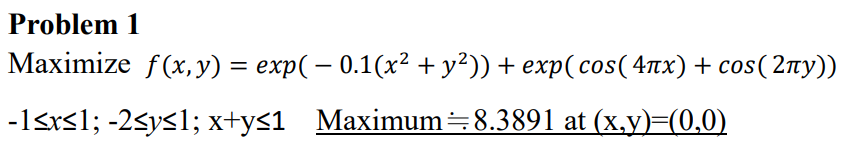
**計算智慧於工程上的應用**

**HW1(GAHomework)**

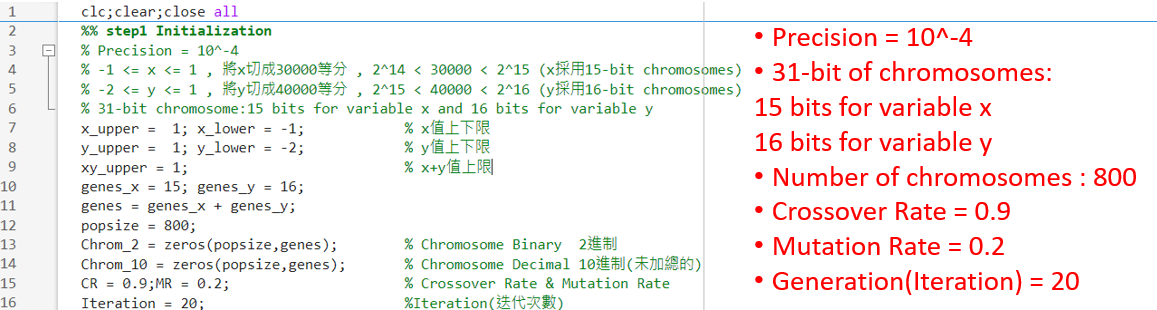
**M11205314**

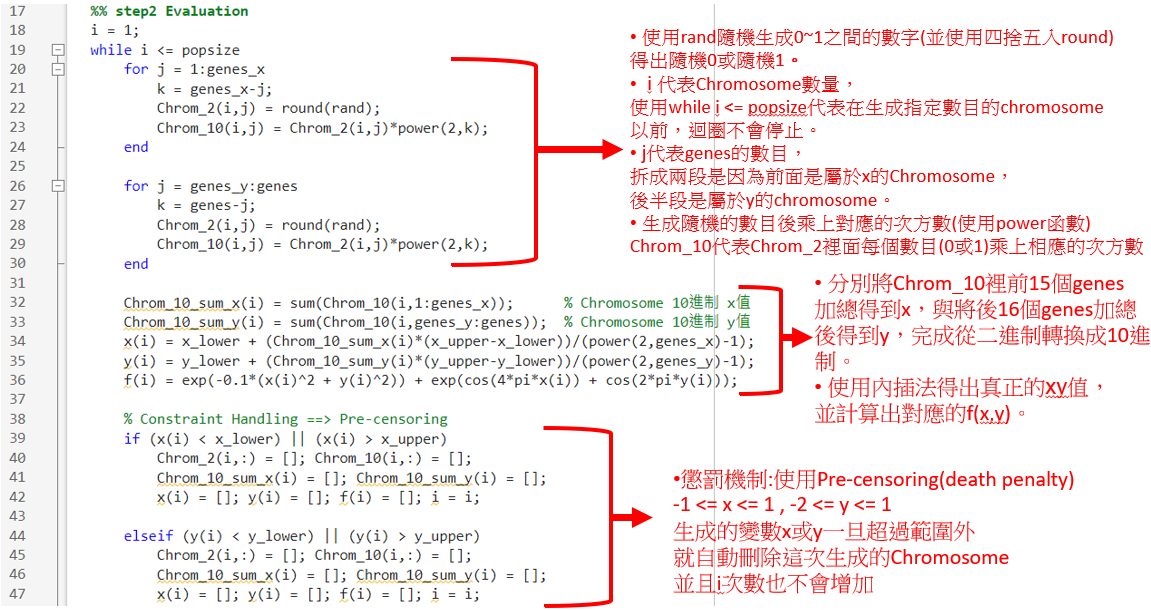
**張原嘉**

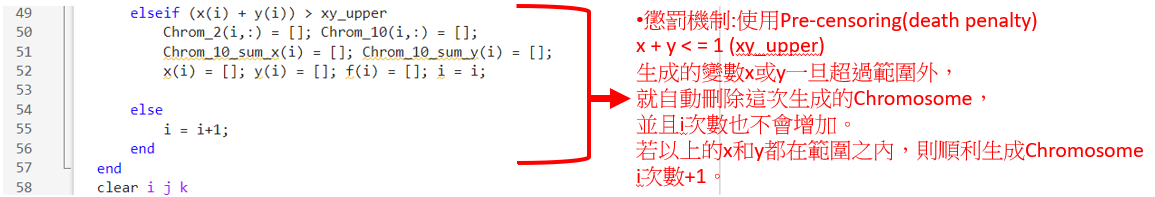
Programming Language : Matlab

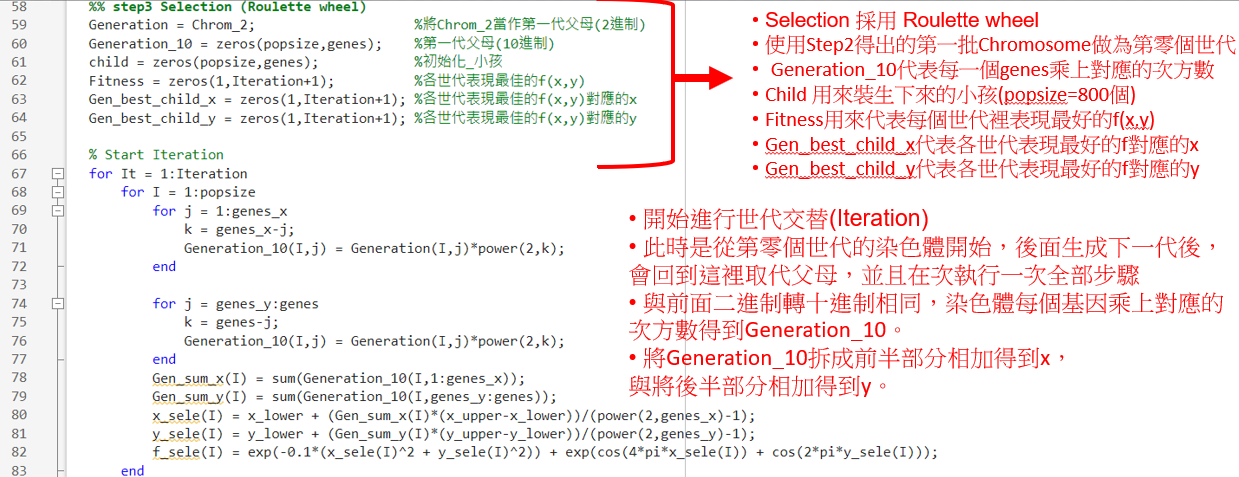


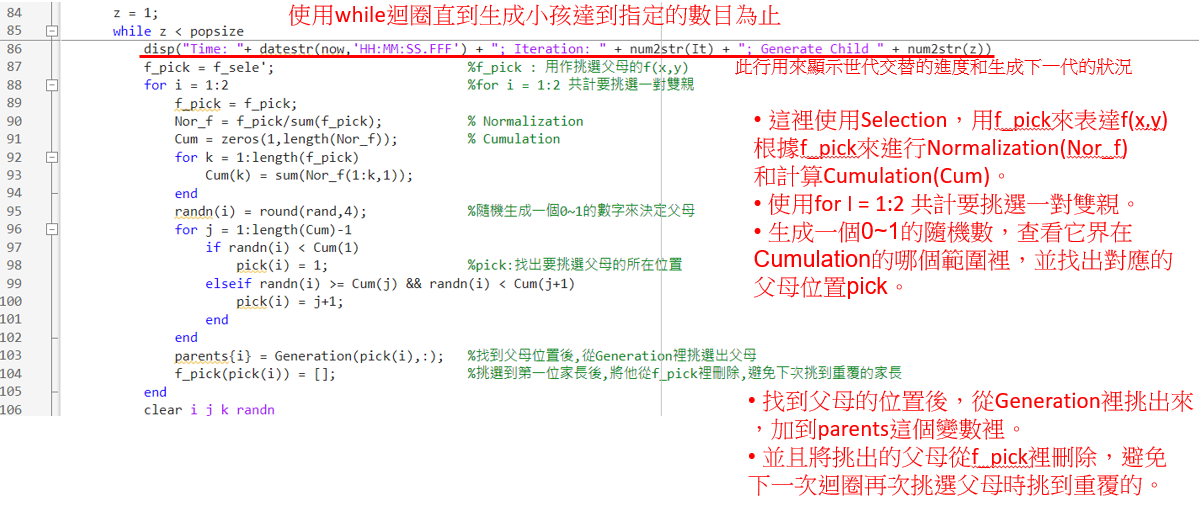
Step1 : Initialization

Step2 : Evaluation

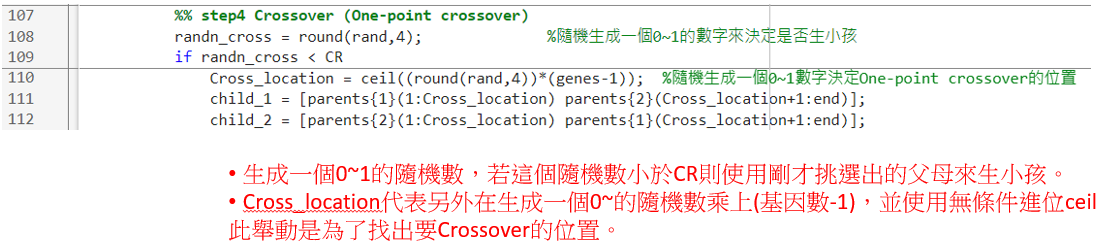


Step3 : Selection (Roulette wheel)

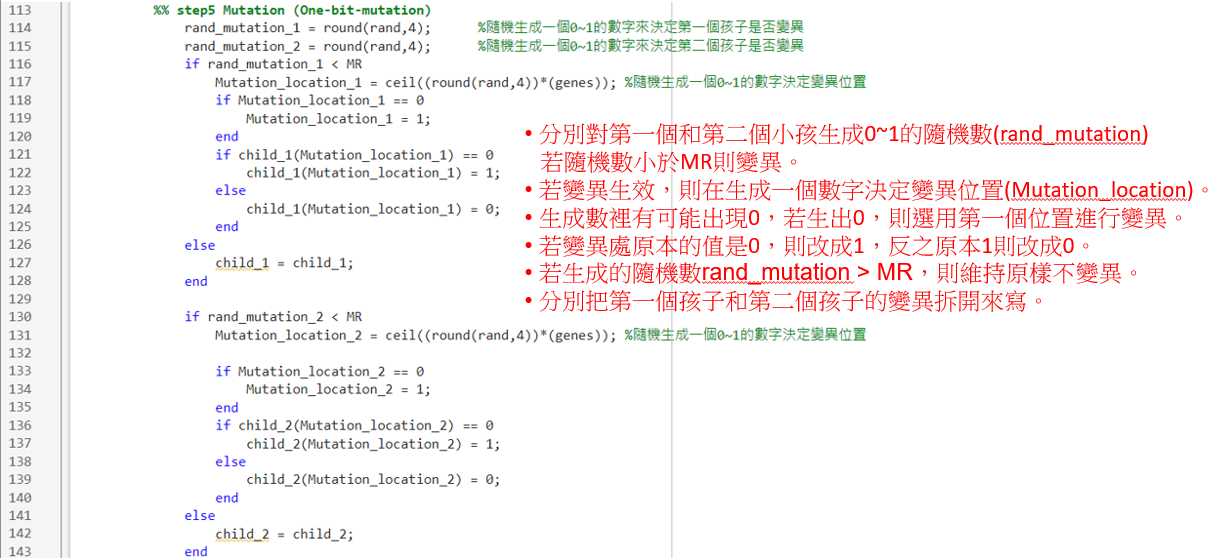




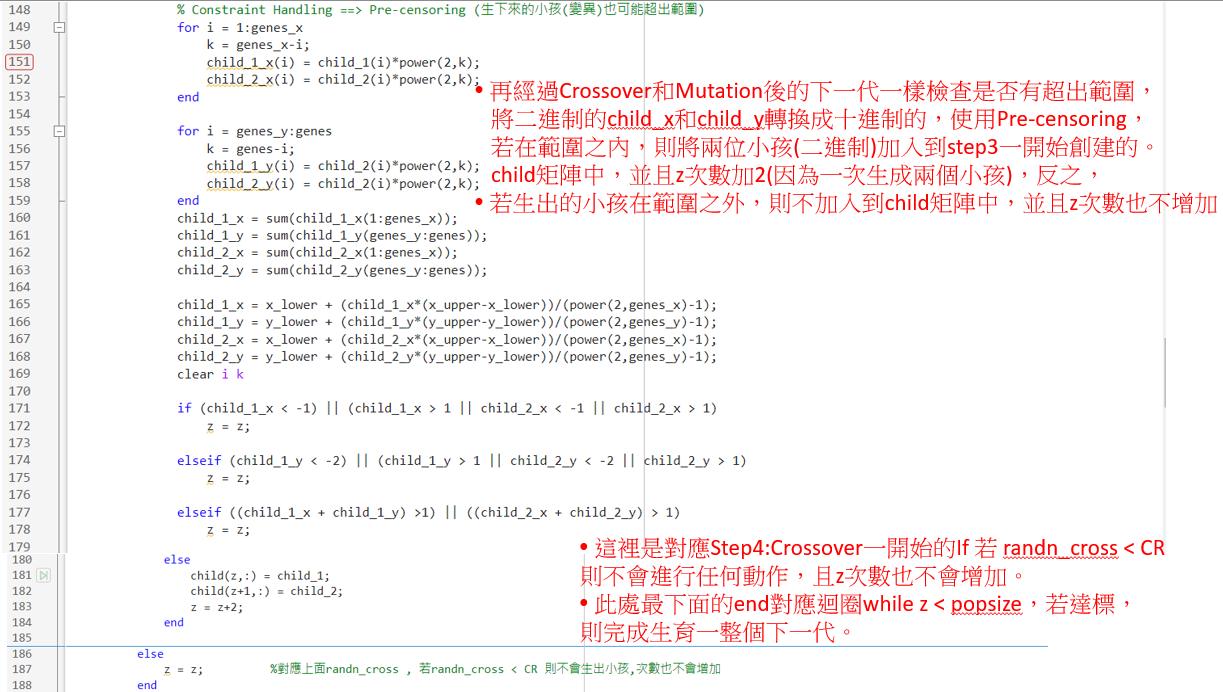
Step4 : Crossover (One-point crossover)



Step5 : Mutation (One-bit-mutation)

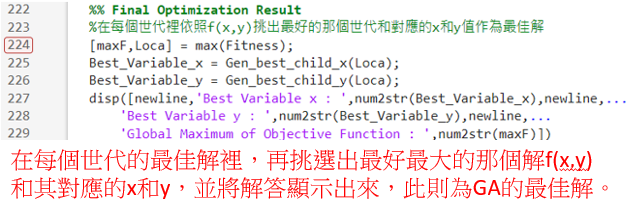


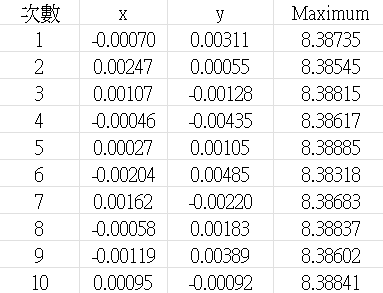
Step 6 : Constraint handling





Final Optimization Result

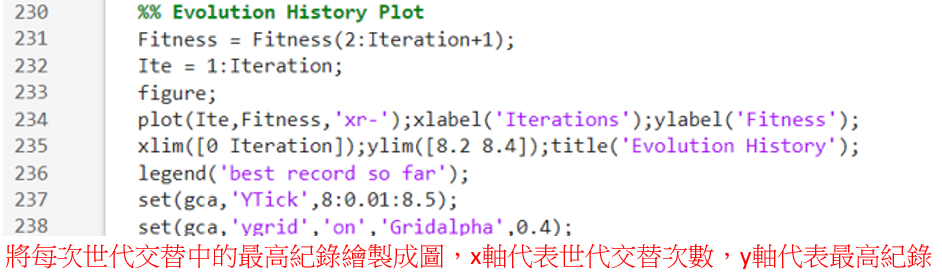


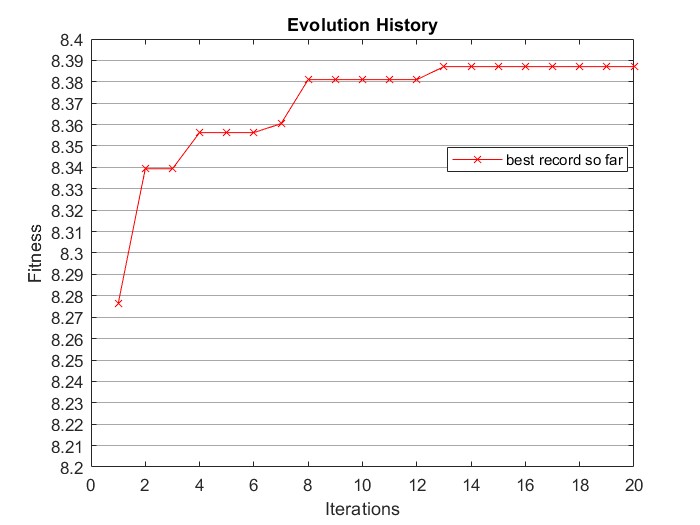


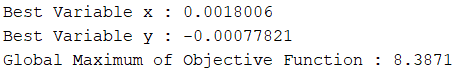


此為執行GA程式碼10次所得出的結果，與作業答案相比,精確度大概到小數點後第二位

Evolution History







此為其中一次的執行結果，Evolution History得出的圖形會是一條單調遞增的線段。

Matlab Code:

clc;clear;close all

%% step1 Initialization

% Precision = 10^-4

% -1 <= x <= 1 , 將x切成30000等分 , 2^14 < 30000 < 2^15 (x採用15-bit chromosomes)

% -2 <= y <= 1 , 將y切成40000等分 , 2^15 < 40000 < 2^16 (y採用16-bit chromosomes)

% 31-bit chromosome:15 bits for variable x and 16 bits for variable y

x\_upper = 1; x\_lower = -1; % x值上下限

y\_upper = 1; y\_lower = -2; % y值上下限

xy\_upper = 1; % x+y值上限

genes\_x = 15; genes\_y = 16;

genes = genes\_x + genes\_y;

popsize = 800;

Chrom\_2 = zeros(popsize,genes); % Chromosome Binary 2進制

Chrom\_10 = zeros(popsize,genes); % Chromosome Decimal 10進制(未加總的)

CR = 0.9;MR = 0.2; % Crossover Rate & Mutation Rate

Iteration = 20; %Iteration(迭代次數)

%% step2 Evaluation

i = 1;

while i <= popsize

for j = 1:genes\_x

k = genes\_x-j;

Chrom\_2(i,j) = round(rand);

Chrom\_10(i,j) = Chrom\_2(i,j)\*power(2,k);

end

for j = genes\_y:genes

k = genes-j;

Chrom\_2(i,j) = round(rand);

Chrom\_10(i,j) = Chrom\_2(i,j)\*power(2,k);

end

Chrom\_10\_sum\_x(i) = sum(Chrom\_10(i,1:genes\_x)); % Chromosome 10進制 x值

Chrom\_10\_sum\_y(i) = sum(Chrom\_10(i,genes\_y:genes)); % Chromosome 10進制 y值

x(i) = x\_lower + (Chrom\_10\_sum\_x(i)\*(x\_upper-x\_lower))/(power(2,genes\_x)-1);

y(i) = y\_lower + (Chrom\_10\_sum\_y(i)\*(y\_upper-y\_lower))/(power(2,genes\_y)-1);

f(i) = exp(-0.1\*(x(i)^2 + y(i)^2)) + exp(cos(4\*pi\*x(i)) + cos(2\*pi\*y(i)));

% Constraint Handling ==> Pre-censoring

if (x(i) < x\_lower) || (x(i) > x\_upper)

Chrom\_2(i,:) = []; Chrom\_10(i,:) = [];

Chrom\_10\_sum\_x(i) = []; Chrom\_10\_sum\_y(i) = [];

x(i) = []; y(i) = []; f(i) = []; i = i;

elseif (y(i) < y\_lower) || (y(i) > y\_upper)

Chrom\_2(i,:) = []; Chrom\_10(i,:) = [];

Chrom\_10\_sum\_x(i) = []; Chrom\_10\_sum\_y(i) = [];

x(i) = []; y(i) = []; f(i) = []; i = i;

elseif (x(i) + y(i)) > xy\_upper

Chrom\_2(i,:) = []; Chrom\_10(i,:) = [];

Chrom\_10\_sum\_x(i) = []; Chrom\_10\_sum\_y(i) = [];

x(i) = []; y(i) = []; f(i) = []; i = i;

else

i = i+1;

end

end

clear i j k

%% step3 Selection (Roulette wheel)

Generation = Chrom\_2; %將Chrom\_2當作第一代父母(2進制)

Generation\_10 = zeros(popsize,genes); %第一代父母(10進制)

child = zeros(popsize,genes); %初始化\_小孩

Fitness = zeros(1,Iteration+1); %各世代表現最佳的f(x,y)

Gen\_best\_child\_x = zeros(1,Iteration+1); %各世代表現最佳的f(x,y)對應的x

Gen\_best\_child\_y = zeros(1,Iteration+1); %各世代表現最佳的f(x,y)對應的y

% Start Iteration

for It = 1:Iteration

for I = 1:popsize

for j = 1:genes\_x

k = genes\_x-j;

Generation\_10(I,j) = Generation(I,j)\*power(2,k);

end

for j = genes\_y:genes

k = genes-j;

Generation\_10(I,j) = Generation(I,j)\*power(2,k);

end

Gen\_sum\_x(I) = sum(Generation\_10(I,1:genes\_x));

Gen\_sum\_y(I) = sum(Generation\_10(I,genes\_y:genes));

x\_sele(I) = x\_lower + (Gen\_sum\_x(I)\*(x\_upper-x\_lower))/(power(2,genes\_x)-1);

y\_sele(I) = y\_lower + (Gen\_sum\_y(I)\*(y\_upper-y\_lower))/(power(2,genes\_y)-1);

f\_sele(I) = exp(-0.1\*(x\_sele(I)^2 + y\_sele(I)^2)) + exp(cos(4\*pi\*x\_sele(I)) + cos(2\*pi\*y\_sele(I)));

end

z = 1;

while z < popsize

disp("Time: "+ datestr(now,'HH:MM:SS.FFF') + "; Iteration: " + num2str(It) + "; Generate Child " + num2str(z))

f\_pick = f\_sele'; %f\_pick : 用作挑選父母的f(x,y)

for i = 1:2 %for i = 1:2 共計要挑選一對雙親

f\_pick = f\_pick;

f\_pick = f\_pick - min(f\_pick);

Nor\_f = f\_pick/sum(f\_pick); % Normalization

Cum = zeros(1,length(Nor\_f)); % Cumulation

for k = 1:length(f\_pick)

Cum(k) = sum(Nor\_f(1:k,1));

end

randn(i) = round(rand,4); %隨機生成一個0~1的數字來決定父母

for j = 1:length(Cum)-1

if randn(i) < Cum(1)

pick(i) = 1; %pick:找出要挑選父母的所在位置

elseif randn(i) >= Cum(j) && randn(i) < Cum(j+1)

pick(i) = j+1;

end

end

parents{i} = Generation(pick(i),:); %找到父母位置後,從Generation裡挑選出父母

f\_pick(pick(i)) = []; %挑選到第一位家長後,將他從f\_pick裡刪除,避免下次挑到重覆的家長

end

clear i j k randn

%% step4 Crossover (One-point crossover)

randn\_cross = round(rand,4); %隨機生成一個0~1的數字來決定是否生小孩

if randn\_cross < CR

Cross\_location = ceil((round(rand,4))\*(genes-1)); %隨機生成一個0~1數字決定One-point crossover的位置

child\_1 = [parents{1}(1:Cross\_location) parents{2}(Cross\_location+1:end)];

child\_2 = [parents{2}(1:Cross\_location) parents{1}(Cross\_location+1:end)];

%% step5 Mutation (One-bit-mutation)

rand\_mutation\_1 = round(rand,4); %隨機生成一個0~1的數字來決定第一個孩子是否變異

rand\_mutation\_2 = round(rand,4); %隨機生成一個0~1的數字來決定第二個孩子是否變異

if rand\_mutation\_1 < MR

Mutation\_location\_1 = ceil((round(rand,4))\*(genes)); %隨機生成一個0~1的數字決定變異位置

if Mutation\_location\_1 == 0

Mutation\_location\_1 = 1;

end

if child\_1(Mutation\_location\_1) == 0

child\_1(Mutation\_location\_1) = 1;

else

child\_1(Mutation\_location\_1) = 0;

end

else

child\_1 = child\_1;

end

if rand\_mutation\_2 < MR

Mutation\_location\_2 = ceil((round(rand,4))\*(genes)); %隨機生成一個0~1的數字決定變異位置

if Mutation\_location\_2 == 0

Mutation\_location\_2 = 1;

end

if child\_2(Mutation\_location\_2) == 0

child\_2(Mutation\_location\_2) = 1;

else

child\_2(Mutation\_location\_2) = 0;

end

else

child\_2 = child\_2;

end

% Constraint Handling ==> Pre-censoring (生下來的小孩(變異)也可能超出範圍)

for i = 1:genes\_x

k = genes\_x-i;

child\_1\_x(i) = child\_1(i)\*power(2,k);

child\_2\_x(i) = child\_2(i)\*power(2,k);

end

for i = genes\_y:genes

k = genes-i;

child\_1\_y(i) = child\_2(i)\*power(2,k);

child\_2\_y(i) = child\_2(i)\*power(2,k);

end

child\_1\_x = sum(child\_1\_x(1:genes\_x));

child\_1\_y = sum(child\_1\_y(genes\_y:genes));

child\_2\_x = sum(child\_2\_x(1:genes\_x));

child\_2\_y = sum(child\_2\_y(genes\_y:genes));

child\_1\_x = x\_lower + (child\_1\_x\*(x\_upper-x\_lower))/(power(2,genes\_x)-1);

child\_1\_y = y\_lower + (child\_1\_y\*(y\_upper-y\_lower))/(power(2,genes\_y)-1);

child\_2\_x = x\_lower + (child\_2\_x\*(x\_upper-x\_lower))/(power(2,genes\_x)-1);

child\_2\_y = y\_lower + (child\_2\_y\*(y\_upper-y\_lower))/(power(2,genes\_y)-1);

clear i k

if (child\_1\_x < x\_lower) || (child\_1\_x > x\_upper || child\_2\_x < x\_lower || child\_2\_x > x\_upper)

z = z;

elseif (child\_1\_y < y\_lower) || (child\_1\_y > y\_upper || child\_2\_y < y\_lower || child\_2\_y > y\_upper)

z = z;

elseif ((child\_1\_x + child\_1\_y) > xy\_upper) || ((child\_2\_x + child\_2\_y) > xy\_upper)

z = z;

else

child(z,:) = child\_1;

child(z+1,:) = child\_2;

z = z+2;

end

else

z = z; %對應上面randn\_cross , 若randn\_cross < CR 則不會生出小孩,次數也不會增加

end

end

Gen\_child\_10 = zeros(popsize,genes);

for i = 1:popsize

for j = 1:genes\_x

k = genes\_x-j;

Gen\_child\_10(i,j) = child(i,j)\*power(2,k);

end

for j = genes\_y:genes

k = genes-j;

Gen\_child\_10(i,j) = child(i,j)\*power(2,k);

end

Gen\_child\_sum\_x(i) = sum(Gen\_child\_10(i,1:genes\_x));

Gen\_child\_sum\_y(i) = sum(Gen\_child\_10(i,genes\_y:genes));

x\_child(i) = x\_lower + (Gen\_child\_sum\_x(i)\*(x\_upper-x\_lower))/(power(2,genes\_x)-1);

y\_child(i) = y\_lower + (Gen\_child\_sum\_y(i)\*(y\_upper-y\_lower))/(power(2,genes\_y)-1);

f\_child(i) = exp(-0.1\*(x\_child(i)^2 + y\_child(i)^2)) + exp(cos(4\*pi\*x\_child(i)) + cos(2\*pi\*y\_child(i)));

end

clear i j k

%將每個世代裡最佳的f(x,y)和對應的最佳x和y記錄下來

[max\_f\_child maxloca] = max(f\_child);

if max\_f\_child >= Fitness(It)

Fitness(It+1) = max\_f\_child;

Gen\_best\_child\_x(It+1) = x\_child(maxloca);

Gen\_best\_child\_y(It+1) = y\_child(maxloca);

else

Fitness(It+1) = Fitness(It);

Gen\_best\_child\_x(It+1) = Gen\_best\_child\_x(It);

Gen\_best\_child\_y(It+1) = Gen\_best\_child\_y(It);

end

Generation = child;

end

clear I It z

%% Final Optimization Result

%在每個世代裡依照f(x,y)挑出最好的那個世代和對應的x和y值作為最佳解

[maxF,Loca] = max(Fitness);

Best\_Variable\_x = Gen\_best\_child\_x(Loca);

Best\_Variable\_y = Gen\_best\_child\_y(Loca);

disp([newline,'Best Variable x : ',num2str(Best\_Variable\_x),newline,...

'Best Variable y : ',num2str(Best\_Variable\_y),newline,...

'Global Maximum of Objective Function : ',num2str(maxF)])

%% Evolution History Plot

Fitness = Fitness(2:Iteration+1);

Ite = 1:Iteration;

figure;

plot(Ite,Fitness,'xr-');xlabel('Iterations');ylabel('Fitness');

xlim([0 Iteration]);ylim([8.2 8.4]);title('Evolution History');

legend('best record so far');

set(gca,'YTick',8:0.01:8.5);

set(gca,'ygrid','on','Gridalpha',0.4);