基因演算法
(Genetic Algorithm)
Part 1

A Brief History of Evolutionary Computation

A Brief History of Evolutionary Computation

- The idea of using simulated evolution to solve engineering and design problems have been around since the 1950's (Fogel, 2000).
  - Bremermann, 1962
  - Box, 1957
  - Friedberg, 1958

- However, it was not until the early 1960's that we began to see three influential forms of EC emerge (Back et al, 1997):
  - Evolutionary Programming (Lawrence Fogel, 1962),
  - Genetic Algorithms (Holland, 1962)
  - Evolution Strategies (Rechenberg, 1965 & Schwefel, 1968)

A Brief History of Evolutionary Computation (cont.)

- The designers of each of the EC techniques saw that their particular problems could be solved via simulated evolution.
  - Fogel was concerned with solving prediction problems.
  - Rechenberg & Schwefel were concerned with solving parameter optimization problems.
  - Holland was concerned with developing robust adaptive systems.

A Brief History of Evolutionary Computation (cont.)

- Each of these researchers successfully developed appropriate ECs for their particular problems independently.

- In the US, Genetic Algorithms have become the most popular EC technique due to a book by David E. Goldberg (1989) entitled, “Genetic Algorithms in Search, Optimization & Machine Learning”.

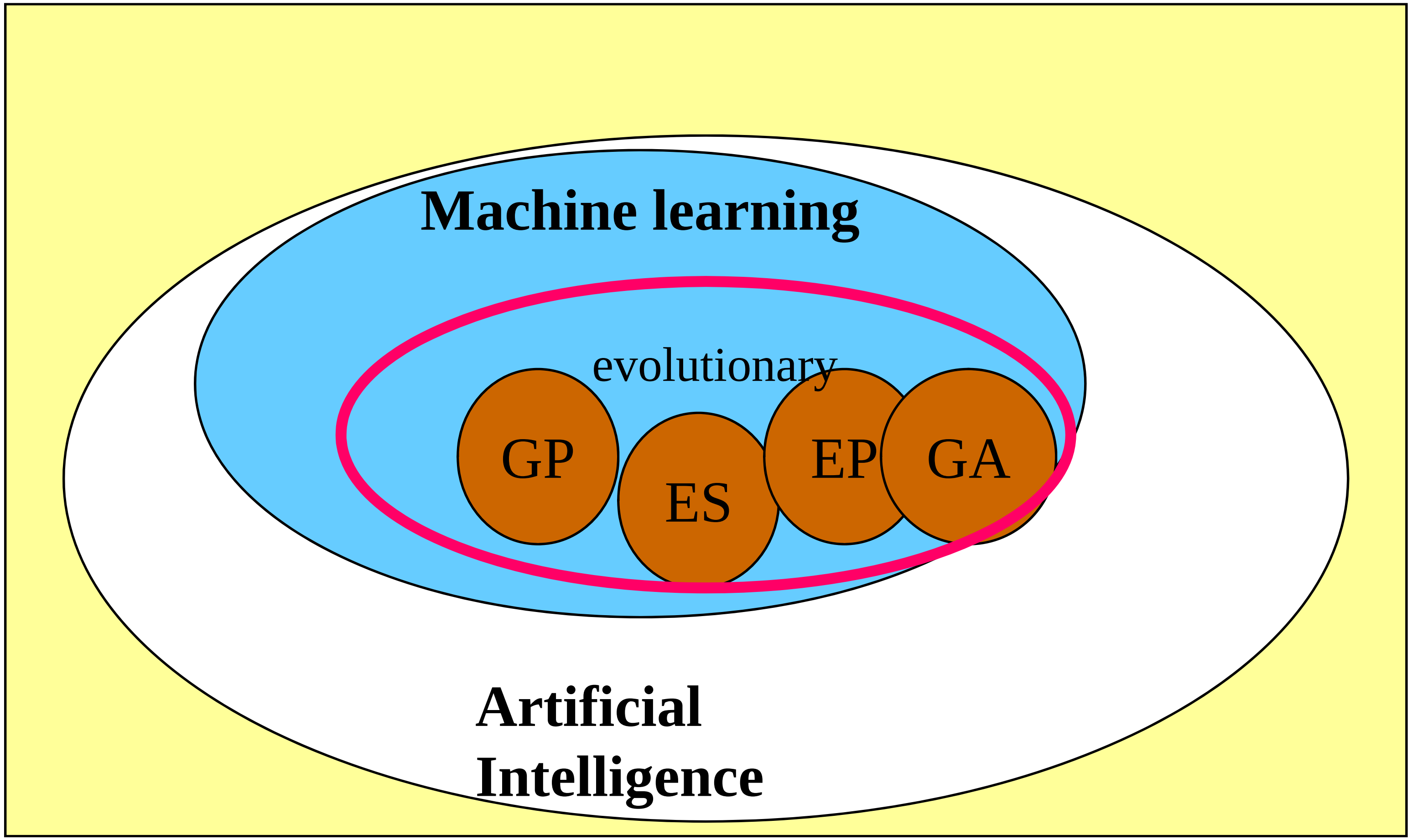
- This book explained the concept of Genetic Search in such a way the a wide variety of engineers and scientist could understand and apply.

A Brief History of Evolutionary Computation (cont.)

- However, a number of other books helped fuel the growing interest in EC:
  - *Lawrence Davis', "Handbook of Genetic Algorithms", (1991),*
  - *Zbigniew Michalewicz' book (1992), "Genetic Algorithms + Data Structures = Evolution Programs.*
  - *John R. Koza's "Genetic Programming" (1992), and*
  - *D. B. Fogel's 1995 book entitled, "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.*

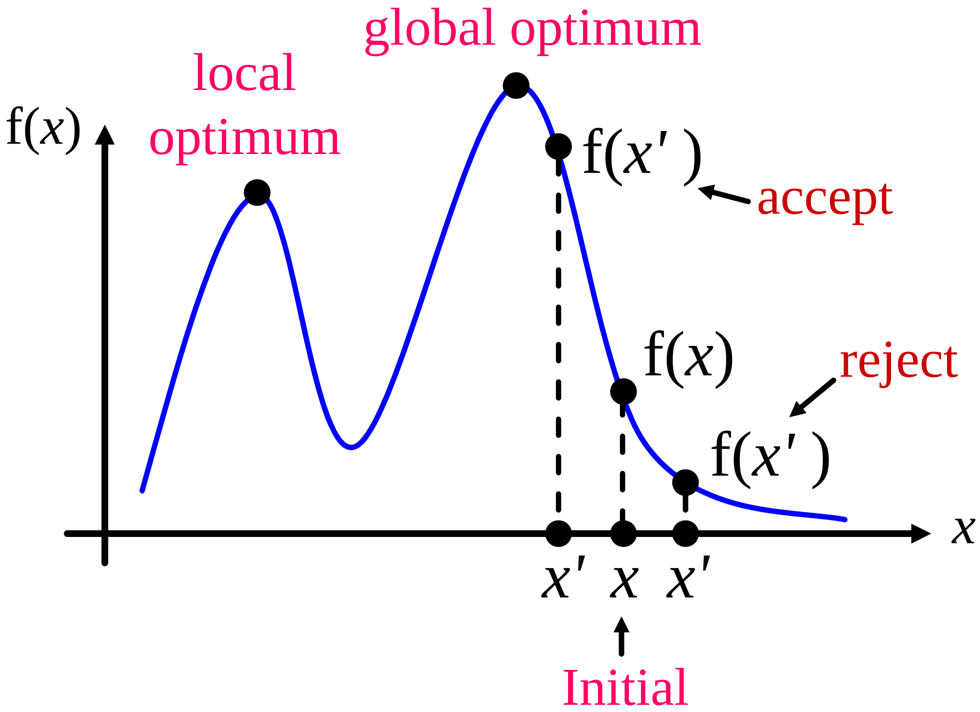
- These books not only fueled interest in EC but they also were instrumental in bringing together the EP, ES, and GA concepts together in a way that fostered unity and an explosion of new and exciting forms of EC.

Scope

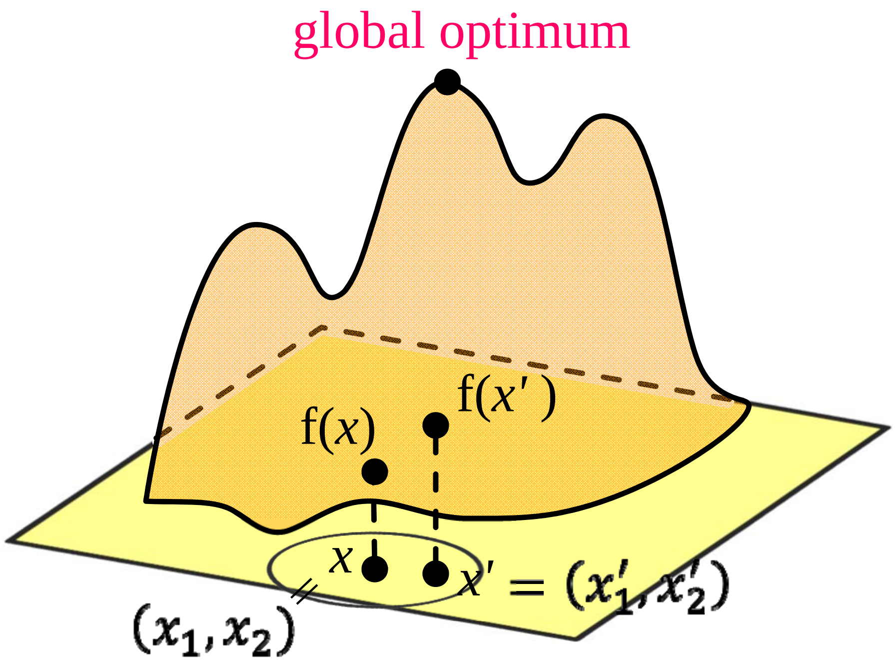

Machine learning
evolutionary
GP
ES
EP
GA
Artificial
Intelligence

Basic Idea of GA

1D



2D



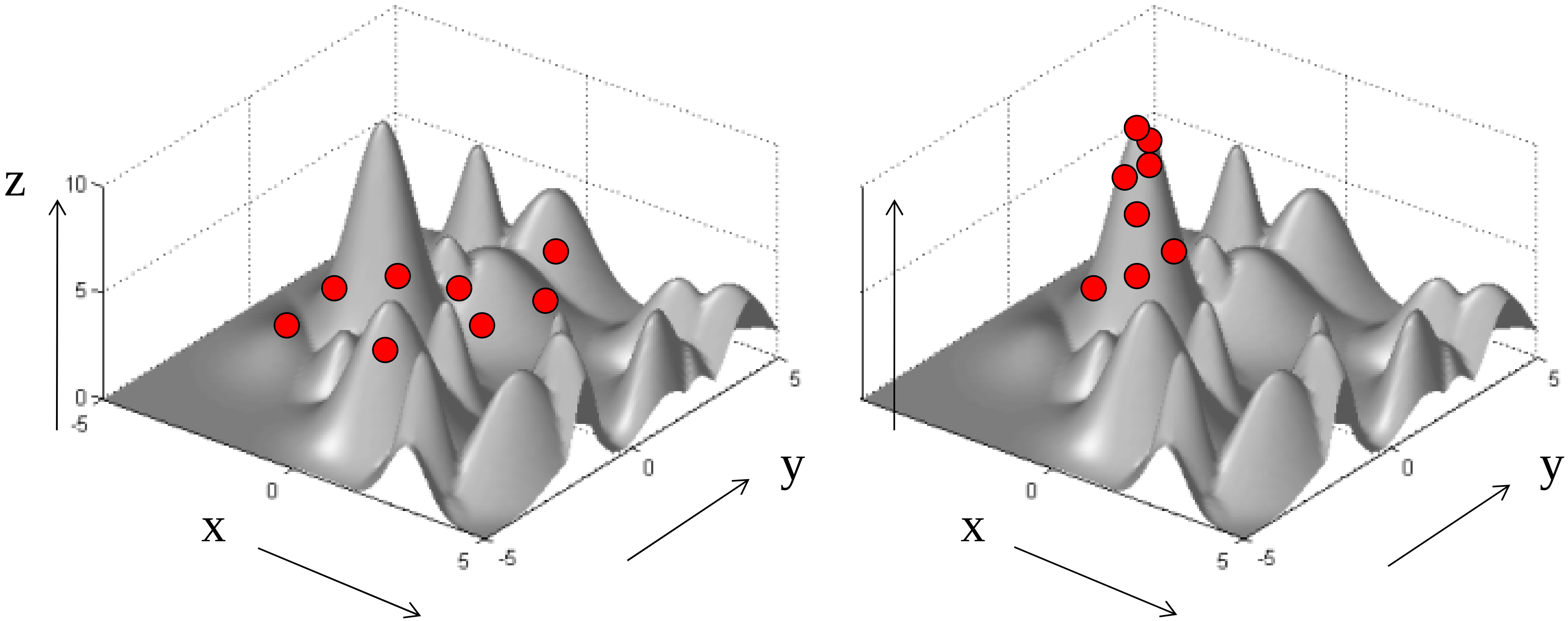
n-D space

$$x = (x_1, x_2, \ldots, x_n)$$

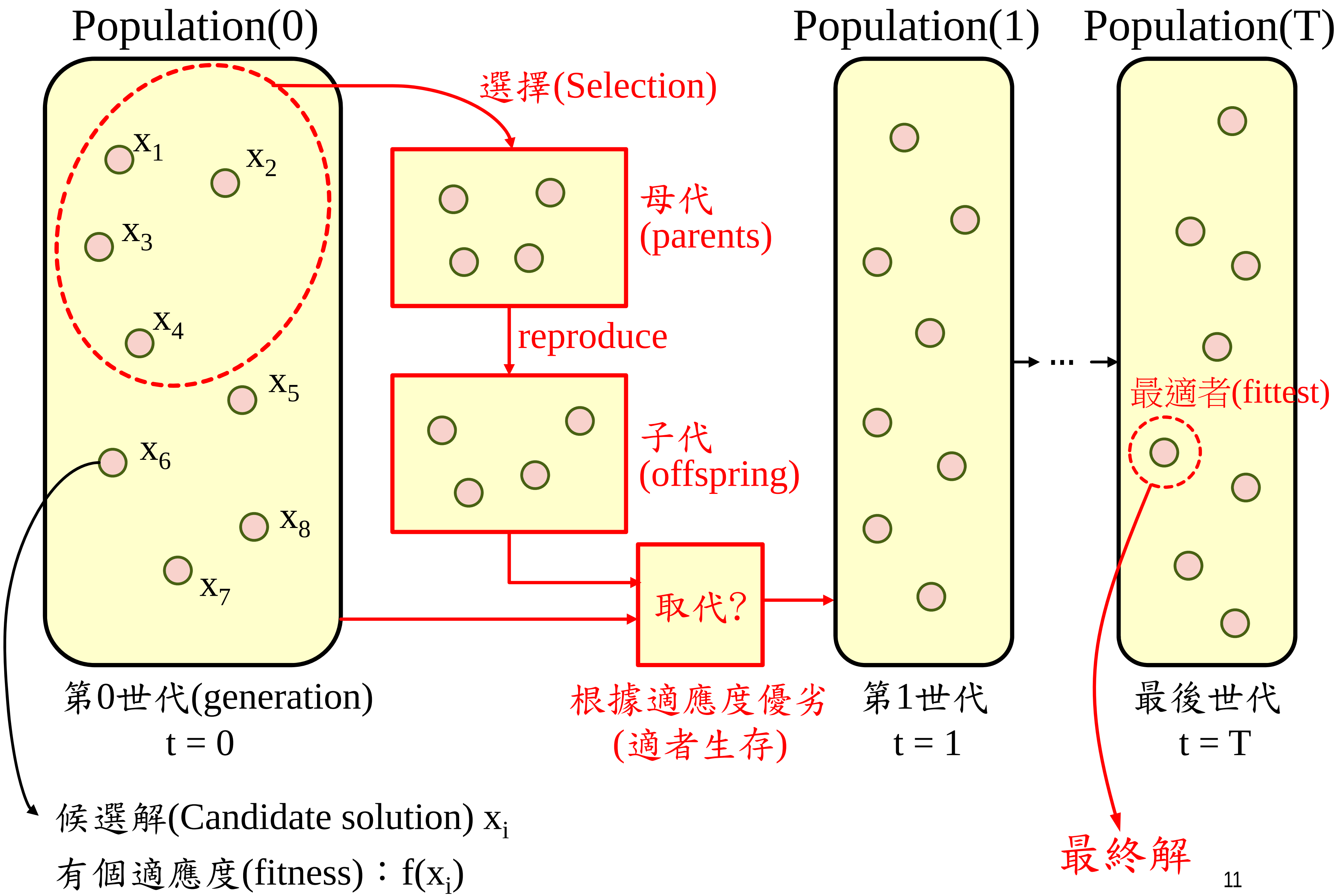
$$\downarrow$$

$$\ldots$$

Population-based metaheuristics

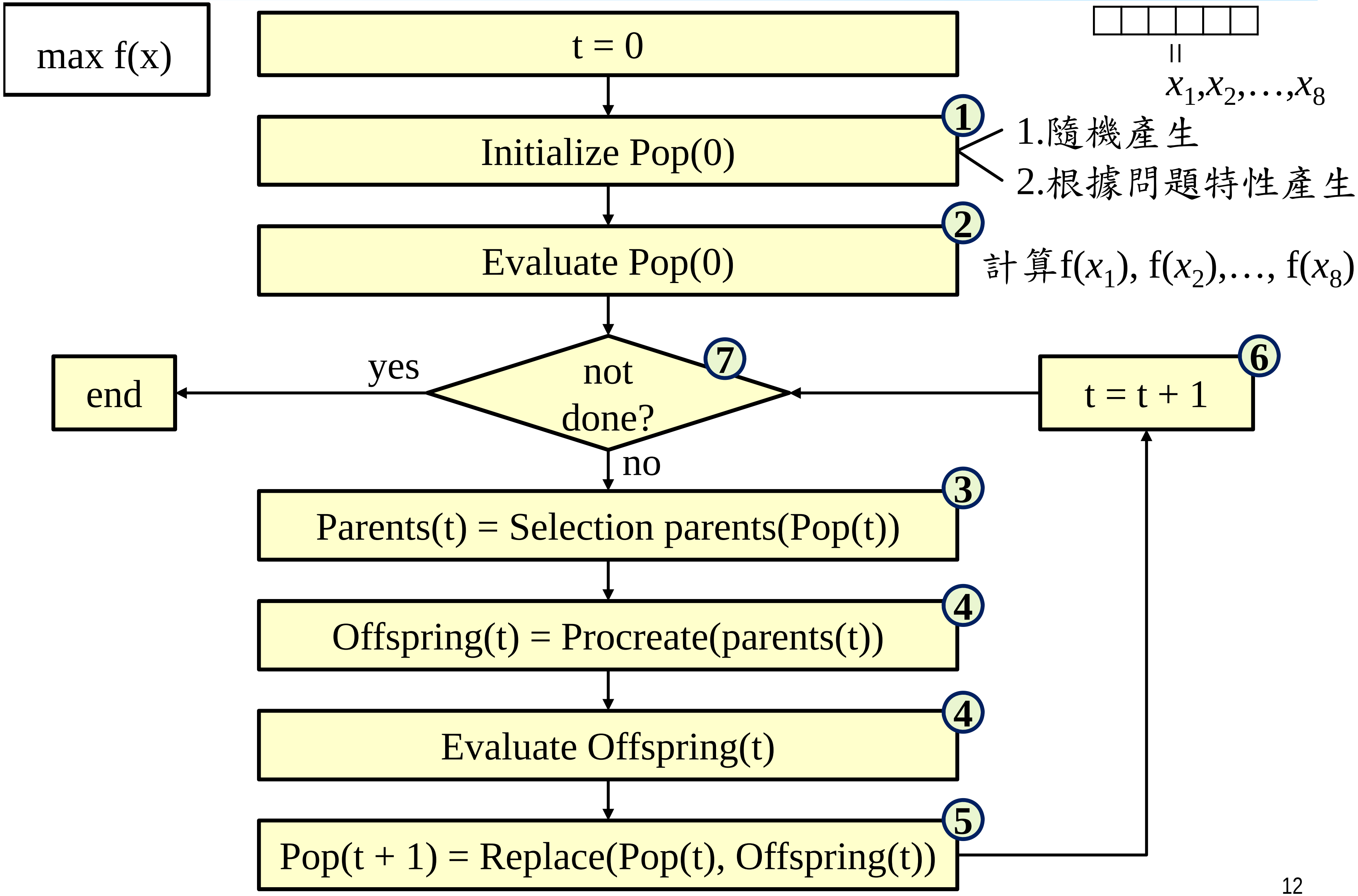


示意圖



Population(0)
Population(1)
Population(T)
選擇(Selection)
x1
x2
x3
x4
x5
x6
x7
x8
母代
(parents)
reproduce
子代
(offspring)
取代?
最適者(fittest)
第0世代(generation)
t = 0
根據適應度優劣
(適者生存)
第1世代
t = 1
最後世代
t = T
候選解(Candidate solution) xi
有個適應度(fitness)：f(xi)
最終解

基因演算法流程圖



max f(x)
t = 0
Initialize Pop(0)
1.隨機產生
2.根據問題特性產生
$x_1, x_2, \ldots, x_8$
Evaluate Pop(0)
計算 $f(x_1), f(x_2), \ldots, f(x_8)$
not done?
yes
end
no
t = t + 1
Parents(t) = Selection parents(Pop(t))
Offspring(t) = Procreate(parents(t))
Evaluate Offspring(t)
Pop(t + 1) = Replace(Pop(t), Offspring(t))
1
2
3
4
4
5
6
7

基因演算法之虛擬碼

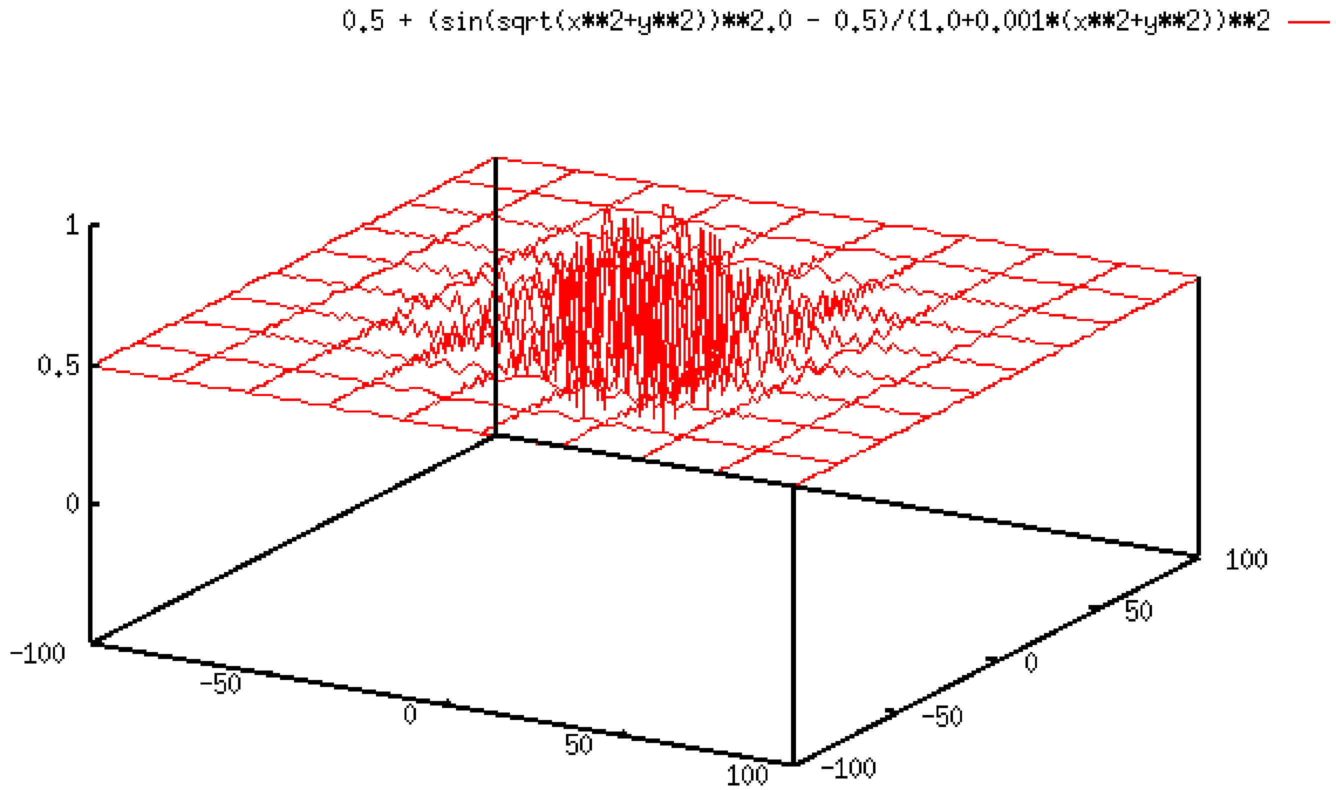
```
t = 0;
Initialize Pop(t);
Evaluate Pop(t);
While (Not Done)
{
   Parents(t) = Select_Parents(Pop(t));
  Offspring(t) = Procreate(Parents(t));
  Evaluate(Offspring(t));
  Pop(t+1)= Replace(Pop(t),Offspring(t));
  t = t + 1;
}
```

A Simple Example

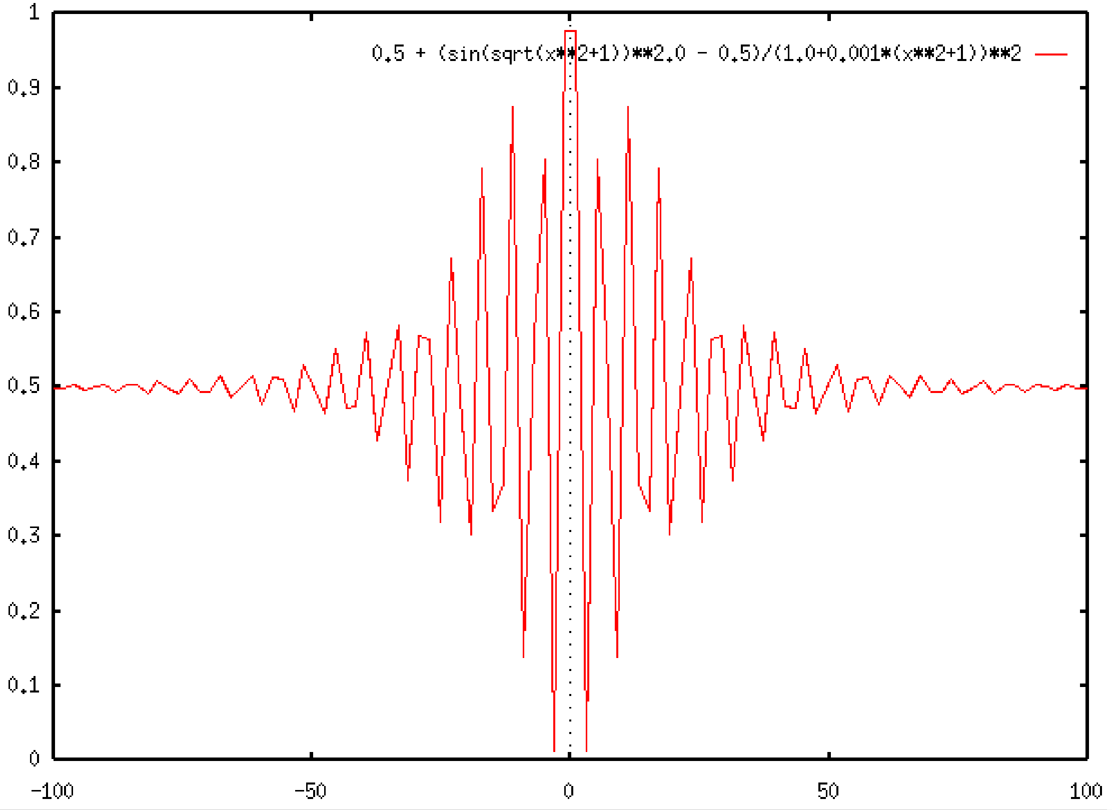
A Simple Example

- Let's walk through a simple example!
- Let's say you were asked to solve the following problem:
  - Maximize:
  - $f(x,y) = 0.5 + (\sin(\mathrm{sqrt}(x^2+y^2))^2 - 0.5) / (1.0 + 0.001(x^2+y^2))^2$
  - Where x and y are take from [-100.0,100.0]
  - You must find a solution that is greater than 0.99754, and you can only evaluate a total of 4000 candidate solutions (CSs)
- This seems like a difficult problem.
  - It would be nice if we could see what it looks like!
  - This may help us determine a good algorithm for solving it.

A 3D view of f(x,y)



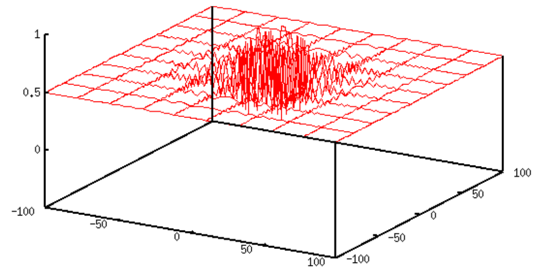
Looking at only one dimension f(x,1.0)

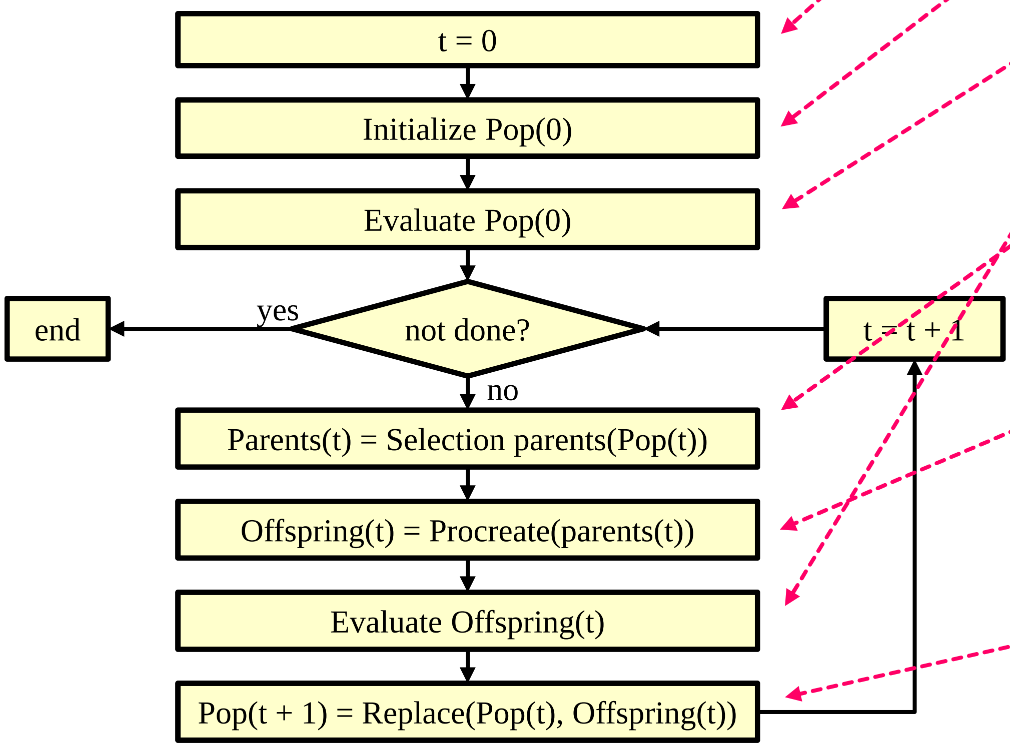


基因演算法流程圖

Max f(x,y)

$= 0.5 + (\sin(\mathrm{sqrt}(x^2+y^2))^2 - 0.5) / (1.0 + 0.001(x^2+y^2))^2$





1. 編碼： (x_i,y_i)

初始化： (x_i,y_i)隨機從[-100,100] × [-100,100]中取一點

2. 適應度： $fit_i = f(x_i,y_i)$

3. 選擇： 隨機從目前的population中選出一對父母(x_{mom}, y_{mon})和(x_{dad}, y_{dad})

4. 繁衍：

$x_{kid} = rnd(x_{mom}, x_{dad}) + N_x(0,\sigma)$

$y_{kid} = rnd(y_{mom}, y_{dad}) + N_y(0,\sigma)$

5. 取代：

$Pop(t+1) = \{Pop(t) - \{worst\}\} \cup \{kid\}$

A Simple Example

t = 0;

Initialize Pop(t); /* of P individuals */

Evaluate Pop(t);

while (t <= 4000){

  Select_Parent($<x_{mom}, y_{mom}>$); /* Randomly */

  Select_Parent($<x_{dad}, y_{dad}>$); /* Randomly */

  Create_Offspring($<x_{kid}, y_{kid}>$):

  $x_{kid} = rnd(x_{mom}, x_{dad}) + N_x(0,\sigma);$

  $y_{kid} = rnd(y_{mom}, y_{dad}) + N_y(0,\sigma);$

  fit_{kid} = Evaluate($<x_{kid}, y_{kid}>$);

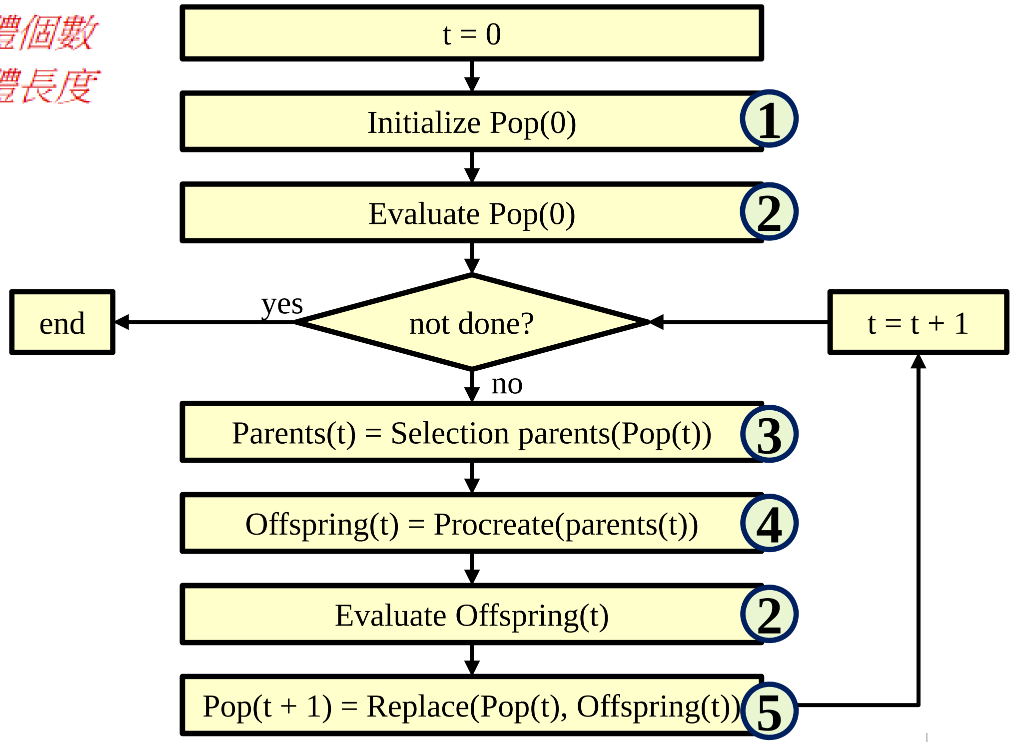
  Pop(t+1) = Replace(worst,kid);$\{Pop(t)-\{worst\}\}\cup\{kid\}$

  t = t + 1;

}

```python
NUM_ITERATION = 100          # 世代數(迴圈數)

NUM_CHROME = 20              # 染色體個數
NUM_BIT = 2                  # 染色體長度
```




```python
# ==== 主程式 ====
pop = initPop()              # 初始化 pop
pop_fit = evaluatePop(pop)   # 算 pop 的 fit

for i in range(NUM_ITERATION) :
    parent = selection(pop)                      # 挑父母
    kid = reproduction(parent)                   # 生子
    kid_fit = fitFunc(kid)                       # 算子代的 fit
    pop, pop_fit = replace(pop, pop_fit, kid, kid_fit)    # 取代

    bestIndex = np.argmax(pop_fit)               # 找此世代最佳解的索引值
    print('iteration %d: x = %s, y = %f'    %(i, pop[bestIndex], pop_fit[bestIndex]))
```

```python
NUM_ITERATION = 100          # 世代數(迴圈數)

NUM_CHROME = 20              # 染色體個數
NUM_BIT = 2                  # 染色體長度
```

1. **編碼**： (x_i,y_i)

   初始化：(x_i,y_i)隨機從$[-100,100] \times [-100,100]$中取一點

```python
# ①
def initPop():               # 初始化群體
    # 產生 NUM_CHROME * NUM_BIT 個[-100, 100]之間的隨機數
    return np.random.uniform(low=-100, high=100, size=(NUM_CHROME,NUM_BIT))
```

2. **適應度**： Max f(x,y)

   $= 0.5 + (\sin(\mathrm{sqrt}(x^2+y^2))^2 - 0.5) / (1.0 + 0.001(x^2+y^2))^2$

```python
# ②
def fitFunc(x):              # 適應度函數
    return 0.5 + ((math.sin(math.hypot(x[0], x[1])))**2 - 0.5) \
           / (1.0 + 0.001 * ( x[0]**2 + x[1]**2 ) )**2

def evaluatePop(p):          # 評估群體之適應度
    return [fitFunc(p[i]) for i in range(len(p))]
```

```python
# ==== 主程式 ====
pop = initPop()              ① # 初始化 pop
pop_fit = evaluatePop(pop)   ② # 算 pop 的 fit

for i in range(NUM_ITERATION) :
    parent = selection(pop)   ③                       # 挑父母
    kid = reproduction(parent)  ④                     # 生子
    kid_fit = fitFunc(kid)  ②                         # 算子代的 fit
    pop, pop_fit = replace(pop, pop_fit, kid, kid_fit)  ⑤    # 取代

    bestIndex = np.argmax(pop_fit)                    # 找此世代最佳解的索引值
    print('iteration %d: x = %s, y = %f'    %(i, pop[bestIndex], pop_fit[bestIndex]))
```

```python
SIGMA = 0.2                        # 生成子代時用到的干擾
```

```python
# ==== 基因演算法會用到的函式 ====
def fitFunc(x):                    # 適應度函數
    return 0.5 + ((math.sin(math.hypot(x
            / (1.0 + 0.001 * ( x[0]**2 + x[1]**2 ) )**2

def initPop():                     # 初始化群體
    # 產生 NUM_CHROME * NUM_BIT 個[-100, 1
    return np.random.uniform(low=-100, hi

def evaluatePop(p):                # 評估群體之適應度
    return [fitFunc(p[i]) for i in range(len(p))]

def selection(p):                  # 隨機找兩個父母
    [i, j] = np.random.choice(NUM_CHROME, 2, replace=False)  # 任選兩個index
    return [p[i], p[j]]
```

```python
def reproduction(p):               # 繁衍子代
    return [np.random.uniform(np.min([p[0][j], p[1][j]]), np.max([p[0][j], p[1][j]])) \
            + np.random.uniform(low=-SIGMA, high=SIGMA) for j in range(NUM_BIT)]
```

```python
def replace(p, p_fit, k, k_fit):                    # 適者生存
    worstIndex = np.argmin(p_fit)
    p[worstIndex] = k
    p_fit[worstIndex] = k_fit

    return p, p_fit
```

1

2

3

4

5

1. 編碼： (x_i,y_i)

初始化： (x_i,y_i) 隨機從 $[-100,100] \times [-100,100]$ 中取一點

2. 適應度： Max f(x,y)

$= 0.5 + (\sin(\mathrm{sqrt}(x^2+y^2))^2 - 0.5) / (1.0 + 0.001(x^2+y^2))^2$

3. 選擇： 隨機從目前的 population 中選出一對父母

4. 繁衍： $x_{kid} = rnd(x_{mom}, x_{dad}) + N_x(0,\sigma)$

$y_{kid} = rnd(y_{mom}, y_{dad}) + N_y(0,\sigma)$

5. 取代： $Pop(t+1) = \{Pop(t) - \{worst\}\} \cup \{kid\}$

Exercise

- Benchmark functions:

Test functions	Feasible spaces	n	$f_{\min}$
$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} \left(-x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)\right)$	$[-500, 500]^n$	30	$-418.983n$
$f_2(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5.12, 5.12]^n$	30	0
$f_3(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$ $+20 + \exp(1)$	$[-32, 32]^n$	30	0
$f_4(\boldsymbol{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	30	0
$f_5(\boldsymbol{x}) = \frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i - 1)^2\,[1 + 10\sin^2(\pi y_{i+1})] +$ $(y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),$ where $y_i = 1 + \frac{1}{4}(x_i + 1)$ and $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	$[-50, 50]^n$	30	0
$f_6 = \sum_{i=1}^{n}\left[\sum_{j=1}^{n}(\chi_{ij}\sin\omega_j + \psi_{ij}\cos\omega_j) - \sum_{j=1}^{n}(\chi_{ij}\sin x_j + \psi_{ij}\cos x_j)\right]^2,$ where χ_{ij} and ψ_{ij} are random intergers in [-100,100], and ω_j is a random number in $[-\pi, \pi]$	$[-\pi, \pi]^n$	100	0
$f_7(\boldsymbol{x}) = \sum_{i=1}^{n-1}\left[100\,(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right]$	$[-5, 10]^n$	30	0
$f_8(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	30	0
$f_9(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^4 + \text{random}\,[0, 1)$	$[-1.28, 1.28]^n$	30	0
$f_{10}(\boldsymbol{x}) = \sum_{i=1}^{n}\lvert x_i\rvert + \prod_{i=1}^{n}\lvert x_i\rvert$	$[-10, 10]^n$	30	0
$f_{11}(\boldsymbol{x}) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	$[-100, 100]^n$	30	0
$f_{12}(\boldsymbol{x}) = \max\{\lvert x_i\rvert,\quad i = 1, 2, \ldots, n\}$	$[-100, 100]^n$	30	0

Exercise (cont.)

- Use the GA sample code "GA04-GA-basic-1.py" to find the optimal solutions of the following three functions:

Test functions	Feasible spaces	n	$f_{\min}$
$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} \left(-x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)\right)$	$[-500, 500]^n$	30	$-418.983n$
$f_2(\boldsymbol{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	$[-5.12, 5.12]^n$	30	0
$f_3(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) +20 + \exp(1)$	$[-32, 32]^n$	30	0

- Note that it is very hard to find the optimal solutions when n = 30. Hence, when testing your program, you can check whether it can find the optimal solution when n = 1.
- Hint
  - 1. (編碼) 改成30維度
  - 2. (編碼) 改初始化群組的範圍
  - 3. (解碼) 改適應度函數成f_1, f_2, f_3
  - 4. (輸出) 不要輸出x，只要看y = f(x)