P7

edgardjfc

March 2022

1 Introduction

For this assignment we observed the behaviour of code that seeks the highest point in a oscillatory function. By going back and forth different values and having a bias towards the highest point, within a function in two dimensions. This type of code can be helpful to analyse the type of surface in a material, like finding the peaks of the surface in a crystal lattice.

2 Code

This code was based on the initial code given by our teacher E. Schaeffer [2] and classmate by the alias FeroxDeitas [1]. With this code we give a function (1) and scan through it to find its highest peaks.

$$g(x, y) = \cos^3 x + 0.1x + \sin^3 y + 0.1y \tag{1}$$

Written in the code as:

```
def g(x, y):
    px = (cos(x))**3 + 0.1 * x
    py = (sin(y))**3 + 0.1 * y
    return px + py

low = -10
high = -low
step = 0.05
p = np.arange(low, high, step)
n = len(p)
z = np.zeros((n, n), dtype=float)
valores=[]
paso = 0.5
```

Here we have the initial conditions for the experiment

```
digitos = floor(log(100, 10)) + 1
guardar = 0
for i in range(n):
    x = p[i]
    for j in range(n):
        y = p[n - j - 1] # voltear
        z[i, j] = g(x, y)
        valores.append(g(x, y))
```

This part of the code calls the function to be analysed, so that it can be turned into a plane with different values corresponding to the equation.

```
tmax=10
for a in range(10, 31, 10):

    resultados = pd.DataFrame()
    for tiem in range(2, 5):
        agentes =  pd.DataFrame()
        agentes['x'] = [uniform(low, high) for i in range(a)]
        agentes['y'] = [uniform(low, high) for i in range(a)]
        agentes['best'] = [min(valores) for i in range(a)]
        bestx = agentes['x'][0]
        besty = agentes['y'][0]
        best = g(bestx, besty)
```

With this part of the code giving the initial position for the different agents that will be searching the lowest point in the graph.

```
 for i in range(a):
                r = agentes.iloc[i]

                xl = r.x - r.dx
                xr = r.x + r.dx
                yd = r.y - r.dy
                yu = r.y + r.dy

                if  xl < low+step:
                    xl = r.x
                if  xr > high-step:
                    xr = r.x
                if  yd < low+step:
                    yd = r.y
                if  yu > high-step:
                    yu = r.y

                g1 = g(xl, yu)
                g2 = g(r.x, yu)
```

```
g3 = g(xr, yu)
g4 = g(xl, r.y)
g5 = g(xr, r.y)
g6 = g(xl, yd)
g7 = g(r.x, yd)
g8 = g(xr, yd)
lista = [g1,g2,g3,g4,g5,g6,g7,g8]
```

This part of the code is in charge of choosing the next step for each of the agents within the plane, searching for the next point to move towards a higher value, while also making sure that the agents do not leave the range of that is being evaluated

```
mayor = lista.index(max(lista))+1

                if mayor == 1:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = yu
                elif mayor ==2:
                    agentes.at[i, 'x'] = r.x
                    agentes.at[i, 'y'] = yu
                elif mayor ==3:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = yu
                elif mayor ==4:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = r.y
                elif mayor ==5:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = r.y
                elif mayor ==6:
                    agentes.at[i, 'x'] = xl
                    agentes.at[i, 'y'] = yd
                elif mayor ==7:
                    agentes.at[i, 'x'] = r.x
                    agentes.at[i, 'y'] = yd
                elif mayor ==8:
                    agentes.at[i, 'x'] = xr
                    agentes.at[i, 'y'] = yd

                mejor = g(r.x, r.y)
                if mejor > best:
                    best = g(r.x, r.y)
                    bestx = r.x
                    besty = r.y
```

```python
        if mejor > r.best:
            agentes.at[i, 'best'] = mejor
```

The next part of the code compares the current value of the agent to the previous one to see which one is higher.

3 Results

The code produced multiple repetitions of the experiment, with 10, 20 and 30 runs of 100, 1000 and 10000 steps, creating the following graphs for the highest values.

	10 runs in 100	10 runs in 1000	10 runs in 10000
0	2.160037180259553	2.160413658081958	2.1604099069282827
1	2.1604014868483787	0.9037402642303191	2.7887334606663914
2	0.7600690545586284	2.788733335386371	2.160415117819295
3	1.53167168417929	-0.49560499496575705	2.1604139563765012
4	1.5314060879629359	1.5320925934323002	2.1604133551907205
5	-0.49651204543336547	-1.9464823331924506	2.1604145391559677
6	0.902891809323771	0.877103671274068	0.90377787725377
7	1.5318005194121884	3.417041070922096	1.5320964739560623
8	2.160382268942294	1.5320680984374628	1.5320946466745233
9	3.416824298705477	2.160395969933341	2.7887323559017663

	20 runs in 100	20 runs in 1000	20 runs in 10000
0	1.532008488386705	1.5320737581862374	2.160409383603041
1	2.1603566362811675	2.160413026034921	1.5320934682866623
2	2.160034775452343	1.3019876992333126	2.16041491294221
3	0.9037402756144008	3.4170428778253696	1.532096268717873
4	2.788349841923174	0.9037223532530994	2.7887319640115065
5	1.5310550500255866	1.532094372611541	0.9037761463001547
6	2.1603718192475956	-0.699743616211008	2.1604134195136533
7	0.9035397908863004	2.788729386406952	-0.6996819962026415
8	0.9036904229947663	1.532095660714538	0.7610366113572377
9	2.788543453619404	2.7887097297746672	2.7887328930641155
10	0.0399620264954999	2.1603963583038794	-2.5634757987083026
11	2.788552496588196	2.160389839086628	2.1604146191066373
12	2.160404764674713	2.7887214895374752	2.7887311761329636
13	1.5315594124875136	3.4170506186602223	2.160414161253486
14	0.9034280405191928	1.53205908952163	2.788732353700122
15	2.1604016299711035	2.1604120772370212	2.7887325223451187
16	2.1603639951453664	3.417037980209532	0.877719768920371
17	1.5319673733154229	1.5320655157941543	2.788733487352597
18	2.1603869709806522	1.5320955731985453	2.7887278742053097
19	2.160385560831832	-1.3166065303328591	2.788732309492447

	30 runs in 100	30 runs in 1000	30 runs in 10000
0	2.788425476231396	0.9037700865977252	0.903776557151953
1	2.7883399582856936	1.532074486133586	2.7887315724131416
2	0.13061953189689335	1.5320957690790096	2.1604140287348708
3	0.1252482188016666	0.9037574706208708	2.160415086819665
4	2.1596563496408256	3.4170480170196145	2.1604144018914972
5	2.788677825690458	2.7886961381945556	1.5320960242660164
6	2.160323404409406	3.416979283798037	2.7887334385484808
7	2.1601902223612743	2.1604060638068976	2.160414406094923
8	1.5315220726751657	-0.495661427242728	0.9037774522317218
9	0.7496536128272596	1.532084575133723	1.5320954433055998
10	0.7586432244304626	3.416992962975852	0.9037769876670709
11	2.7884383985196286	3.417033166214299	2.7887330404606163
12	3.4166043883441466	0.7608196491546576	2.7887321950852852
13	2.160410653092123	0.9037669606104153	0.9037742800688822
14	0.9035821326731487	-0.4896608822683287	1.5320953133146744
15	2.160354423989408	2.1603880366490262	1.5320944032126522
16	0.13089258673889237	3.4170423320938426	2.788729319809106
17	2.7884916947192195	0.9037497409674335	2.1604135225456753
18	1.5320572964273174	2.160393474721097	2.788731285306568
19	2.7887303862043913	2.160391952696722	2.1604120312262105
20	0.8745881333053249	0.9037531949467185	1.5320963268607
21	2.1597740611792418	2.16040585743221	2.1604149162815025
22	1.5315183702894102	2.7887320628666594	2.1604142294845445
23	3.416879401152432	0.9037528747844704	2.7887331292075146
24	0.139504890723134	2.7887313667511666	2.1604141909546803
25	2.7880879138598837	2.160403451745668	3.4170503544871917
26	1.5319172451676188	1.5320957951600296	2.7887254519855733
27	2.160394790459378	2.160412464667898	2.7887311933770826
28	3.416981709847669	-1.3150881057724608	0.9037741248823364
29	0.9037479211056643	1.5320819437119357	1.532095341682333

4 Analysis

We can see in the graphs with the highest values obtained, the highest point that was found is 3.4 and that despite the changing parameters, there was very little variation found between the results obtained. This is possibly due to the effect of the code and the limited step range. To obtain more consistent results that would find the highest point possible, it'd be needed to change the step range and the amount of repetitions by an unreasonable margin.

5 Conclusion

This experiment is very helpful when it comes to analysing a surface for it's peaks and valleys, however, in the case of functions that are oscillatory, the peaks and valleys are far too many to be able to reliably find all of them through this method, since the agents generally get stuck within one single peak that may or may not be the highest value.

References

[1] J.FeroxDeitas. *GitHub,P7*. URL: `https://github.com/FeroxDeitas/Simulacion-Nano/tree/main/Tareas/P7`.

[2] E. Schaeffer. *GitHub,LocalSearch*. URL: `https://github.com/satuelisa/Simulation/tree/master/LocalSearch`.