#转载https://github.com/guofei9987/scikit-opt/tree/master/sko

import numpy as np

from .base import SkoBase

from sko.operators import mutation

class SimulatedAnnealingBase(SkoBase):

"""

DO SA(Simulated Annealing)

Parameters

----------------

func : function

The func you want to do optimal

n\_dim : int

number of variables of func

x0 : array, shape is n\_dim

initial solution

T\_max :float

initial temperature

T\_min : float

end temperature

L : int

num of iteration under every temperature（Long of Chain）

Attributes

----------------------

Examples

-------------

See https://github.com/guofei9987/scikit-opt/blob/master/examples/demo\_sa.py

"""

#无约束

def \_\_init\_\_(self, func, x0, T\_max=100, T\_min=1e-7, L=300, max\_stay\_counter=150, \*\*kwargs):

assert T\_max > T\_min > 0, 'T\_max > T\_min > 0'

self.func = func

self.T\_max = T\_max # initial temperature

self.T\_min = T\_min # end temperature

self.L = int(L) # num of iteration under every temperature（also called Long of Chain）一个温度下可允许迭代的次数

# stop if best\_y stay unchanged over max\_stay\_counter times (also called cooldown time) 如果 best\_y 在 max\_stay\_counter 次数（也称为冷却时间）内保持不变，则停止

self.max\_stay\_counter = max\_stay\_counter

self.n\_dim = len(x0)

self.best\_x = np.array(x0) # initial solution

self.best\_y = self.func(self.best\_x)

self.T = self.T\_max

self.iter\_cycle = 0

self.generation\_best\_X, self.generation\_best\_Y = [self.best\_x], [self.best\_y]

# history reasons, will be deprecated

self.best\_x\_history, self.best\_y\_history = self.generation\_best\_X, self.generation\_best\_Y

def get\_new\_x(self, x):

u = np.random.uniform(-1, 1, size=self.n\_dim)

x\_new = x + 20 \* np.sign(u) \* self.T \* ((1 + 1.0 / self.T) \*\* np.abs(u) - 1.0)

return x\_new

def cool\_down(self):

self.T = self.T \* 0.7

def isclose(self, a, b, rel\_tol=1e-09, abs\_tol=1e-30):

return abs(a - b) <= max(rel\_tol \* max(abs(a), abs(b)), abs\_tol)

#用于判断连续若干个新解是否都没有被接受

def run(self):

x\_current, y\_current = self.best\_x, self.best\_y

stay\_counter = 0

while True:

for i in range(self.L):

x\_new = self.get\_new\_x(x\_current)

y\_new = self.func(x\_new)

# Metropolis

df = y\_new - y\_current

if df < 0 or np.exp(-df / self.T) > np.random.rand():

x\_current, y\_current = x\_new, y\_new

if y\_new < self.best\_y:

self.best\_x, self.best\_y = x\_new, y\_new

self.iter\_cycle += 1

self.cool\_down()

self.generation\_best\_Y.append(self.best\_y)

self.generation\_best\_X.append(self.best\_x)

# if best\_y stay for max\_stay\_counter times, stop iteration

if self.isclose(self.best\_y\_history[-1], self.best\_y\_history[-2]):表达式结果为1代表连续两个新解都没有被接受

stay\_counter += 1

else:

stay\_counter = 0

if self.T < self.T\_min:

stop\_code = 'Cooled to final temperature'

break

if stay\_counter > self.max\_stay\_counter:

stop\_code = 'Stay unchanged in the last {stay\_counter} iterations'.format(stay\_counter=stay\_counter)

break

return self.best\_x, self.best\_y

fit = run

class SimulatedAnnealingValue(SimulatedAnnealingBase):

"""

SA on real value function

"""

#有约束

def \_\_init\_\_(self, func, x0, T\_max=100, T\_min=1e-7, L=300, max\_stay\_counter=150, \*\*kwargs):

super().\_\_init\_\_(func, x0, T\_max, T\_min, L, max\_stay\_counter, \*\*kwargs)

lb, ub = kwargs.get('lb', None), kwargs.get('ub', None)

if lb is not None and ub is not None:

self.has\_bounds = True

self.lb, self.ub = np.array(lb) \* np.ones(self.n\_dim), np.array(ub) \* np.ones(self.n\_dim)

assert self.n\_dim == len(self.lb) == len(self.ub), 'dim == len(lb) == len(ub) is not True'

assert np.all(self.ub > self.lb), 'upper-bound must be greater than lower-bound'

self.hop = kwargs.get('hop', self.ub - self.lb)

elif lb is None and ub is None:

self.has\_bounds = False

self.hop = kwargs.get('hop', 10)

else:

raise ValueError('input parameter error: lb, ub both exist, or both not exist')

self.hop = self.hop \* np.ones(self.n\_dim)