mina ilkhani 610398191

HW5 PSO

import all we need for solving and visualizing:

In [1]:

```
import numpy as np
import random
import math

import matplotlib.pyplot as plt
```

problem:

In [2]:

```
XYMaxF = 10
XYMinF = -10
def f(x,y):
    return  abs( math.sin(x) * math.cos(y) * math.exp( abs( 1 - ( (math.sqrt( pow(x,2) + pow(y,2) ) ) / math.pi ) ) ) )

XYMaxG = 100
XYMinG = -100
def g(x,y):
    if x == 0 or math.cos(y/x) == -1:
        # return math.inf  #make problme for ploting.
        return 41000       #if it be a very large number it would not look well
    return x * math.sin( math.pi * math.cos(x) * math.tan(y))   *   (math.sin(y/x)) / (1 + math.cos(y/x))
```

visualizing:

In [3]:

```
def point_visualizing(swarm,func,i) :
    xdata = []
    ydata = []
    zdata = []
    for p in swarm:
        xdata.append(p.currX[0])
        ydata.append(p.currX[1])
        zdata.append(func(p.currX[0],p.currX[1]))

    ax = plt.axes(projection='3d')
    ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='viridis');
    plt.savefig('pic/'+func.__name__+str(i)+'.png')               #saving
                                                                   #very slow

def continuous_visualizing(func,maxX,minX,maxY,minY ):
    xdata = []
    ydata = []
    zdata = []

    for i in list(np.arange(minX,maxX,1/2)):
        for j in list(np.arange(minY,maxY,1/2)):
            xdata.append(i)
            ydata.append(j)
    for i in range(len(xdata)):
        zdata.append(func(xdata[i],ydata[i]))

    ax = plt.axes(projection='3d')
    ax.plot_trisurf(xdata, ydata, zdata, cmap='viridis');
```

Parameter variable:

omega, c1, c2, randmin, randmax (used in cal_next() in class particle)

GBlocation : (x,y) , GBest : z for global best (x,y,z)

n : number of particles

In [4]:

```
omega = 1/2
c1 = 1
c2 = 1
randMin = 0
randMax = 1
GBest = 0 # it will be inf for  minimizing and -inf for maximazing
GBlocation = tuple

n = 500
```

currX : x(k)

currV : v(k)

extermum can be min or max

cal_next_v:

```
v(k) -> v(k+1)

x(k) -> x(k+1)
```

```
and update personal best
```

inertia = omega * v(k)

cognitive = c1 *rand* (Personal Best-x(k))

social = c2 *rand* (Global Best-x(k))

vi(k+1) = inertia + cognitive + social

let the location be int. I'll try float too

first self.currV is 0. we should try to keep currX in range ((maxX, minX), (maxY, minY)).usually personal & global best can't tend currX out of the range, but a radom currV can. So I prefer to use small currV in start, but I'll try other numbers.

outOfRange: i used it for fixing parameters(see in readme)

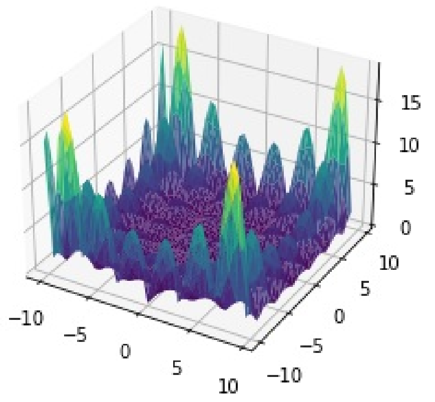
In [5]:

```
class particle:
    def __init__(self,maxX,minX,maxY,minY,func,extermum) :
        self.currX = (random.randint(minX,maxX), random.randint(minY, maxY))
        self.currV = (0,0)
        self.extermum = extermum
        self.func = func

        self.minX = minX
        self.maxX = maxX
        self.minY = minY
        self.maxY = maxY
        # self.outOfRange = 0

        self.PBest = func(self.currX[0], self.currX[1])
        self.PBestlocation = self.currX

    def cal_next(self):
        rand1 = random.uniform(randMin, randMax)
        rand2 = random.uniform(randMin, randMax)

        interia    = (omega*self.currV[0],omega*self.currV[1])
        congnitive = c1*((self.PBestlocation[0]-self.currX[0])*rand1,  (self.PBestlocation[1]-self.currX[1])*rand1 )
        social     = c2*((GBlocation[0]        -self.currX[0])*rand2,  (GBlocation[1]        -self.currX[1])*rand2 )

        self.currV =  ( interia[0]+congnitive[0]+social[0] , interia[1]+congnitive[1]+social[1] )
        nextX  = (self.currX[0] + self.currV[0], self.currX[1] + self.currV[1])
        if self.minX<=nextX[0] and nextX[0]<=self.maxX and self.minY<=nextX[1] and nextX[1]<=self.maxY :
            self.currX = (self.currX[0] + self.currV[0], self.currX[1] + self.currV[1])


            if self.extermum == "min" :
                if self.func(self.currX[0], self.currX[1]) <= self.PBest :
                    self.PBest = self.func(self.currX[0], self.currX[1])
                    self.PBestlocation = self.currX
            elif self.extermum == "max" :
                if self.func(self.currX[0], self.currX[1]) >= self.PBest :
                    self.PBest = self.func(self.currX[0], self.currX[1])
                    self.PBestlocation = self.currX

        # else:
        #     self.outOfRange+=1                                #
```

visualize f

In [6]:

```
continuous_visualizing(f,XYMaxF, XYMinF, XYMaxF, XYMinF)
```

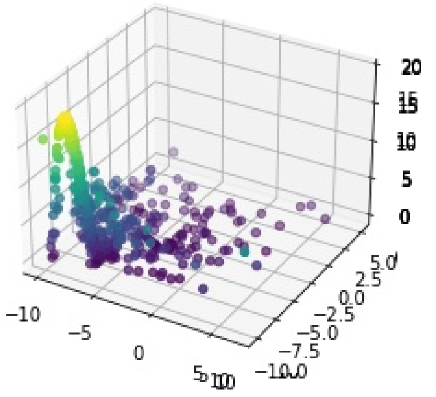


solving f:

In [7]:

```
GBest = -math.inf
swarm = np.array([particle(XYMaxF,XYMinF,XYMaxF,XYMinF,f,"max") for _ in range(n)])
i = 0
while GBest<=19.2:
    point_visualizing(swarm, f,i)
    for p in swarm :
        pass
        if p.PBest > GBest:                               #I tried > too. there was no difference
            GBest = p.PBest
            GBlocation = p.PBestlocation
    for p in swarm :
        p.cal_next()

    i+=1
print(GBest,i)
print(GBlocation)
```

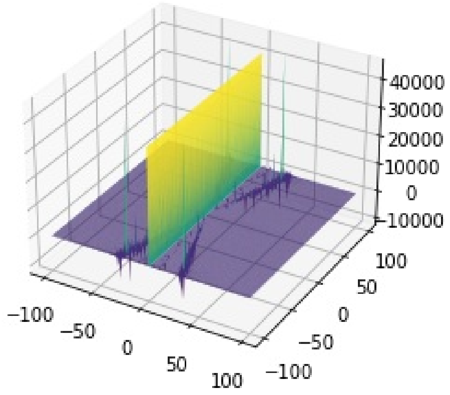
```
19.201122324892612 4
(-8.048384443890424, -9.638359030542738)
```



visualize g

In [8]:

```
continuous_visualizing(g,XYMaxG, XYMinG, XYMaxG, XYMinG)
```



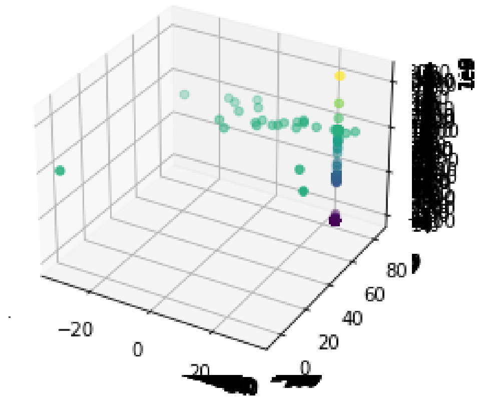
In [9]:

```
GBest = math.inf
swarm = np.array([particle(XYMaxG,XYMinG,XYMaxG,XYMinG,g,"min") for _ in range(n)])
i = 0
# import time
# curr = time.perf_counter()
while i<1000:
    for p in swarm :
        if p.PBest < GBest:                         #I tried > too. there was no difference
            GBest = p.PBest
            GBlocation = p.PBestlocation
            # print(GBest,i, time.perf_counter()-curr)
            print(GBest,i)
            point_visualizing(swarm, g, i)

    for p in swarm :
        p.cal_next()

    i+=1
print("======================")
print(GBest)
print(GBlocation)
```

```
-356.2049429735594 0
-701.5161146873234 0
-1953.9270049353966 0
-4525.333690239846 2
-4842.833424898149 3
-8634.565680577623 5
-18391.47903443583 8
-33830.89829665237 10
-60090.95780847158 12
-1306135.614404037 14
-1364592.3892285936 21
-3052518.026442055 22
-13308692.794147462 24
-31177552.259327404 29
-94576477.88649112 31
-202345680.18814838 36
-688381383.2889314 37
-3018868428.5174685 40
-3078777346.9055295 47
-3136395388.860655 47
-3187314842.1601386 47
-3606079790.7343273 47
-3686572094.3698463 50
-3711380953.5827394 50
-3798406788.9918 52
-3826848781.0005774 55
-3826982837.333254 57
-3829967349.2788343 61
-3833729471.5348287 64
-3837475878.4924994 68
```

```
-3843142381.1364303 69
-3843976720.994551 69
-3845394220.3482533 70
-3845845926.557491 70
-3850460984.2502165 71
-3851982519.7242618 71
-3855567578.6560946 71
-3857961141.1456847 72
-3858712517.420754 72
-3864340061.6946764 73
-3865410958.7179384 74
-3865561431.5830793 74
-3869608392.867829 74
-3870036619.489402 75
-3870446554.4225526 75
-3873974984.390079 75
-3875105698.1466646 75
-3877132962.1326265 76
-3877256119.0531697 76
-3880476876.6417828 76
-3882880205.381526 77
-3886817949.63751 77
-3891457007.9705863 78
-3892497565.862824 79
-3898484582.126428 79
-3898854844.610162 79
-3900455704.51049 80
-3902151779.625427 80
-3902167228.8731813 80
-3905070912.8250337 80
-3905329596.504096 81
-3909155970.2902694 81
-3911195604.0926228 81
-3915080396.3871613 82
-3919268155.2643814 82
-3921011973.590339 82
-3922583437.0019712 83
-3925721353.040852 83
-3926519596.406769 83
-3927407133.656759 83
-3935693553.035016 84
-3935881346.9403796 84
-3936367641.142929 84
-3936767265.9548583 84
-3939089976.7458754 85
-3944695778.470041 85
-3946409576.672734 85
-3949309304.3683386 86
-3951029172.6045713 86
-3952234585.7609143 86
-3956022928.772623 86
-3959060786.3967094 86
-3963184111.53459 87
-3964114092.715457 87
-3965622342.5129423 87
-3974038849.4312215 87
-3977899127.937784 88
-3978005046.047895 88
-3979312078.9542866 88
-3981112502.6757526 88
-3981252596.0248876 88
-3985560835.1669564 88
-3988451351.4075537 88
-3997085392.2590246 88
-4001211275.3423038 89
-4002072315.7671185 89
-4007918546.997605 89
-4011997260.3951306 89
-4015504699.6235943 89
-4017887944.6413593 90
-4018429552.219232 90
-4020640870.890917 90
-4020832253.136549 90
-4021448044.5595784 90
-4022796138.042233 90
-4023248510.093947 90
-4023972251.379673 90
-4024352412.2480345 90
-4028116602.697451 91
-4029118140.5171456 91
-4033473738.5877204 91
-4033518308.8531303 91
-4035144231.9344378 91
-4043076917.2104864 91
-4043536553.8005815 92
-4046417059.3805676 92
-4047581145.6243505 92
-4050462004.650523 92
-4052061249.7536583 92
-4052383830.901971 93
-4058363283.5858274 93
-4058370209.156457 93
-4061252409.5461473 93
-4061584176.1525784 94
-4064133099.256012 94
-4064555439.3789587 94
-4065309514.2085757 94
-4067776905.4755325 94
-4067888079.0824723 94
-4068023711.380972 94
-4068819064.6523623 95
-4074241347.257676 95
-4074936123.504792 95
```

```
-4077236307.6040916 96
-4079048413.9298544 96
-4079600047.066464 96
-4080056656.2728477 96
-4080619557.4582105 96
-4083140093.879086 97
-4083349010.1865406 97
-4083960560.3022094 98
-4084208153.334422 98
-4084239875.038892 98
-4084255515.528129 99
-4084373158.7458024 99
-4084419990.3207235 99
-4084436765.633432 99
-4084441162.30159 101
-4084447395.761326 102
-4084448143.079987 103
-4084448434.205296 103
-4084448768.432027 105
-4084449001.1125517 106
-4084449701.7399745 106
-4084449899.216323 108
-4084449900.866697 109
-4084449900.9553747 113
-4084449901.0303483 114
-4084449901.031764 115
-4084449901.0329714 117
-4084449901.032983 121
-4084449901.0330763 123
-4084449901.0330963 124
======================
-4084449901.0330963
(24.847385070152978, 78.06036194384153)
```



create gif one time whole imgs are for f and other time are g

In [ ]:

```python
from PIL import Image
import glob

frames = []
imgs = glob.glob("*.png")
for i in imgs:
    new_frame = Image.open(i)
    frames.append(new_frame)

frames[0].save('g-f.gif', format='GIF', append_images=frames[1:], save_all=True, duration=1000, loop=1)
```