

Sep 08, 14 12:24

csc710sbse:hw1:VivekNair:vnair2

Page 1/4

```

from __future__ import division
import sys
import random
import math
5 sys.dont_write_bytecode = True

def say(x):
    sys.stdout.write(str(x)); sys.stdout.flush()

10 def probFunction(old,new,t):
    # print "probFunction : old : %f new : %f t: %f return : %f exp: %f" %(old,new
    , t,-1*(old-new)/t,math.exp(1 *(old-new)/t))
    return math.exp(1 *(old-new)/t)

def neighbour(s):
15 if(s==99):
    return s-1
    elif(s=-99):
    return s+1
    else:
20 if(random.randint(0,1) == 1):
    return s+1
    else:
    return s-1

25 class Model:
    def schaffer(self,independentVariable):
        global minVal,maxVal
        f1 = independentVariable**2
        f2 = (independentVariable -2)**2
30 return (f1+f2)

class BaseLine:
    def __init__(self):
        self.minVal=10000000
        self.maxVal=0
35
    def returnMin(self,num):
        if(num<self.minVal):
            return num
40 else:
            return self.minVal

    def returnMax(self,num):
        if(num>self.maxVal):
            return num
45 else:
            return self.maxVal

    def findBaseLine(self):
50 low = -100
    high = 100
    model = Model()
    for index in range(0,1000):
        inputRand = low + (high-low)*random.random()
55 temp = model.schaffer(inputRand)
        self.minVal=self.returnMin(temp)
        self.maxVal=self.returnMax(temp)
        print("Max: %d Min: %d"%(self.maxVal,self.minVal))

60 class FindEnergy:
    emax=0

    def __init__(self,minimum,maximum):
        self.minimum = minimum
        self.maximum = maximum
65 self.maxVal=0

    def returnMax(self,num):
        if(num>self.maxVal):
            self.maxVal=num
70

    def evaluate(self,num):

```

Sep 08, 14 12:24

csc710sbse:hw1:VivekNair:vnair2

Page 2/4

```

    model = Model()
    temp = model.schaffer(num)
    energy = (temp -self.minimum)/(self.maximum-self.minimum)
75 #print "Energy: %f Temp: %f Self.Max: %f Self.Min: %f Num: %f" %(energy,temp
    ,self.minimum,self.maximum,num)
    return energy

    def evaluateEmax(self):
80 low = -100
    high = 100
    model = Model()
    for index in range(0,1000):
        inputRand = low + (high-low)*random.random()
85 temp = model.schaffer(inputRand)
        energy = (temp - self.minimum)/(self.maximum-self.minimum)
        self.returnMax(energy)
    return self.maxVal

90 def doSomethingCool():
    base = BaseLine()
    base.findBaseLine()
    energy = FindEnergy(base.minVal,base.maxVal)
    emax = energy.evaluate(10000)
95 print emax
    # print emax.evaluate()

#class SimulatedAnnealing:
    def evaluate():
100 low=-100
    high=100
    jump = True
    base = BaseLine()
    base.findBaseLine()
105 energy = FindEnergy(base.minVal,base.maxVal)
    emax = 0
    print "Base Line Values: Minimum: %f Maximum: %f Emax: %f" %(base.minVal,base.maxVal,e
    max)

    s = low + (high-low)*random.random() #Initial State
    e = energy.evaluate(s) #Initial Enenergy
110 sb = s #Initial Best Solution
    eb = e #Initial Best Energy
    k = 1
    kmax = 2000
    count=0
115 while(k ≤ kmax ^ e > emax):
        if(jump=False):
            sn = neighbour(s)
        else:
120 sn = low + (high-low)*random.random()
            #jump= False #change
            en = energy.evaluate(sn)
            if(en < eb):
                sb = sn
                eb = en
125 say("!") #we get to somewhere better globally
                tempProb = probFunction(e,en,k/kmax)
                tempRand = random.random()
                # print " tempProb: %f tempRand: %f " %(tempProb,tempRand)
130 if(en < e):
                    s = sn
                    e = en
                    say("+") #we get to somewhere better locally
                elif(tempProb ≤ tempRand):
135 jump = True
                    s = sn
                    e = en
                    say("?") #we are jumping to something sub-optimal;
                    count+=1
140 say(".")
                    k += 1
                    if(k % 50 == 0):
                        print "\n"

```

