 步驟 1：計算適應度

代入目標函數  $f(x, y) = x^2 + y^2$  :

- Aaron 的適應度 :  $f(5, 4) = 5^2 + 4^2 = 41$
- Betty 的適應度 :  $f(-3, 7) = (-3)^2 + 7^2 = 58$
- Cathy 的適應度 :  $f(6, -2) = 6^2 + (-2)^2 = 40$

目前 :

- Aaron 的最佳位置 :  $(5, 4)$
- Betty 的最佳位置 :  $(-3, 7)$
- Cathy 的最佳位置 :  $(6, -2)$

全局最佳位置  $g_{best}$  是 Cathy 的  $(6, -2)$  ( 因為  $f = 40$  最小 ) 。

##### 步驟 2：更新速度與位置

使用速度公式計算粒子下一步的速度 :

$$V_{n,t+1} = wV_{n,t} + c_1r_1(p_{best} - X_{n,t}) + c_2r_2(g_{best} - X_{n,t})$$

- $w$  : 慢性權重，控制移動穩定性。
- $c_1, c_2$  : 學習因子。
- $r_1, r_2$  : 隨機數。

例如 :

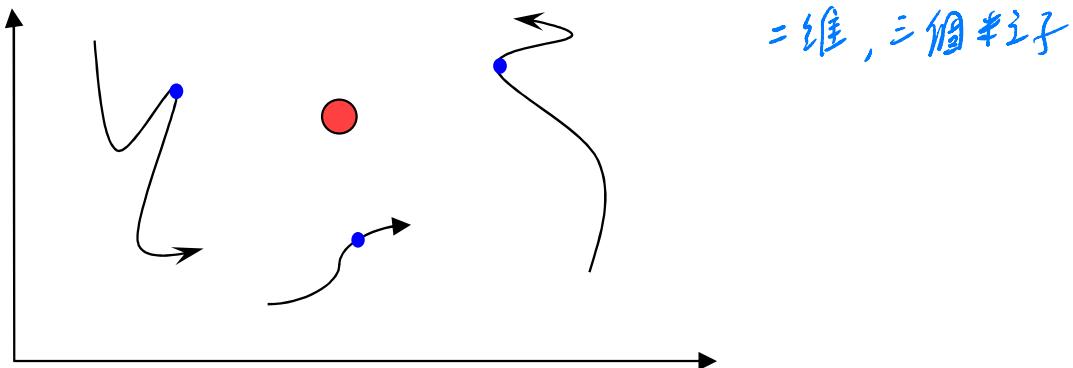
- Aaron 根據自身最佳位置和全局最佳位置  $(6, -2)$  調整方向，下一步可能移動到  $(4, 2)$  。
- Betty 和 Cathy 也會根據各自的經驗和全局最佳來更新位置。

##### 步驟 3：迭代收斂

粒子會反覆更新位置和速度，逐步靠近目標函數的最小值。最終所有粒子會聚集到  $(0, 0)$  。

# Particle Swarm Optimization, PSO

- Let's simplify the case into 2 dimensions & only 3 particles



- Now, we have to decide  $V_{n,t+1}$  by  $pbest$  and  $gbest$

$$V_{n,t+1} = w^* V_{n,t} + c1 * r1(V_{pbest}) + c2 * r2(V_{gbest})$$

$r1$  &  $r2 \sim \text{Uniform}(0, 1)$

Take particle *aaron* for instance

$pbest$  : best self-experience:  $X_{a,2}$

$gbest$  : best group-experience:  $X_{c,t}$

$$V_{a,t+1} = w^* V_{a,t} + c1 * r1 * (X_{a,2} - X_{a,t}) + c2 * r2 * (X_{c,t} - X_{a,t})$$

$$V_{i+1} = w V_i + c_1 r_1 (p_{best} - x_i) + c_2 r_2 (g_{best} - x_i)$$

$$x_i = x_i + v_i$$

- $w$  : 慣性權重 (控制粒子的移動穩定性)。
- $c_1, c_2$  : 學習因子 (自我經驗和群體經驗的影響程度)。
- $r_1, r_2$  : 隨機因子 (引入隨機性)。



## 最上面的公式

$$V_{n,t+1} = w \cdot V_{n,t} + c1 \cdot r1 \cdot (V_{pbest}) + c2 \cdot r2 \cdot (V_{gbest})$$

特點：

1. 這裡使用的符號是  $V_{pbest}$  和  $V_{gbest}$ ，看起來是速度 ( $V$ ) 的概念。
2. 這種寫法將  $pbest$  和  $gbest$  表述為速度方向或差異，而不是直接計算位置差異。
3. 比較抽象：公式上省略了具體的位置向量  $X$ 。

## 最下面的公式

$$V_{a,t+1} = w \cdot V_{a,t} + c1 \cdot r1 \cdot (X_{a,2} - X_{a,t}) + c2 \cdot r2 \cdot (X_{c,t} - X_{a,t})$$

特點：

1. 具體化了位置的差異：

- $X_{a,2}$ ：個人最佳位置  $pbest$ 。
- $X_{c,t}$ ：群體最佳位置  $gbest$ 。
- $X_{a,t}$ ：當前位置。

## 2. 速度更新公式解釋

公式如下：

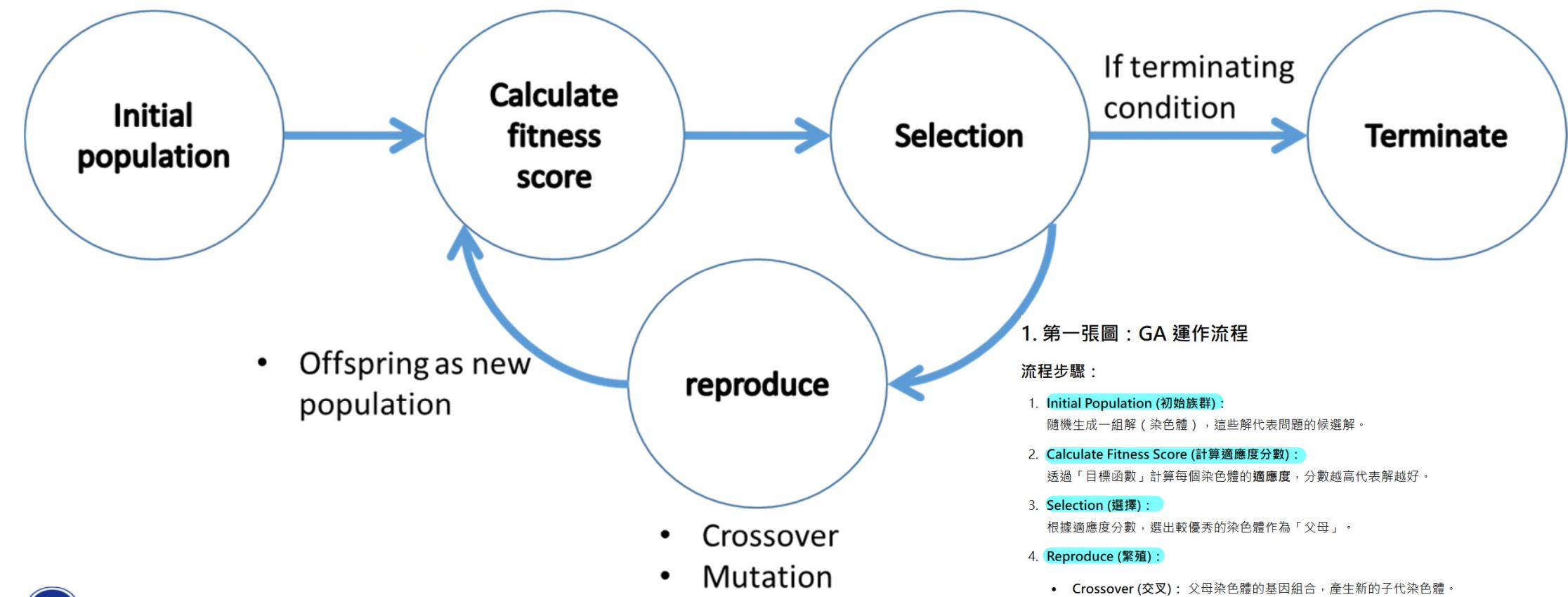
$$V_{n,t+1} = w \cdot V_{n,t} + c1 \cdot r1 \cdot (pbest - X_{n,t}) + c2 \cdot r2 \cdot (gbest - X_{n,t})$$

其中：

- $V_{n,t+1}$ ：時間  $t + 1$  時的速度。
- $w$ ：慣性權重，用來平衡粒子的前進趨勢與當前探索。
- $c1$ ：自我學習因子，表示粒子向自己最佳位置  $pbest$  靠攏的速度權重。
- $r1$ ：介於  $[0, 1]$  的隨機值，增加搜尋隨機性。
- $c2$ ：群體學習因子，表示粒子向群體最佳位置  $gbest$  靠攏的速度權重。
- $r2$ ：同樣介於  $[0, 1]$  的隨機值。
- $pbest$ ：個人最佳位置。
- $gbest$ ：群體最佳位置。
- $X_{n,t}$ ：當前位置。

# Genetic Algorithms (GA)

自然選擇, 遺傳算制。



## 1. 第一張圖：GA 運作流程

流程步驟：

1. Initial Population (初始族群)：  
隨機生成一組解（染色體），這些解代表問題的候選解。
2. Calculate Fitness Score (計算適應度分數)：  
透過「目標函數」計算每個染色體的適應度，分數越高代表解越好。
3. Selection (選擇)：  
根據適應度分數，選出較優秀的染色體作為「父母」。
4. Reproduce (繁殖)：
  - Crossover (交叉)：父母染色體的基因組合，產生新的子代染色體。
  - Mutation (突變)：對部分基因進行隨機變異，增加多樣性。  
子代形成新的族群。
5. Terminate (終止)：  
若達到終止條件（如最大迭代次數或找到最優解），停止演算法。



# Genetic Algorithms (GA)

- An extremely popular optimization algorithm
  1. Initial population with `pop_size` pop-size 個 chromosome
  2. Calculate fitness score of population 染色体
  3. Randomly select `pop_size` of chromosome with replacement

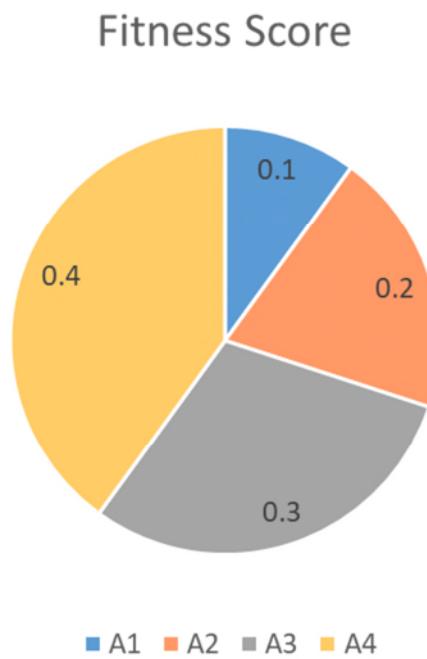


# Genetic Algorithms (GA)

- Selection

Chromosome	Fitness Score	Proportion	選擇的機率
A1	10	0.1	1/100
A2	20	0.2	2/100
A3	30	0.3	3/100
A4	40	0.4	4/100

輪盤選擇法  
Roulette wheel selection



# Genetic Algorithms (GA)

- An extremely popular optimization algorithm
  1. Initial population with `pop_size` 建立 pop-size 個 chromosome
  2. Calculate fitness score of population
  3. Randomly select `pop_size` of chromosome with replacement
  4. if  $rand_1 \leq pcrossover$ , make population into pairs and do crossover
  5. If  $rand_2 \leq pmutation$ , for each child, do mutate
  6. select elitism of highest fitness scores from previous population & randomly select (`pop_size` – elitism) of children as new population

## hyper-parameters

`pop_size` : population size

`pcrossover` : probability of crossover

`pmutation` : probability of mutation

`elitism` : number of best fitness to survive

`maxiter` : max iteration



# 1. 第一張圖：GA 運作流程

## 流程步驟：

### 1. Initial Population (初始族群) :

隨機生成一組解（染色體），這些解代表問題的候選解。

### 2. Calculate Fitness Score (計算適應度分數) :

透過「目標函數」計算每個染色體的適應度，分數越高代表解越好。

### 3. Selection (選擇) :

根據適應度分數，選出較優秀的染色體作為「父母」。

### 4. Reproduce (繁殖) :

- Crossover (交叉) : 父母染色體的基因組合，產生新的子代染色體。

- Mutation (突變) : 對部分基因進行隨機變異，增加多樣性。

子代形成新的族群。

### 5. Terminate (終止) :

若達到終止條件（如最大迭代次數或找到最優解），停止演算法。

## 舉例說明：簡單基因演算法流程

問題：找到目標函數  $f(x) = -x^2 + 10x$  的最大值， $x \in [0, 10]$ 。

### 步驟：

#### 1. 初始族群：

隨機生成 4 個染色體： $x_1 = 2, x_2 = 5, x_3 = 7, x_4 = 8$ 。

#### 2. 適應度計算：

計算每個染色體的適應度：

$$f(x) = -x^2 + 10x$$

- $x_1 = 2, f(2) = -4 + 20 = 16$
- $x_2 = 5, f(5) = -25 + 50 = 25$
- $x_3 = 7, f(7) = -49 + 70 = 21$
- $x_4 = 8, f(8) = -64 + 80 = 16$ 。

#### 3. 選擇：

使用輪盤選擇法，根據適應度比例選出父代。

#### 4. 交叉：

$p_{crossover} = 0.8$ ：隨機選擇父代進行交叉，生成新的子代（例如  $x = 6$ ）。

#### 5. 突變：

$p_{mutation} = 0.1$ ：對子代的基因進行隨機變異（例如  $x = 6$  變成  $x = 6.2$ ）。

#### 6. 菁英保留：

保留適應度最高的個體，形成下一代族群。

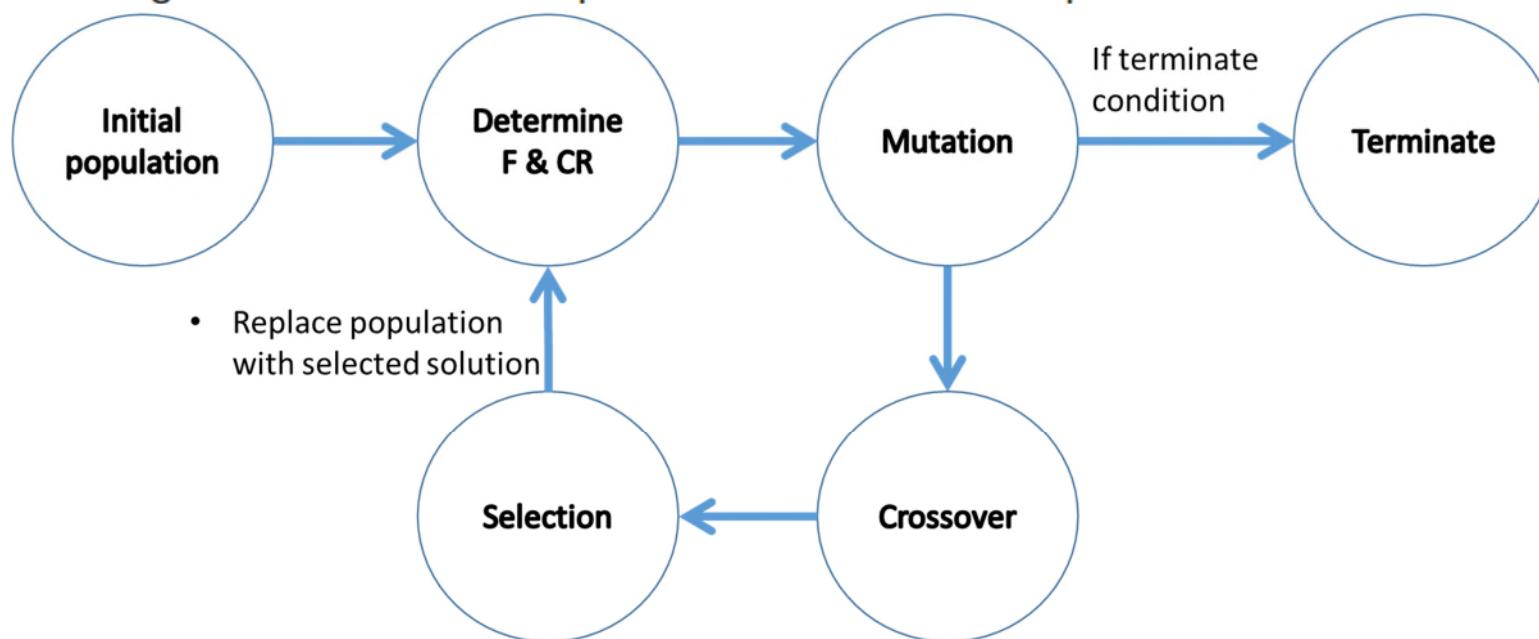
### 結果：

經過多次迭代，演算法會收斂到最佳解  $x = 5$  左右，因為此時  $f(x)$  最大。

# Differential Evolution (DE)

種群 (Population), 突異 (mutation)

- Choose the parameters  $NP \geq 4$ ,  $CR \in [0, 1]$ , and  $F \in [0, 2]$ .
  - $NP$  is the population size, i.e. the number of candidate agents or "parents"; a typical setting is  $10n$ .
  - The parameter  $CR \in [0, 1]$  is called the *crossover probability* and the parameter  $F \in [0, 2]$  is called the *differential weight*. Typical settings are  $F = 0.8$  and  $CR = 0.9$ .
  - Initialize all agents  $\mathbf{x}$  with random positions in the search-space.



## 第一張圖：DE 的整體流程

### 步驟解析

#### 1. 參數設定：

- $NP$  (種群大小)：候選解的數量，通常設定為問題維度的 10 倍（如  $10n$ ）。
- $CR$  (交叉機率)：控制父代與變異解的組合程度，範圍為  $[0,1]$ 。典型值是  $CR = 0.9$ 。
- $F$  (差分權重)：控制變異強度的參數，範圍為  $[0,2]$ ，典型值是  $F = 0.8$ 。

#### 2. 初始化族群 (Initial Population)：

- 隨機生成所有候選解（稱為 agents）的位置，分布於問題的搜尋空間內。

#### 3. 變異 (Mutation)：

- 使用「差分策略」生成新的候選解，通過多個父解之間的差異進行變異。

#### 4. 交叉 (Crossover)：

- 結合變異後的候選解和原始解，生成新的候選解。

#### 5. 選擇 (Selection)：

- 將新的候選解與原始解進行比較，如果新解的適應度較優或相等，則替換原始解。

#### 6. 終止條件 (Terminate)：

- 若達到最大迭代次數或解的誤差滿足要求，則演算法結束。

## 步驟 2：交叉 (Crossover)

將變異解  $\mathbf{v} = (3.2, 2.8)$  和原始解  $\mathbf{x}_1 = (2, 3)$  進行交叉。

- 交叉機率  $CR = 0.9$ ：表示 90% 的機率選擇變異解的分量。

對於每個維度  $i$ ：

- 產生一個隨機數  $r_i \sim U(0, 1)$ 。
- 如果  $r_i < CR$ ，則取  $v_i$ ；否則保留  $x_i$ 。

假設隨機數  $r_1 = 0.8, r_2 = 0.4$ ：

- 對於  $x_1 : r_1 = 0.8 < 0.9$ ，取  $v_1 = 3.2$ 。
- 對於  $x_2 : r_2 = 0.4 < 0.9$ ，取  $v_2 = 2.8$ 。

因此，交叉後的向量  $\mathbf{u} = (3.2, 2.8)$ 。

## 步驟 3：選擇 (Selection)

比較交叉後的向量  $\mathbf{u} = (3.2, 2.8)$  與原始向量  $\mathbf{x}_1 = (2, 3)$ ：

- 計算適應度  $f(\mathbf{u})$ ：

$$f(\mathbf{u}) = (3.2 - 5)^2 + (2.8 - 5)^2 = (-1.8)^2 + (-2.2)^2 = 3.24 + 4.84 = 8.08$$

- 比較  $f(\mathbf{u})$  和  $f(\mathbf{x}_1)$ ：

- $f(\mathbf{u}) = 8.08$
- $f(\mathbf{x}_1) = 13$

因為  $f(\mathbf{u}) < f(\mathbf{x}_1)$ ，用  $\mathbf{u} = (3.2, 2.8)$  替換  $\mathbf{x}_1$ 。

## 步驟 1：變異 (Mutation)

對每個候選解  $\mathbf{x}_i$ ，選擇 3 個不同的解  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  進行變異。

舉例處理  $\mathbf{x}_1 = (2, 3)$ ：

- 隨機選擇  $\mathbf{a} = \mathbf{x}_2 = (4, 6), \mathbf{b} = \mathbf{x}_3 = (7, 1), \mathbf{c} = \mathbf{x}_4 = (8, 5)$ 。

- 使用變異公式生成新的向量  $\mathbf{v}$ ：

$$\mathbf{v} = \mathbf{a} + F \cdot (\mathbf{b} - \mathbf{c})$$

- $F = 0.8$ ：差分權重

- 計算  $\mathbf{b} - \mathbf{c}$ ：

$$\mathbf{b} - \mathbf{c} = (7 - 8, 1 - 5) = (-1, -4)$$

- 計算  $F \cdot (\mathbf{b} - \mathbf{c})$ ：

$$F \cdot (\mathbf{b} - \mathbf{c}) = 0.8 \cdot (-1, -4) = (-0.8, -3.2)$$

- 計算  $\mathbf{v}$ ：

$$\mathbf{v} = \mathbf{a} + F \cdot (\mathbf{b} - \mathbf{c}) = (4, 6) + (-0.8, -3.2) = (3.2, 2.8)$$

新的向量  $\mathbf{v} = (3.2, 2.8)$ 。

# Differential Evolution (DE) 差分進化演算法

- For each agent  $\mathbf{x}$  in the population do:
  - Pick three agents  $\mathbf{a}$ ,  $\mathbf{b}$ , and  $\mathbf{c}$  from the population at random, they must be distinct from each other as well as from agent  $\mathbf{x}$ . ( $\mathbf{a}$  is called the "base" vector.)
  - Pick a random index  $R \in \{1, \dots, n\}$  where  $n$  is the dimensionality of the problem being optimized.
  - Compute the agent's potentially new position  $\mathbf{y} = [y_1, \dots, y_n]$  as follows:
    - For each  $i \in \{1, \dots, n\}$ , pick a uniformly distributed random number  $r_i \sim U(0, 1)$
    - If  $r_i < CR$  or  $i = R$  then set  $y_i = a_i + F \times (b_i - c_i)$  otherwise set  $y_i = x_i$ .  
(Index position  $R$  is replaced for certain.)
  - If  $f(\mathbf{y}) \leq f(\mathbf{x})$  then replace the agent  $\mathbf{x}$  in the population with the improved or equal candidate solution  $\mathbf{y}$ .

[https://en.wikipedia.org/wiki/Differential\\_evolution](https://en.wikipedia.org/wiki/Differential_evolution)



<sup>Minimum-Bidding</sup>  
Project : Profit =  $(\beta - \zeta) * \underset{\text{Decision}}{\text{Prob}}(\underline{A} > \beta)$

<sup>Maximum Bidding</sup>  
 $A = \text{Min}(\text{others' Bid})$

Juan Soto: Payoff =  $((\text{Gain} - \beta) * \underset{\text{Decision}}{\text{Prob}}(\underline{\beta} > A))$   
 $A = \text{Max}(\underline{\text{Yankees}}, \underline{\text{Red Sox}}, \underline{\text{Dodge}}, \dots)$

