Name: Chenxu Yang

UNI:cy2554

Course name: Evolutionary Computation & Design Automation (MECS E4510)

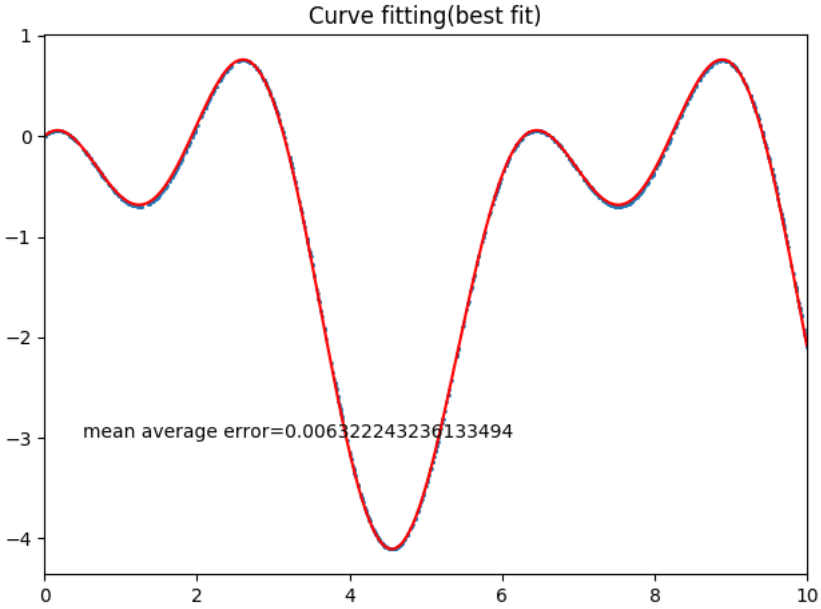
Instructor: Hod Lipson

Date Submitted:2019/10/19/4PM.

Grace hours used:0

Grace hours remaining:96+31(HW1)=127

Results summary table



F(x)= 2.491566422309853∗sin(x)∗sin(x − 2.712154266303183) + 1.758211775106354∗sin(x)

Methods

Representation:

Use a 256_long list to represent tree, which follows those rules:

1) Gene[i] is an operator only if gene[2*i] and gene[2*i+1] !=0,
2) Gene[sin] is a sin/cos only if gene[2*i] is an operator or x/coefficient, and gene[2*i+1]=0,
3) The element in the 8th level of the tree must be coefficient or x or 0,
4) If gene[int(i/2)]=coefficient or x, then gene[i] must be 0,
5) Use 0 to represent useless node.

Random search:

for every time, generate a 256_long list which represents the function tree, calculate the mean average error, keep the shortest distance, and draw it. *Poor performance*

Hill climber:

generate a 256_long list which represent the function tree, for every time, change one part of the function tree, for example, it can change the operator into another operator or sin/cos, it can also change sin/cos into operator or constant, and it can change a little bit of all coefficients, if the distance is shorter, keep that change, else change back.

Better performance than random search, but stuck at the mean average error of 0.36.

GP:

1) generate a population which includes 100 individuals, each individual is a random 256_long list which represent the function tree, the list is also called gene.
2) Get each individual's fitness(100/sum_distance), calculate the sum of total fitness and find the best individual (with smallest mean average error).
3) Generate a new population:
   (1) Cross: select two individuals, change the subtree with each other, the subtree is in the same lever,
   (2) Mutation: select an individual, use the method in Hill Climber, make a small change in the gene,
   (3) Select: according to every individual's fitness, use Roulette method to select individual,
   (4) Always keep the best individual in the previous population.

The outcome is a better than Hill climber, but cost more time to calculate

GP with diversity and deterministic crowding:

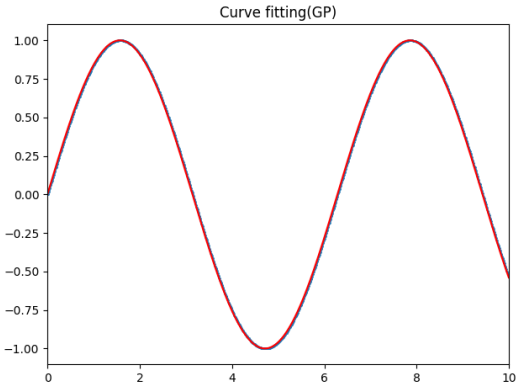
Almost same as GP, except two different ways in generating new population and cross:

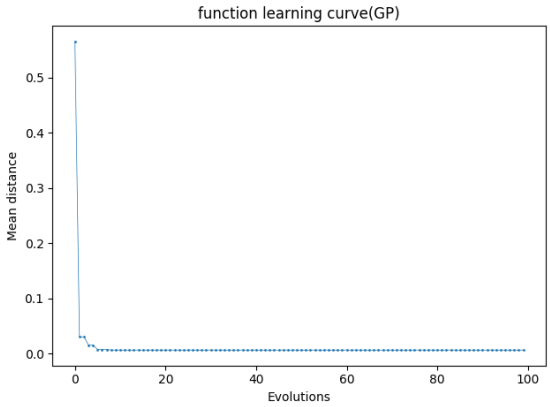
1) When generating a new population, the last person in the population will be replaced by an individual which generated by hill climber method with mean average error less than 0.7,
2) When a child is born, compare its fitness with the parents, keep the better individual.

The outcome is much better than the three methods above, nearly find the answer.

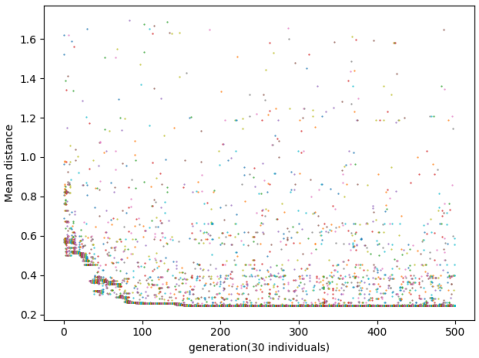
Method	evaluations	Mean average error
Rondom search	1000000	0.49
Hill climber	1000000	0.36
GP	1000000	0.17
GP with diversity and deterministic crowding	1000000	0.0063

Simple problem tested: y=sin(x)

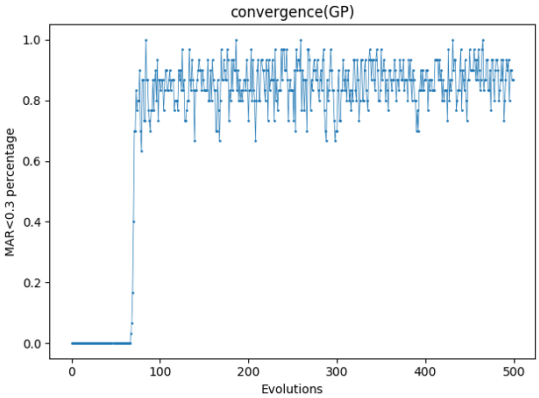




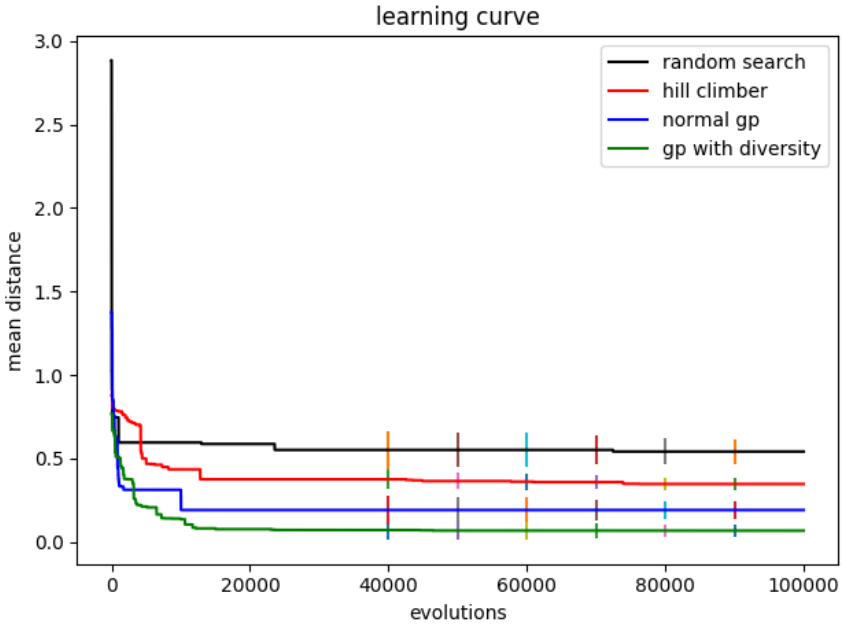
Dot plot for GP



Convergence plot for GP(MAR<0.3)



Learning curve of all methods



Code including

1) GP
2) GP with diversity
3) Hill climber

1)

```
import random
import sympy
import numpy as np
import matplotlib.pyplot as plt
from math import sin,cos
from numba import njit
GENERATION=5
POPULATION_SIZE=30
GENE_SIZE=256
CROSS_RATE=0.8
MUTATION_RATE=0.3
operator_dic={1:'+',2:'-',3:'*',4:'/',5:'x',6:'c',7:'sin',8:'cos'}
operator=['+','-','*','/']
trangle=['sin','cos']
x_or_const=['x','c']
a = np.loadtxt('data.txt')
POINT_POSITION = np.reshape(a, (1000, 2))
new_position=[]
for i in range(0,999,5):
    new_position.append(POINT_POSITION[i])
new_position = np.array(new_position)
def make_population():
    population=[]
    for _ in range(POPULATION_SIZE):
        individual=generate()
        error1 = 'zoo'
        error2 = 'nan'
        string = str(sympy.expand(to_string(individual, 1)))
        while error1 in string or error2 in string:
            individual = generate()
            string = str(sympy.expand(to_string(individual, 1)))
        population.append(individual)
    return population
def fitness(population,y):
    best_individual = population[0]
    best_fitness = get_fitness(best_individual)
    fitness_list = [0] * POPULATION_SIZE
    sumfitness = 0
```

```
    a = []
    for i, individual in enumerate(population):
        fitness_list[i] = get_fitness(individual)


        if fitness_list[i] > best_fitness:
            best_individual = individual
            best_fitness = fitness_list[i]
        sumfitness += fitness_list[i]
    return sumfitness, best_individual, fitness_list, best_fitness,a
'''def is_stright_line(gene):
    gene = str(sympy.expand(to_string(gene, 1)))
    x = 0.01
    y1=eval(gene)
    x=0.02
    y2 = eval(gene)
    x=0.03
    y3 = eval(gene)
    if y1==y2 and y1==y3:
        a=generate()
        return a
    else:
        return gene'''

def cross(mom,dad):
    child1=mom[:]
    child2=dad[:]
    index= random.randint(2,255)# 随机生成突变起始位置 #
    while child1[index]==0 or child2[index]==0:
        index = random.randint(2, 255)  # 随机生成突变起始位置 #
    child1,child2=change_subtree(index,child1,child2)
    error1 = 'zoo'
    error2 = 'nan'
    child1_gene = str(sympy.expand(to_string(child1, 1)))
    child2_gene = str(sympy.expand(to_string(child2, 1)))
    if (error1 in child1_gene or error2 in child1_gene) or (error1 in child2_gene or
error2 in child2_gene) :
        return mom
    else: return child1 if get_fitness(child1)>get_fitness(child2) else child2
def change_subtree(index,childa,childb):
    child1=childa[:]
    child2=childb[:]
    child1[index],child2[index]=child2[index],child1[index]
    i=2
```

```
    while 2*index<256:
        index*=2
        for j in range(i):
            child1[index+j],child2[index+j]=child2[index+j],child1[index+j]
        i*=2
    return child1,child2
def mutation(a):
    child = a[:]
    index = random.randint(1, 255)
    while child[index] == 0:
        index = random.randint(1, 255)
    if child[index] in operator:
        rate = random.uniform(0, 1)
        if rate < 0.1:
            child[index] = operator_dic[random.randint(1, 4)]
        elif rate>0.1 and rate<0.2:
            child[index]=operator_dic[random.randint(7, 8)]
            child[2 * index + 1] = 0
            index=2*index+1
            while 2*index+1<=256:
                index=2*index
                child[index]=0
                child[index+1]=0
    elif child[index] in trangle:
        rate = random.uniform(0, 1)
        if rate<0.4:
            child[index]=operator_dic[random.randint(7,8)]

        else:
            child[index]=operator_dic[random.randint(1, 4)]
            child[2*index+1]=str(random.uniform(-10,10))
    elif child[index] == 'x':
        if 2*index<256:
            rate = random.uniform(0, 1)
            if rate<0.5:
                child[index]=child[index]=operator_dic[random.randint(7,8)]
                child[index*2]='x'
        else:
            child[index] = str(random.uniform(-10, 10))
    else:
        rate = random.uniform(0, 1)
        if rate < 0.4:
            child[index] = str(float(child[index]) + 0.1)
        elif rate < 0.8 and rate>=0.4:
```

```
            child[index] = str(float(child[index]) - 0.1)
        else:
            rate = random.uniform(0, 1)
            if 2*index<256 and rate<0.5:
                child[index]=operator_dic[random.randint(7,8)]
                child[2*index]='x'
            else:
                child[index] = 'x'
    error1 = 'zoo'
    error2 = 'nan'
    this_gene = str(sympy.expand(to_string(child, 1)))
    if error1 in this_gene or error2 in this_gene:
        return a
    else: return child
def tox(a,index):
    a[index]='x'
    while 2*index<=255:
        index=index*2
        a[index]=0
        a[index+1]=0
    return a
def select(population,sumfitness):
    flag=random.uniform(0,sumfitness)
    for individual in population:
        flag-=get_fitness(individual)
        if flag<=0:
            return individual
def born(population,sumfitness):
    dad=select(population,sumfitness)
    while dad is None:
        dad = select(population, sumfitness)
    mom = select(population, sumfitness)
    while mom is None:
        mom=select(population,sumfitness)
    rate = random.uniform(0, 1)
    if rate < CROSS_RATE:
        child = cross(dad, mom)
    else:
        child = dad
    rate = random.uniform(0, 1)
    if rate < MUTATION_RATE:
        child=mutation(child)
    return child
def generate():
```

```
    gene=[0]*256
    gene[1]=operator_dic[random.randint(1,4)]
    for i in range(1,7):
        for j in range(2**i,2**(i+1)):
            if gene[int(j/2)] in operator:
                gene[j]=operator_dic[random.randint(1,8)]
            if gene[int(j/2)] in trangle:
                if j%2==0:
                    gene[j]=operator_dic[random.randint(5,8)]
    for i in range(2**7,2**8):
        if gene[int(i/2)] in trangle:
            if i%2==0:
                gene[i] = operator_dic[random.randint(5, 6)]
        if gene[int(i/2)] in operator:
            gene[i]=operator_dic[random.randint(5,6)]
    error1 = 'zoo'
    error2 = 'nan'
    string = str(sympy.expand(to_string(gene, 1)))
    while error1 in string or error2 in string:
        gene = generate()
        string = str(sympy.expand(to_string(gene, 1)))
    set_c(gene)
    return gene
def generation(population,y):
    sumfitness,best_individual,fitness_list,best_fitness,a=fitness(population,y)
    s = []
    while len(s) < len(a):
        s.append(y)
    plt.scatter(s, a, marker='.', s=1)
    plt.xlabel('generation(30 individuals)')
    plt.ylabel('Mean distance')
    new_population=[]
    new_population.append(best_individual[:])
    while len(new_population) < POPULATION_SIZE:
        #child=born(population,sumfitness)
        #child=is_stright_line(child)
        new_population.append(born(population,sumfitness))
    return new_population

def to_string(gene,x):
    result=""
    if 2*x>=256 or gene[2*x]==0:
        result+=gene[x]
    elif gene[x] in trangle:
```

```
        result=result+gene[x]+'('+to_string(gene,2*x)+')'
    else :
        result=result+'('+to_string(gene,2*x)+gene[x]+to_string(gene,2*x+1)+')'
    return result
def set_c(a):
    for index,item in enumerate(a):
        if item=='c' :
            a[index]=str(random.uniform(-10,10))
    return a
#@njit
def get_fitness(gene):
    sum=0
    gene=str(sympy.expand(to_string(gene,1)))
    for i in range(0,len(POINT_POSITION),50):
        x=POINT_POSITION[i][0]
        sum+=abs(POINT_POSITION[i][1]-eval(gene))
    return 100/sum
def draw(shortest_distance):
    generation=[i for i in range(GENERATION)]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(generation,shortest_distance, lw=0.5, marker='o', markersize=1, mfc='w')
    ax1.set_title('function learning curve(GP)')
    ax1.set_xlabel('Evolutions')
    ax1.set_ylabel('Mean distance')
    plt.show()
def draw_graph(gene):
    point_x=[]
    point_y=[]
    for i in range(len(POINT_POSITION)):
        point_x.append(POINT_POSITION[i][0])
        point_y.append(POINT_POSITION[i][1])
    npgene=''
    for index in range(len(gene)):
        if (gene[index]=='s' and gene[index+1]=='i') or gene[index]=='c':
            npgene+='np.'
        npgene+=gene[index]
    print(npgene)
    x = np.arange(0, 10, 0.005)
    npgene+='+(x-x)'
    print(npgene)
    y = eval(npgene)
    plt.xlim(0, 10)
    plt.plot(x, y,c='r')
```

```
    plt.scatter(point_x,point_y,marker='o',s=1)
    plt.title('Curve fitting(GP)')
    plt.show()
if __name__=="__main__":
    population = make_population()
    i = GENERATION
    shortest_distance = []
    while i > 0:
        sumfitness, best_individual, fitness_list, best_fitness,a =
fitness(population,GENERATION-1)
        print(str(sympy.expand(to_string(best_individual, 1))))
        print("mean_distance:%f,  generation:%d" % (100/best_fitness/20,GENERATION-i))
        shortest_distance.append(100/best_fitness/20)
        population = generation(population, GENERATION - i)
        i -= 1
    best_string = str(sympy.expand(to_string(best_individual, 1)))
    draw_graph(best_string)
    #np.savetxt('best_individual.txt', best_individual)
    #np.savetxt('short_distance.txt', shortest_distance)
    draw(shortest_distance)
    #np.savetxt('ea data1.txt', shortest_distance)
```

2)GP with diversity

Change funtions 'cross' and 'generation' into:

```
def cross(mom,dad):
    child1=mom[:]
    child2=dad[:]
    index= random.randint(2,255)# 随机生成突变起始位置 #
    while child1[index]==0 or child2[index]==0:
        index = random.randint(2, 255)  # 随机生成突变起始位置 #
    child1,child2=change_subtree(index,child1,child2)
    error1 = 'zoo'
    error2 = 'nan'
    child1_gene = str(sympy.expand(to_string(child1, 1)))
    child2_gene = str(sympy.expand(to_string(child2, 1)))
    if (error1 in child1_gene or error2 in child1_gene) or (error1 in child2_gene or
error2 in child2_gene) :
        return mom
    else:
        list=[mom,dad,child1,child2]
```

```
fitness_list1=[get_fitness(mom),get_fitness(dad),get_fitness(child1),get_fitness(child
2)]
        dic=dict(zip(fitness_list1,list))
        return dic[max(fitness_list1)]
def generation(population,y):
    sumfitness,best_individual,fitness_list,best_fitness,a=fitness(population,y)
    s = []
    while len(s) < len(a):
        s.append(y)
    plt.scatter(s, a, marker='.', s=1)
    plt.xlabel('generation(30 individuals)')
    plt.ylabel('Mean distance')
    new_population=[]
    new_population.append(best_individual[:])
    while len(new_population) < POPULATION_SIZE-1:
        #child=born(population,sumfitness)
        #child=is_stright_line(child)
        new_population.append(born(population,sumfitness))
    new_individual=generate()
    while get_fitness(new_individual)>0.7:
        new_individual=generate()
    new_population.append(new_individual)
    return new_population
```

3)Hill climber:

```
import random
import sympy
import numpy as np
import matplotlib.pyplot as plt
from math import sin,cos
from numba import njit
GENERATION=10
operator_dic={1:'+',2:'-',3:'*',4:'/',5:'x',6:'c',7:'sin',8:'cos'}
operator=['+','-','*','/']
trangle=['sin','cos']
x_or_const=['x','c']
a = np.loadtxt('data.txt')
POINT_POSITION = np.reshape(a, (1000, 2))
new_position=[]
for i in range(0,999,5):
    new_position.append(POINT_POSITION[i])
new_position = np.array(new_position)
```

```
def generate():
    gene=[0]*256
    gene[1]=operator_dic[random.randint(1,4)]
    for i in range(1,7):
        for j in range(2**i,2**(i+1)):
            if gene[int(j/2)] in operator:
                gene[j]=operator_dic[random.randint(1,8)]
            if gene[int(j/2)] in trangle:
                if j%2==0:
                    gene[j]=operator_dic[random.randint(5,8)]
    for i in range(2**7,2**8):
        if gene[int(i / 2)] in trangle:
            if i % 2 == 0:
                gene[i] = operator_dic[random.randint(5, 6)]
        if gene[int(i / 2)] in operator:
            gene[i] = operator_dic[random.randint(5, 6)]
    return set_c(gene)
def climber(a):
    origin=a[:]
    origin_gene=str(sympy.expand(to_string(a, 1)))
    origin_distance=get_fitness(origin_gene)
    index=random.randint(1,255)
    while a[index]==0:
        index=random.randint(1,255)
    if a[index] in operator:
        rate = random.uniform(0, 1)
        if rate < 0.6:
            a[index] = operator_dic[random.randint(1, 4)]
        else:
            a[index]=operator_dic[random.randint(7, 8)]
            a[2 * index + 1] = 0
            index=2*index+1
            while 2*index+1<=256:
                index=2*index
                a[index]=0
                a[index+1]=0
    elif a[index] in trangle:
        rate=random.uniform(0,1)
        if rate<0.4:
            a[index]=operator_dic[random.randint(7,8)]
        else:
            a[index]=operator_dic[random.randint(1, 4)]
            rate=random.uniform(0,1)
            if rate<0.5:
```

```
                a[2*index+1]=str(random.uniform(-10,10))
            else: a[2*index+1]='x'
    elif a[index] == 'x':
        if 2*index<256:
            rate = random.uniform(0, 1)
            if rate<0.5:
                a[index]=a[index]=operator_dic[random.randint(7,8)]
                a[index*2]='x'
        else:
            a[index] = str(random.uniform(-10, 10))
    else:
        rate = random.uniform(0, 1)
        if rate < 0.4:
            a[index] = str(float(a[index]) + 0.1)
        elif rate < 0.8 and rate>=0.4:
            a[index] = str(float(a[index]) - 0.1)
        else:
            rate = random.uniform(0, 1)
            if 2*index<256 and rate<0.5:
                a[index]=operator_dic[random.randint(7,8)]
                a[2*index]='x'
            else:
                a[index] = 'x'
    error1 = 'zoo'
    error2 = 'nan'
    this_gene = str(sympy.expand(to_string(a, 1)))
    while error1 in this_gene or error2 in this_gene:
        a = climber(a)
        this_gene = str(sympy.expand(to_string(a, 1)))
    this_distance = get_fitness(this_gene)
    if this_distance<origin_distance:
        return a
    else:
        return origin

def tox(a,index):
    a[index]='x'
    while 2*index<=255:
        index=index*2
        a[index]=0
        a[index+1]=0
    return a
def to_string(gene,x):
    result=""
```

```
    if 2*x>=256 or gene[2*x]==0:
        result+=gene[x]
    elif gene[x] in trangle:
        result=result+gene[x]+'('+to_string(gene,2*x)+')'
    else :
        result=result+'('+to_string(gene,2*x)+gene[x]+to_string(gene,2*x+1)+')'
    return result
def set_c(a):
    for index,item in enumerate(a):
        if item=='c' :
            a[index]=str(random.uniform(-10,10))
    return a
#@njit
def get_fitness(gene):
    sum=0
    for i in range(0,len(POINT_POSITION),20):
        x=POINT_POSITION[i][0]
        sum+=abs(POINT_POSITION[i][1]-eval(gene))
    return sum
def draw(shortest_distance):
    generation=[i for i in range(GENERATION)]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(generation,shortest_distance, lw=0.5, marker='o', markersize=1, mfc='w')
    ax1.set_title('function learning curve(random search)')
    ax1.set_xlabel('Evolutions')
    ax1.set_ylabel('Mean distance')
    plt.show()
def draw_graph(gene):
    point_x=[]
    point_y=[]
    for i in range(len(POINT_POSITION)):
        point_x.append(POINT_POSITION[i][0])
        point_y.append(POINT_POSITION[i][1])
    npgene=''
    for index in range(len(gene)):
        if (gene[index]=='s' and gene[index+1]=='i') or gene[index]=='c':
            npgene+='np.'
        npgene+=gene[index]
    print(npgene)
    x = np.arange(0, 10, 0.005)
    npgene+='+(x-x)'
    print(npgene)
    y = eval(npgene)
```

```
    plt.xlim(0, 10)
    plt.plot(x, y,c='r')
    plt.scatter(point_x,point_y,marker='o',s=1)
    plt.title('Curve fitting(Hillclimber)')
    plt.show()
if __name__=="__main__":
    i=0
    shortest_distance=[]
    shortest=1000000
    a = generate()
    error1 = 'zoo'
    error2 = 'nan'
    gene = str(sympy.expand(to_string(a, 1)))
    while error1 in gene or error2 in gene:
        a = generate()
    while(i<GENERATION):
        a=climber(a)
        best_gene = str(sympy.expand(to_string(a, 1)))
        shortest=get_fitness(best_gene)/50
        shortest_distance.append(shortest)
        i+=1
        print('generation:%d,distance:%f'%(i,shortest))

    print(best_gene)
    draw(shortest_distance)
    draw_graph(best_gene)
    #np.savetxt('climber data.txt', shortest_distance)
```