```
from __future__ import division
import sys
import random
import math
import numpy as np
sys.dont_write_bytecode = True

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

class Fonseca:
  maxVal=-10000
  minVal=10000

  def returnMin(self,num):
    if(num<self.minVal):
      self.minVal=num
      return num
    else:
      return self.minVal

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

  def fx(self,listpoint,version):
    n=len(listpoint)
    rootn=(n**0.5)**-1
    sum=0
    for i in range(0,n):
      if version == 1:
        sum+=(listpoint[i]-rootn)**2
      elif version == 2:
        sum+=(listpoint[i]+rootn)**2
      else:
        print "STOP MESSING AROUND"
    return (1 - math.exp(-sum))

  def evaluate(self,listpoint):
    energy = self.fx(listpoint,1)+ self.fx(listpoint,2)
    return (energy-self.minVal)/(self.maxVal-self.minVal)

  def baseline(self,minR,maxR):
    for x in range(0,1000):
      solution = [(minR + random.random()*(maxR-minR)) for z in range(0,3)]
      self.returnMax(self.fx(solution,1)+ self.fx(solution,2))
      self.returnMin(self.fx(solution,1)+ self.fx(solution,2))

  def  neighbour(self,minN,maxN):
    return minN + (maxN-minN)*random.random()

  def info(self):
    return "Fonseca~"

class Kursawe:
  maxVal=-10000
  minVal=10000

  def returnMin(self,num):
    if(num<self.minVal):
      self.minVal=num
      return num
    else:
      return self.minVal

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

  def f1(self,listpoint):
    n=len(listpoint)
    #inspired by 'theisencr'
    return np.sum([-10*math.exp(-0.2*(np.sqrt(listpoint[i]**2 + listpoint[i+1]**2))) for i in range (0, n-1)])
    return sum

  def f2(self,listpoint):
    a=0.8
    b=3
    n=len(listpoint)
    #inspired by 'theisencr'
    return np.sum([math.fabs(listpoint[i])**a + 5*np.sin(listpoint[i])**b for i in range (0, n)])


  def evaluate(self,listpoint):
    energy = (self.f1(listpoint)+self.f2(listpoint))
    return (energy-self.minVal)/(self.maxVal-self.minVal)

  def baseline(self,minR,maxR):
    for x in range(0,50000):
      solution = [(minR + random.random()*(maxR-minR)) for z in range(0,3)]
      self.returnMax(self.f1(solution)+ self.f2(solution))
      self.returnMin(self.f2(solution)+ self.f2(solution))

  def  neighbour(self,minN,maxN):
    return minN + (maxN-minN)*random.random()

  def info(self):
    return "Kursawe~"
  def test(self):
    file = open("Kursawe.txt","w")
    for x in range(-5,6):
      for y in range(-5,6):
        for z in range(-5,6):
          solution = [x,y,z]
          file.write("%f\n"%self.evaluate(solution))
    file.close()

class MaxWalkSat():
  model = None
  minR=0
  maxR=0
  def __init__(self,modelName):
    #print "init"
    if modelName == "Fonseca":
      self.model = Fonseca()
      self.minR=-4
      self.maxR=4
      #print "here"
    elif modelName == "Kursawe":
      self.model = Kursawe()
      self.minR=-5
      self.maxR=5
      self.model.test()
      #print "there"
    else:
      print "STOP MESSING AROUND"

  def evaluate(self):
    model = self.model
    print "Model used: %s"%model.info()
    minR=self.minR
    maxR=self.maxR
    maxTries=50
    maxChanges=2000
    n=3
    threshold=0.1
    probLocalSearch=0.25
    bestScore=100
```

```
      bestSolution=[]


     # model = Fonseca()
      model.baseline(minR,maxR)
      print model.maxVal,model.minVal

      for i in range(0,maxTries): #Outer Loop
        solution=[]
        for x in range(0,n):
          solution.append(minR + random.random()*(maxR-minR))
        #print "Solution: ",
        #print solution
        for j in range(0,maxChanges):      #Inner Loop
           score = model.evaluate(solution)
           #print score
           # optional-start
           if(score < bestScore):
             bestScore=score
             bestSolution=solution

           # optional-end
           if(score < threshold):
             print "threshold reached"
             return solution,score
           if random.random() > probLocalSearch:
               c = int(0 + (2-0)*random.random())
               solution[c]=model.neighbour(minR,maxR)
           else:
               tempBestScore=score
               tempBestSolution=solution
               interval = (maxR-minR)/10
               c = int(0 + (2-0)*random.random())
               for itr in range(0,10):
                  solution[c] = minR + (itr*interval)*random.random()
                  tempScore = model.evaluate(solution)
                  if tempBestScore < tempScore:     # score is correlated to max?
                    tempBestScore=tempScore
                    tempBestSolution=solution
               solution=tempBestSolution

      return bestSolution,bestScore

   def doSomethingCool():
      test = MaxWalkSat("Kursawe")
      solution,score = test.evaluate()
      print "Solution: ",
      print solution
      print "Score: ",
      print score
      test = MaxWalkSat("Fonseca")
      solution,score = test.evaluate()
      print "Solution: ",
      print solution
      print "Score: ",
      print score

   if __name__ = '__main__':
     doSomethingCool();


   """
   Model used: Kursawe~
   20.0956977772 -21.4884127271
   threshold reached
   Solution:  [0.027085207995591, -1.3080970300867412, -2.2218969561169333]
   Score:  0.095955894695

   Model used: Fonseca~
   2.0 1.15944597748
   threshold reached
   Solution:  [0.2715980383022991, 0.18246188010940578, 0.8312501646816788]
   Score:  0.0861405819363
```



```
   """
```