NTUST IM

M10901021 陳謙德

1. Optimization Problem

Objective function:

Maximize: f(x,y) = -x sin(4x) - 1.1y sin(2y) + 1

Constraints:

8≦x≦10

10≦y≦13

x+y≦22

2. Given the precision being 10-4, **16-bit chromosomes: 8 bits for x and 8 bits for y**

3. **16 of chromosomes and 400 generations**

4. **‘Rank Based Wheel Selection’** **of selection**, **one-point crossover for 2 variables x and y**, and **one-bit mutation for 2 variables x and y**

5. Constraint handling by **pre-censoring**

6. **Crossover rate = 0.9; mutation rate = 0.6**

7.

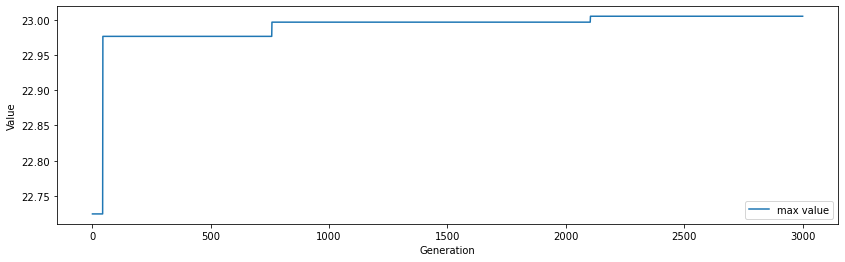
(1)

maximum solution: 23.004894161321587 position: 9.04313725490196 11.8



(2)

The evolution history chart with the X-axis showing generations and the Y-axis showing the record of the best objective function up to the current generation.



**Num: 22.996**

**position : (9.027 , 11.8)**

**Num: 23.004**

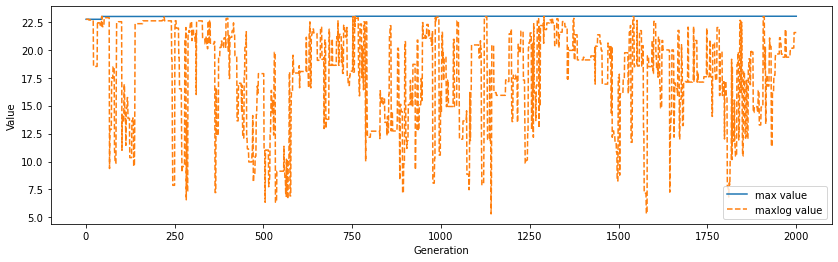
**position : (9.043 , 11.8)**

**Num: 22.976**

**position : (9.043 , 11.835)**

**Num: 22.724**

**position : (9.035 , 11.905)**



Blue line: the max value of whole iteration

Yellow -- : the max value of each iteration

8. Program code

# Define important variables and arrays

D = 2 #2 dimensions

it = 1 #initial iteration is 1

maxit = 3000 #max iteration

population\_size = 16 #population of chromosomes

nbit = 8 #each of variable have 8 bits

mr = 0.2 #mutation rate

cr = 0.9 #crossover rate

maxlog = [] #recording the max value of each iteration

maxmax = [] #recording the max value of all result

num1cool = [] #recording the parent

num2cool = [] #recording the parent

chromosome = D \* nbit #total bits

max\_solution = 0 #give the initial max solution

pos\_solution = [0,0] #give the initial position solution

# Define important variables and arrays\_end

#initialization

population = np.ones((population\_size,chromosome)) #the position of chromosomes

for i in range(population\_size):

for j in range(chromosome):

population[i,j]=np.random.randint(0,2)

x,y=bin\_to\_dec(population[i])

while x+y>22:

for j in range(chromosome):

**Constraints handling**

population[i,j]=np.random.randint(0,2)

x,y=bin\_to\_dec(population[i])

#initialization\_end

#fitness evaluation

def bin\_to\_dec(X):

x=0

y=0

bin\_to\_dec(X): binary to decimal function

for i in range(nbit):

x = x + X[i]\*2\*\*i

y = y + X[i+nbit]\*2\*\*i

x = 8+ x \* 2 / (2\*\*nbit-1)

y = 10+ y\* 3 / (2\*\*nbit-1)

return x,y

def fun(X):

xlist=[]

ylist=[]

anslist=[]

for i in range(population\_size):

Objective function

x,y=bin\_to\_dec(X[i])

ans = -x\*np.sin(4\*x) - 1.1\*y\*np.sin(2\*y)+1

xlist.append(x)

ylist.append(y)

anslist.append(ans)

return np.array(xlist),np.array(ylist),np.array(anslist)

decx,decy,value = fun(population)

#fitness evaluation\_end

#selection

def select(X):

arr = np.argsort(X)

percentage = []

cumulative = []

Rank Based Wheel Selection

for i in range(population\_size):

percentage.append(1/((population\_size+1)\*population\_size/2)\*(i+1))

valuepercentage = np.zeros(population\_size)

j=0

for i in arr:

valuepercentage[i] =percentage[j]

j+=1

cumulative = np.cumsum(valuepercentage,dtype = float)

selection = []

x=np.random.uniform(0,1)

k=0

while True:

if x<cumulative[k]:

break

k=k+1

if k==population\_size:

k=0

break

selection.append(X[k])

value\_new = np.delete(X,k)

arr\_new = np.argsort(value\_new)

percentage\_new = []

cumulative\_new = []

for i in range(population\_size-1):

percentage\_new.append(1/((population\_size+1-1)\*(population\_size-1)/2)\*(i+1))

valuepercentage\_new = np.zeros(population\_size-1)

j=0

for i in arr\_new:

valuepercentage\_new[i]=percentage\_new[j]

j+=1

cumulative\_new = np.cumsum(valuepercentage\_new,dtype = float)

x=np.random.uniform(0,1)

k=0

while True:

if x<cumulative\_new[k]:

break

k+=1

if k==population\_size-1:

k=0

break

selection.append(value\_new[k])

num1 = np.where( X==selection[0])

num2 = np.where( X==selection[1])

return selection,num1,num2

#selection\_end

#crossover

def crossover(X,Y):

docr = False

ax = population[X]

bx = population[Y]

if np.random.rand(1) <= cr:

a = population[X]

b = population[Y]

x1,y1 = bin\_to\_dec(a[0])

x2,y2 = bin\_to\_dec(b[0])

err=0

while (docr == False or x1+y1>22 or x2+y2>22):

ax=[]

ay=[]

bx=[]

by=[]

for i in range(nbit):

ax.append(a[0][i])

bx.append(b[0][i])

ay.append(a[0][i+nbit])

by.append(b[0][i+nbit])

t = math.floor(np.random.rand(1)\*(nbit))

temp = np.zeros(nbit)

temp[0:t],ax[0:t],bx[0:t]=ax[0:t],bx[0:t],ax[0:t]

t = math.floor(np.random.rand(1)\*(nbit))

temp[0:t],ay[0:t],by[0:t]=ay[0:t],by[0:t],ay[0:t]

for i in ay :

ax.append(i)

for i in by:

bx.append(i)

x1,y1 = bin\_to\_dec(ax)

x2,y2 = bin\_to\_dec(bx)

docr = True

err+=1

if(x1+y1>22 or x2+y2>22):

while(err==5 or x1+y1>22 or x2+y2>22):

ax=[]

bx=[]

**Constraints handling**

for i in range(population\_size):

ax.append(np.random.randint(0,2))

bx.append(np.random.randint(0,2))

x1,y1 = bin\_to\_dec(ax)

x2,y2 = bin\_to\_dec(bx)

err+=1

break

return ax,bx,docr

#crossover\_end

#mutation

def mutation(bool1,dad,mom):

dad = dad

mom = mom

x1 =100

y1 =100

if bool1 == True:

a=np.random.rand(1)

b=np.random.rand(1)

if a<= mr:

while(x1+y1>22):

x = np.floor(np.random.rand(1)\*(nbit\*D))

**Constraints handling**

dad[int(x[0])]= abs(dad[int(x[0])]-1)

x1,y1=bin\_to\_dec(dad)

bool1=False

bool1 = True

if b<= mr:

while( bool1==True or x1+y1>22):

x = np.floor(np.random.rand(1)\*(nbit\*D))

**Constraints handling**

mom[int(x[0])]= abs(mom[int(x[0])]-1)

x1,y1=bin\_to\_dec(mom)

bool1=False

return dad,mom

#mutation\_end

selection,num1\_arg,num2\_arg = select(value)

aco1,aco2,docr1 = crossover(num1\_arg,num2\_arg)

child1,child2 = mutation(docr1,aco1,aco2)

while (it<=maxit):

num1cool.append(num1\_arg)

Recording parents of each generations

num2cool.append(num2\_arg)

Replacing parents with children to population

population[num1\_arg]=child1

population[num2\_arg]=child2

fitness evaluation

decx,decy,value = fun(population)

Recording the max value of each of iterations to maxlog list

maxlog.append(np.max(value))

if np.max(value) >= max\_solution:

Update the best solution

max\_solution = np.max(value)

pos\_solution [0] = decx[np.where(value==np.max(value))]

pos\_solution[1] = decy[np.where(value==np.max(value))]

selection,num1\_arg,num2\_arg = select(value)

Selection

Crossover

Mutation

aco1,aco2,docr1 = crossover(num1\_arg,num2\_arg)

child1,child2 = mutation(docr1,aco1,aco2)

print("第",it,"個 : ", np.max(maxlog),"位置 :",pos\_solution[0][0],pos\_solution[1][0],"變動值 :",value[num1cool[it-1]]," ",value[num2cool[it-1]])

Update the best solution to maxmax list

maxmax.append(np.max(maxlog))

it+=1

# Finish GA algorithm

# Result

print("maximum solution: ",max\_solution,"position: ",pos\_solution[0][0],pos\_solution[1][0])

plt.xlabel("Generation")

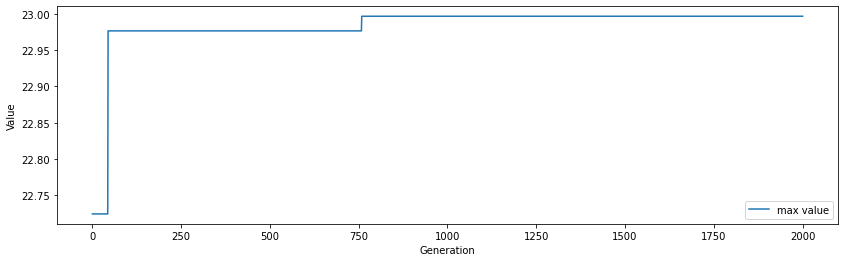
plt.ylabel("Value")

plt.plot(np.arange(1,len(maxmax)+1),maxmax,label='max value')

plt.legend(loc='lower right')

plt.show()

“””



‘’’

plt.xlabel("Generation")

plt.ylabel("Value")

plt.rcParams["figure.figsize"] = (14, 4)

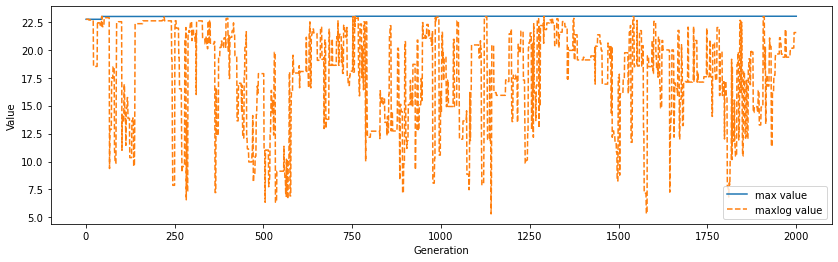
plt.plot(np.arange(1,len(maxmax)+1),maxmax,label='max value')

plt.plot(np.arange(1,len(maxmax)+1),maxlog,'--',label='maxlog value')

plt.legend(loc='lower right')

plt.show()

‘’’



‘’’