```python
#jupyter matplotlib内置绘图的魔术命令
%matplotlib inline
#matplotlib绘图、3D相关包
from matplotlib import pyplot as plt
import matplotlib
from mpl_toolkits.mplot3d import Axes3D
#用于动态显示图像
from IPython import display

#运算、操作处理相关包
import numpy as np
import pandas as pd


class GA():

    def __init__(self, nums, bound, func, DNA_SIZE=None, cross_rate=0.8, mutation=0.003):
        nums = np.array(nums)
        bound = np.array(bound)
        self.bound = bound
        if nums.shape[1] != bound.shape[0]:
            raise Exception('You have {nums.shape[1]} variables, but {bound.shape[0]} ranges')

        for var in nums:
            for index, var_curr in enumerate(var):
                if var_curr < bound[index][0] or var_curr > bound[index][1]:
                    raise Exception('{var_curr}is not within the range')

        for min_bound, max_bound in bound:
            if max_bound < min_bound:
                raise Exception('Sorry,({min_bound}, {max_bound})is not reasonable')

        # 所有变量的最小值和最大值
        # var_len为所有变量的取值范围大小
        # bit为每个变量按整数编码最小的二进制位数
        min_nums, max_nums = np.array(list(zip(*bound)))
        self.var_len = var_len = max_nums - min_nums
        bits = np.ceil(np.log2(var_len + 1))

        if DNA_SIZE == None:
            DNA_SIZE = int(np.max(bits))
        self.DNA_SIZE = DNA_SIZE

        # POP_SIZE为进化的种群数
        self.POP_SIZE = len(nums)
        POP = np.zeros((*nums.shape, DNA_SIZE))
        for i in range(nums.shape[0]):
            for j in range(nums.shape[1]):
                # 编码方式:
                num = int(round((nums[i, j] - bound[j][0]) * ((2 ** DNA_SIZE) / var_len[j])))
                # 用python自带的格式化转化为前面空0的二进制字符串，然后拆分成列表
                POP[i, j] = [int(k) for k in ('{0:0' + str(DNA_SIZE) + 'b}').format(num)]
        self.POP = POP
        # 用于后面重置（reset）
        self.copy_POP = POP.copy()
        self.cross_rate = cross_rate
        self.mutation = mutation
        self.func = func
#将编码后的DNA翻译回来（解码）
    def translateDNA(self):
        W_vector = np.array([2 ** i for i in range(self.DNA_SIZE)]).reshape((self.DNA_SIZE, 1))[::-1]
        binary_vector = self.POP.dot(W_vector).reshape(self.POP.shape[0:2])
        for i in range(binary_vector.shape[0]):
            for j in range(binary_vector.shape[1]):
                binary_vector[i, j] /= ((2 ** self.DNA_SIZE) / self.var_len[j])
                binary_vector[i, j] += self.bound[j][0]
        return binary_vector
```

```python
#得到适应度
    def get_fitness(self, non_negative=False):
        result = self.func(*np.array(list(zip(*self.translateDNA()))))
        self.maxresult = np.max(result)
        print(self.maxresult)
        self.maxPOP = self.POP[np.argmax(result)]
        if non_negative:
            min_fit = np.min(result, axis=0)
            result -= min_fit
        return result
#自然选择
    def select(self):
        fitness = self.get_fitness(non_negative=True)
        self.POP = self.POP[np.random.choice(np.arange(self.POP.shape[0]),
        size=self.POP.shape[0], replace=True,
                                             p=fitness / np.sum(fitness))]
#染色体交叉
    def crossover(self):
        for people in self.POP:
            if np.random.rand() < self.cross_rate:
                i_ = np.random.randint(0, self.POP.shape[0], size=1)
                cross_points = np.random.randint(0, 2, size=(len(self.var_len),
                self.DNA_SIZE)).astype(np.bool)
                people[cross_points] = self.POP[i_, cross_points]
#基因变异
    def mutate(self):
        for people in self.POP:
            for var in people:
                for point in range(self.DNA_SIZE):
                    if np.random.rand() < self.mutation:
                        if var[point] == 1:
                            var[point] = 0
                        else:
                            var[point] = 1
                       # var[point] = 1 if var[point] == 0 else 1
#进化
    def evolution(self):
        self.select()
        self.crossover()
        self.mutate()
#重置
    def reset(self):
        self.POP = self.copy_POP.copy()
#打印当前状态日志
    def log(self):
        return pd.DataFrame(np.hstack((self.translateDNA(),
        self.get_fitness().reshape((len(self.POP), 1)))),
                            columns=['x{i}' for i in range(len(self.var_len))] +
                            ['F'])
#一维变量作图
    def plot_in_jupyter_1d(self, iter_time=200):
        is_ipython = 'inline' in matplotlib.get_backend()
        if is_ipython:
            from IPython import display

        plt.ion()
        for _ in range(iter_time):
            plt.cla()
            x = np.linspace(*self.bound[0], self.var_len[0] * 50)
            plt.plot(x, self.func(x))
            x = self.translateDNA().reshape(self.POP_SIZE)
            plt.scatter(x, self.func(x), s=200, lw=0, c='red', alpha=0.5)
            if is_ipython:
                display.clear_output(wait=True)
                display.display(plt.gcf())
            self.evolution()

# 单变量遗传算法测试
func = lambda x:np.sin(10*x)*x + np.cos(2*x)*x
ga = GA([[np.random.rand()*5] for _ in range(100)], [(0,5)], DNA_SIZE=10, func=func)
ga.plot_in_jupyter_1d()
print(ga.log().max())
print(ga.maxPOP)
print(ga.maxresult)

# 多变量遗传算法测试
# nums = list(zip(np.arange(-2, 2, 0.2), np.arange(-2, 2, 0.2)))
# bound = [(-2, 2), (-2, 2)]
# func = lambda x, y: x*np.cos(2*np.pi*y)+y*np.sin(2*np.pi*x)
# DNA_SIZE = 20
# cross_rate = 0.7
# mutation = 0.01

# ga = GA(nums=nums, bound=bound, func=func, DNA_SIZE=DNA_SIZE, cross_rate=cross_rate, mutation=mutation)
# is_ipython = 'inline' in matplotlib.get_backend()
# if is_ipython:
#     from IPython import display

# plt.ion() #打开交互式
# for _ in range(200):
#     plt.cla() #清空画布
#     X = Y = np.arange(-2, 2, 0.2)
#     X, Y=np.meshgrid(X,Y)
#     Z = func(X, Y)

#     fig1=plt.figure()
#     ax=Axes3D(fig1)
#     ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=plt.cm.coolwarm)#做曲面
#     ax.set_xlabel('x0 label', color='r')
#     ax.set_ylabel('x1 label', color='g')
#     ax.set_zlabel('F label', color='b')#给三个坐标轴注明

#     ax.scatter(*list(zip(*ga.translateDNA())), ga.get_fitness(), s=100, lw=0, c='red', alpha=0.5)

#     if is_ipython:
#         display.clear_output(wait=True)
#         display.display(plt.gcf())

#     ga.evolution()

# print(ga.log().max())
# print(ga.maxPOP)
# print(ga.maxresult)
```