有了刚才的基础知识，知道了遗传算法主要是通过搜索从而得到最优解（往往就是极值）

那就来看看下面这个问题使用遗传算法来解决

需要求解

$$f(x,y)=0.5-\frac{sin^2\sqrt{x^2+y^2}-0.5}{1+0.001(x^2+y^2)^2}$$

在x,y均属于[-10,10]之间的最大值

1.**首先我们要规定编码方式。** 这里我一共25位二进制编码（精确到小数点后六位）

2.**找到我们的适应度函数。** 在这里要求我们去找f(x,y)的最大值，所以我们只要挑选能让f(x,y)越大越好的x,y，所以f(x,y)就是我们的适应度函数

3.**选择、交叉和变异。** 选择就是会较大概率选择适应度高的个体；交叉就是遍历所有个体，挑选另一个进行交叉，规定取待配对的前半部分和挑选出来个体的后半部分进行交叉；变异就是随机挑选基因序列的某一位置上的1和0互换。

4.**我们要找到一个能把25位二进制数还原到[-10,10]之间的函数**

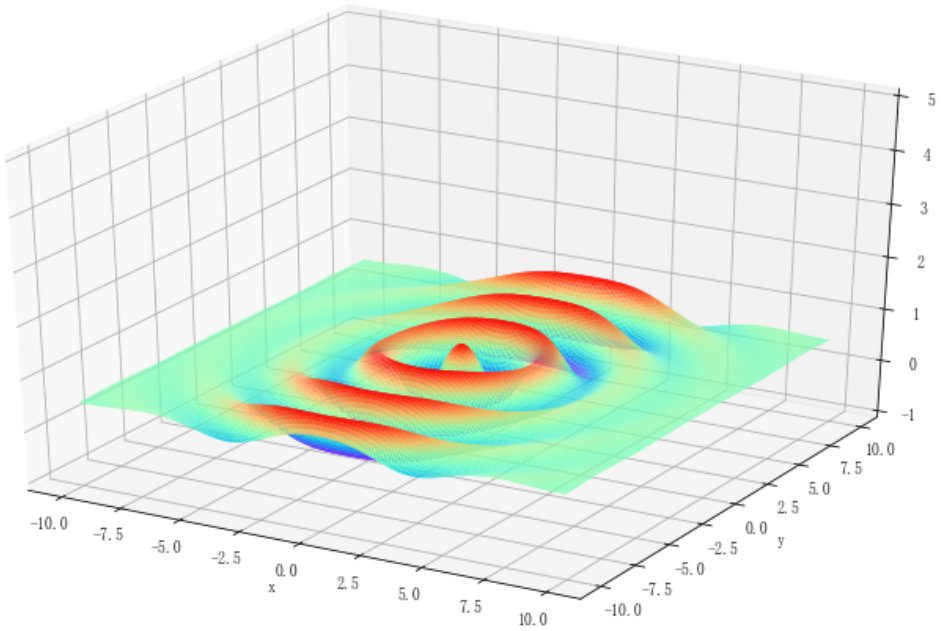
$$f(c)=-10+c\cdot\frac{10-(-10)}{2^{25}-1}$$

In [1]:

```
# 开始写代码
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['font.sans-serif']='Fangsong'
plt.rcParams['axes.unicode_minus']=False
```

In [2]:

```
# 首先可视化一下这个f(x,y)
fig = plt.figure(figsize=(10,6))
ax = Axes3D(fig)
x = np.arange(-10, 10, 0.1)
y = np.arange(-10, 10, 0.1)
X, Y = np.meshgrid(x, y)
Z = 0.5 - (np.sin(np.sqrt(X**2+Y**2))**2 - 0.5)/(1 + 0.001*(x**2 +y**2)**2)
plt.xlabel('x')
plt.ylabel('y')
ax.set_zlim([-1,5])
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='rainbow')
plt.show()
```



In [3]:

```
#开始遗传算法求最大值
class Population:
    # 种群的设计
    def __init__(self, size, chrom_size, cp, mp, gen_max):
        # 种群信息合
        self.individuals = []          # 个体集合
        self.fitness = []              # 个体适应度集
        self.selector_probability = [] # 个体选择概率集合
        self.new_individuals = []      # 新一代个体集合

        self.elitist = {'chromosome':[0, 0], 'fitness':0, 'age':0} # 最佳个体的信息

        self.size = size # 种群所包含的个体数
        self.chromosome_size = chrom_size # 个体的染色体长度
        self.crossover_probability = cp   # 个体之间的交叉概率
        self.mutation_probability = mp    # 个体之间的变异概率

        self.generation_max = gen_max # 种群进化的最大世代数
        self.age = 0                  # 种群当前所处世代

        # 随机产生初始个体集，并将新一代个体、适应度、选择概率等集合以 0 值进行初始化
        v = 2 ** self.chromosome_size - 1
        for i in range(self.size):
            self.individuals.append([random.randint(0, v), random.randint(0, v)])
            self.new_individuals.append([0, 0])
            self.fitness.append(0)
            self.selector_probability.append(0)

    # 基于轮盘赌博机的选择
    def decode(self, interval, chromosome):
        '''将一个染色体 chromosome 映射为区间 interval 之内的数值'''
        d = interval[1] - interval[0]
        n = float (2 ** self.chromosome_size -1)
        return (interval[0] + chromosome * d / n)

    def fitness_func(self, chrom1, chrom2):
        '''适应度函数，可以根据个体的两个染色体计算出该个体的适应度'''
        interval = [-10.0, 10.0]
        (x, y) = (self.decode(interval, chrom1),
                  self.decode(interval, chrom2))
        n = lambda x, y: math.sin(math.sqrt(x*x + y*y)) ** 2 - 0.5
        d = lambda x, y: (1 + 0.001 * (x*x + y*y)) ** 2
        func = lambda x, y: 0.5 - n(x, y)/d(x, y)
        return func(x, y)

    def evaluate(self):
        '''用于评估种群中的个体集合 self.individuals 中各个个体的适应度'''
        sp = self.selector_probability
        for i in range (self.size):
            self.fitness[i] = self.fitness_func (self.individuals[i][0],    # 将计算结果保存在 se
lf.fitness 列表中
                                                 self.individuals[i][1])
        ft_sum = sum (self.fitness)
        for i in range (self.size):
            sp[i] = self.fitness[i] / float (ft_sum)    # 得到各个个体的生存概率
        for i in range (1, self.size):
            sp[i] = sp[i] + sp[i-1]    # 需要将个体的生存概率进行叠加，从而计算出各个个体的选择概
率
```

```
    # 轮盘赌博机（选择）
    def select(self):
        (t, i) = (random.random(), 0)
        for p in self.selector_probability:
            if p > t:
                break
            i = i + 1
        return i

    # 交叉
    def cross(self, chrom1, chrom2):
        p = random.random()     # 随机概率
        n = 2 ** self.chromosome_size -1
        if chrom1 != chrom2 and p < self.crossover_probability:
            t = random.randint(1, self.chromosome_size - 1)    # 随机选择一点（单点交叉）
            mask = n << t     # << 左移运算符
            (r1, r2) = (chrom1 & mask, chrom2 & mask)    # & 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0
            mask = n >> (self.chromosome_size - t)
            (l1, l2) = (chrom1 & mask, chrom2 & mask)
            (chrom1, chrom2) = (r1 + l2, r2 + l1)
        return (chrom1, chrom2)

    # 变异
    def mutate(self, chrom):
        p = random.random ()
        if p < self.mutation_probability:
            t = random.randint (1, self.chromosome_size)
            mask1 = 1 << (t - 1)
            mask2 = chrom & mask1
            if mask2 > 0:
                chrom = chrom & (~mask2)  # ~ 按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1
            else:
                chrom = chrom ^ mask1    # ^ 按位异或运算符：当两对应的二进位相异时，结果为1
        return chrom

    # 保留最佳个体
    def reproduct_elitist (self):
        # 与当前种群进行适应度比较，更新最佳个体
        j = -1
        for i in range (self.size):
            if self.elitist['fitness'] < self.fitness[i]:
                j = i
                self.elitist['fitness'] = self.fitness[i]
        if (j >= 0):
            self.elitist['chromosome'][0] = self.individuals[j][0]
            self.elitist['chromosome'][1] = self.individuals[j][1]
            self.elitist['age'] = self.age

    # 进化过程
    def evolve(self):
        indvs = self.individuals
        new_indvs = self.new_individuals
        # 计算适应度及选择概率
        self.evaluate()
        # 进化操作
        i = 0
        while True:
            # 选择两个个体，进行交叉与变异，产生新的种群
            idv1 = self.select()
```

```
            idv2 = self.select()
            # 交叉
            (idv1_x, idv1_y) = (indvs[idv1][0], indvs[idv1][1])
            (idv2_x, idv2_y) = (indvs[idv2][0], indvs[idv2][1])
            (idv1_x, idv2_x) = self.cross(idv1_x, idv2_x)
            (idv1_y, idv2_y) = self.cross(idv1_y, idv2_y)
            # 变异
            (idv1_x, idv1_y) = (self.mutate(idv1_x), self.mutate(idv1_y))
            (idv2_x, idv2_y) = (self.mutate(idv2_x), self.mutate(idv2_y))
            (new_indvs[i][0], new_indvs[i][1]) = (idv1_x, idv1_y)  # 将计算结果保存于新的个体集合self.new_individuals中
            (new_indvs[i+1][0], new_indvs[i+1][1]) = (idv2_x, idv2_y)
            # 判断进化过程是否结束
            i = i + 2          # 循环self.size/2次，每次从self.individuals 中选出2个
            if i >= self.size:
                break

        # 最佳个体保留
        # 如果在选择之前保留当前最佳个体，最终能收敛到全局最优解。
        self.reproduct_elitist()

        # 更新换代：用种群进化生成的新个体集合 self.new_individuals 替换当前个体集合
        for i in range (self.size):
            self.individuals[i][0] = self.new_individuals[i][0]
            self.individuals[i][1] = self.new_individuals[i][1]

    def run(self):
        '''根据种群最大进化世代数设定了一个循环。
        在循环过程中，调用 evolve 函数进行种群进化计算，并输出种群的每一代的个体适应度最大值、平均值和最小值。'''
        for i in range (self.generation_max):
            self.evolve ()
            if i%(self.generation_max/10)==0:
                print(i, max (self.fitness), sum (self.fitness)/self.size, min (self.fitness))
        return max(self.fitness),self.elitist
if __name__ == '__main__':
    # 种群的个体数量为 150，染色体长度为 25，交叉概率为 0.85，变异概率为 0.2,进化最大世代数为 200
    pop = Population (150, 25, 0.85, 0.2, 200)
    result,final_elitist=pop.run()
```

```
0 0.9870583538015455 0.49552063082004677 0.018472797197006563
20 0.9898884778845176 0.7600616941986033 0.02186413248353203
40 0.989480559496503 0.8364586618339408 0.025150255036704816
60 0.990275454200723 0.797441966436177 0.026072698534625238
80 0.9902653706811677 0.8197422876363014 0.019338985465759573
100 0.9902376353460567 0.8516011811775622 0.02536936725357758
120 0.9902814225281639 0.8555302702196285 0.014155324082010312
140 0.9902584765213842 0.918190408119946 0.13303098624249887
160 0.9902810110074733 0.8921573473416449 0.027224042270887372
180 0.9902840677472808 0.8976544130324127 0.023073770941760274
```

In [4]:

```
print(result)
print(final_elitist)
```

```
0.9902840413606654
{'chromosome': [16878614, 16729291], 'fitness': 0.9955334733339789, 'age': 0}
```

In [5]:

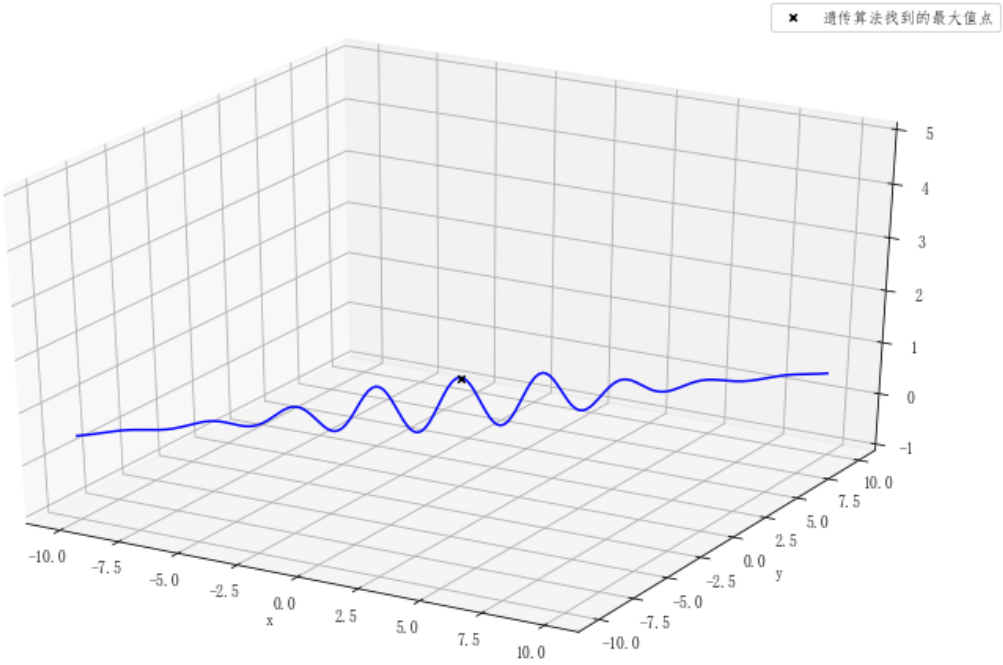
```
def decode(interval, chromosome,chromosome_size):
        '''将一个染色体 chromosome 映射为区间 interval 之内的数值'''
        d = interval[1] - interval[0]
        n = float (2 **chromosome_size -1)
        return (interval[0] + chromosome * d / n)

interval=[-10.0,10.0]
x,y=final_elitist['chromosome']
x1=decode(interval,x,25)
y1=decode(interval,y,25)
print(x1,y1)
```

0.06043821753377365 -0.028565228836692214

In [6]:

```
# 带入求得的最大点再来可视化一下这个f(x,y)
fig = plt.figure(figsize=(10,6))
ax = Axes3D(fig)
x = np.arange(-10, 10, 0.1)
y = np.arange(-10, 10, 0.1)
z=0.5 - (np.sin(np.sqrt(x**2+y**2))**2 - 0.5)/(1 + 0.001*(x**2 + y**2)**2)
ax.scatter(x1, y1,result,color='black',label='遗传算法找到的最大值点',marker='x')
ax.plot(x,y,z,color='b')
plt.xlabel('x')
plt.ylabel('y')
ax.set_zlim([-1,5])
plt.legend()
plt.show()
```



可以发现很贴近真实最大值，当然这也显示了遗传算法并不能完全的求出解，而是得到近似解。并且遗传算法还有可能陷入局部极值，所以设置好参数也是特别重要的。

to sthx