P9

edgardjfc

April 2022

1 Introduction

In this assignment we had the objective to perform a simulation of a group of particles that moved according to their charge and their mass, in 3 different experiments, one for mass, one for charge and one where both acted together. We then observed the statistical behaviour of the particles through graphs and statistical analysis.

The code was developed based on the original code given by our teacher E. Schaeffer [3] and leaning into the code written and developed by our classmate J. "FeroxDeitas" [2]

2 Code

The first part of the process is running the code designed to generate the particles properties, randomly assigning values to the different parameters of each particle. This is done running the code values.py in the github repository of E. "edgardjfc" [1].

Once this code is executed there will be a values.csv file that will feed information to P9_Charge.py, P9_Mass.py, P9_Both.py, P9_Statistical_Test.py and P9_Graphs.py. All of these files can be found withing the github repository of E. "edgardjfc" [1].

The three experiments (P9_Charge.py, P9_Mass.py and P9_Both.py) start the same way, by calling all the necessary libraries that will be used to perform the different mathematical equations and graphic libraries to draw the graphs.

For the first experiment of charge the different functions that will be called throughout the process of simulating the experiment. This following function for force describes the repulsion and attraction between particles depending on their charge, described as follows:

```
def fuerza(i, shared):
    p = shared.data
    n = shared.count
    pi = p.iloc[i]
    xi = pi.x
```

```
    yi = pi.y
    ci = pi.c
    fx, fy = 0, 0
    for j in range(n):
        pj = p.iloc[j]
        cj = pj.c
        dire = (-1)**(1 + (ci * cj < 0))
        dx = xi - pj.x
        dy = yi - pj.y
        factor = dire * fabs(ci - cj) / (sqrt(dx**2 + dy**2) + eps)
        fx -= dx * factor
        fy -= dy * factor
    return (fx, fy)
```

For an experiment where mass is the only factor the only way particles can interact is through attraction, therefore, the force function needs to change accordingly, in the following way:

```
G = 0.025
def fuerza(i, shared):
    p = shared.data
    n = shared.count
    pi = p.iloc[i]
    xi = pi.x
    yi = pi.y
    mi = pi.m
    fx, fy = 0, 0
    for j in range(n):
        pj = p.iloc[j]
        mj = pj.m
        dx = xi - pj.x
        dy = yi - pj.y
        factor = G * ((mi * mj) / (sqrt(dx**2 + dy**2) + eps))
        fx -= dx * factor
        fy -= dy * factor
    return (fx, fy)
```

In the following experiment, both the charge and mass will be taken into consideration, therefor the function that describes the force must be changed accordingly in this third experiment, where the changes are as follows:

```
G = 0.025
def fuerza(i, shared):
    p = shared.data
    n = shared.count
    pi = p.iloc[i]
    xi = pi.x
```

```
    yi = pi.y
    ci = pi.c
    mi = pi.m
    fx1, fy1 = 0, 0
    fx2, fy2 = 0, 0
    for j in range(n):
        pj = p.iloc[j]
        cj = pj.c
        mj = pj.m
        dire = (-1)**(1 + (ci * cj < 0))
        dx = xi - pj.x
        dy = yi - pj.y
        factor = dire * fabs(ci - cj) / (sqrt(dx**2 + dy**2) + eps)
        factor1 = G * ((mi * mj) / (sqrt(dx**2 + dy**2) + eps))
        fx1 = fx1 - dx * factor
        fy1 = fy1 - dy * factor
        fx2 = fx2 - dx * factor1
        fy2 = fy2 - dy * factor1

    fx = fx1 + fx2
    fy = fy1 + fy2
    return (fx, fy)
```

In all three of the different experiments the experiment runs for 60 steps and produces every step as an image, then yields a .csv file that has the statistical information of the experiment. All thre .csv files are fed into the programs P9_Statistical_Test.py and P9_Graphs.py where the statistical analysis test is run and where the graphs are produced.

3 Results

The different graphical simulations were run and uploaded to the repository of E. "edgardjfc" [1]. in the form of animated gifs.

The three experiments yielded a respective .csv file, which were run through the code of P9_Graphs.py which produced fig.1, fig.2 and fig.3. In these figures we can see the variation of velocity throughout the experiment. With this we can infer the behaviour of attraction and repulsion.

4 Analysis

Afterwards the code for statistical analysis was run, where an "anova" or analysis of variance was run. Where the significance of the statistical test was considered as smaller than 0.05. Through this method it was found that there was a statistical significance to the different results of each experiment.

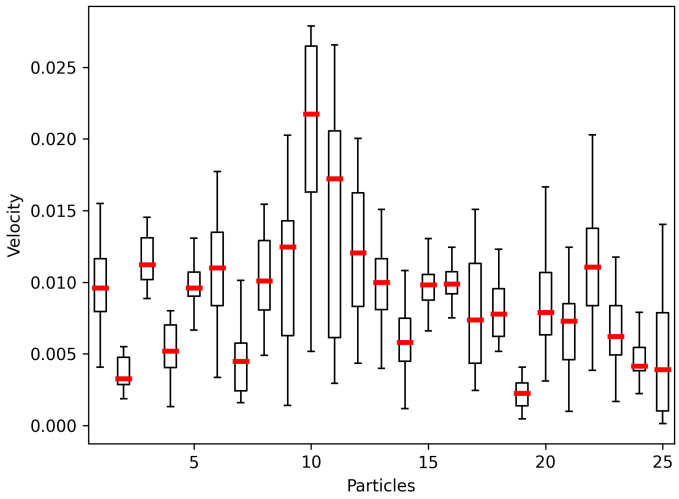

Figure 1: Behaviour with charge

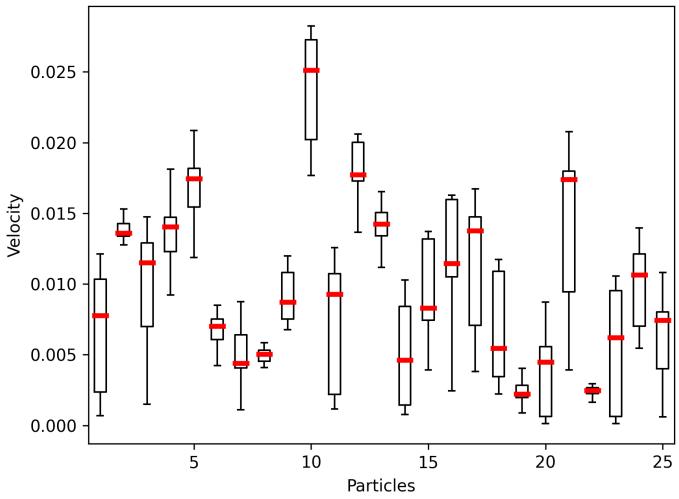

Figure 2: Behaviour with mass

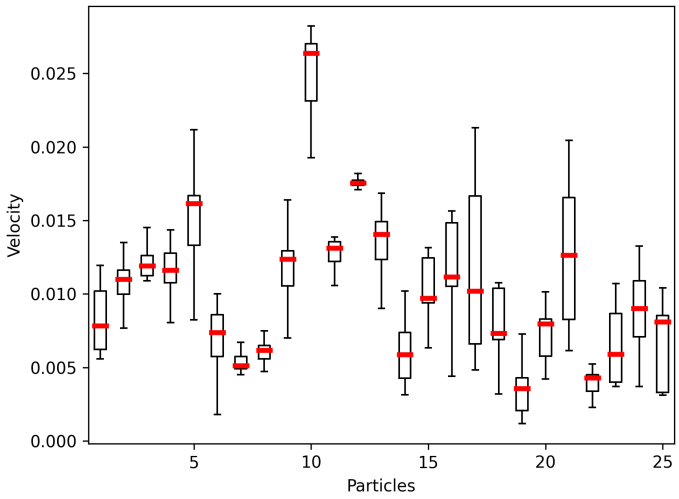

Figure 3: With both behaviours

5 Conclusion

After observing the behaviour of the particles in motion, the graph figures and the statistical analysis we can conclude that each different experiment had statistical variation between each other, where the forces of attraction and repulsion from the charges and gravity. The different forces affect directly the velocity in which the particles move, and we can confirm this by both seeing the diagrams and observing the result of the statistical analysis.

6 First challenge

For the first challenge of this assignment the objective is to simulate an experiment where large hard particles interact with smaller brittle particles, where the large particles break the smaller ones down when the impact is with only one small particle, or where multiple small particles are pressed together into one.

This simulation is very similar to the formation of mechanical alloys, where in a similar fashion, small particles are broken down and combined together by impacting with larger particles.

The code is broken down in different segments. First the initial particles and crushers are added to the space with their initial conditions, such as their positions in the x an y axis and their velocities, as follows:

```
m = 7
```

```
vx = (2 * (np.random.uniform(size = m) < 0.5) - 1) * np.random.uniform
    (low = 0.01, high = 0.04, size = m)
vy = (2 * (np.random.uniform(size = m) < 0.5) - 1) * np.random.uniform
    (low = 0.01, high = 0.04, size = m)
x = np.random.uniform(size = m)
y = np.random.uniform(size = m)
r = np.random.uniform(low = 0.05, high = 0.1, size = m)
bolas = pd.DataFrame({'x': x, 'y': y, 'dx': vx, 'dy': vy, 'r': r})
n = 50
vx = (2 * (np.random.uniform(size = n) < 0.5) - 1) * np.random.uniform
    (low = 0.02, high = 0.05, size = n)
vy = (2 * (np.random.uniform(size = n) < 0.5) - 1) * np.random.uniform
    (low = 0.02, high = 0.05, size = n)
x = np.random.uniform(size = n)
y = np.random.uniform(size = n)
print(x)
r = np.random.uniform(low = 0.01, high = 0.03, size = n)
particulas = pd.DataFrame
    ({'x': x, 'y': y, 'dx': vx, 'dy': vy, 'r': r, 'v': [1] * n, 'a': [1] * n})
```

Next is the functions that checks to see if there is an overlap between the particles and the crushers:

```
    for i in range(n):
        p = particulas.iloc[i]
        v = p.v

        if v > 0:
            pr = p.r
            px = p.x
            py = p.y
            conteo = 0
            for k in range(m):
                pk = bolas.iloc[k]
                pkr = pk.r
                pkx = pk.x
                pky = pk.y
                dx = px - pkx
                dy = py - pky
                dr = pkr + pr
                d = sqrt(dx**2 + dy**2)
                if d < dr:
                    conteo += 1
```

After this function we have another one in charge of checking if there's more than one particle overlapping between the crushers, if so, the particles get combined, updating their velocities and mass.

```
if conteo >= 2:
                conteo2 = 0
                for j in range(n):
                    if i != j:
                        pj = particulas.iloc[j]
                        pjr = pj.r
                        pjx = pj.x
                        pjy = pj.y
                        dx = px - pjx
                        dy = py - pjy
                        dr = pjr + pr
                        d = sqrt(dx**2 + dy**2)
                        if d < dr: # Unir particulas
                            conteo2 += 1
                            a1 = np.pi * (pr**2)
                            a2 = np.pi * (pjr**2)
                            a = a1 + a2
                            rt = sqrt(a/np.pi)
                            particulas.at[i, 'r'] = rt
                            particulas.at[j, 'v'] = -1
```

The next part of the code is in charge of diviing a particle if it is alone when overlapping the crushers:

```
 if conteo2 == 0: # Romper particulas
                    v = random()
                    v1 = 1 - v
                    vx1 = p.dx * -1
                    vy1 = p.dy * -1
                    a = np.pi * (pr**2)
                    a1 = a * v
                    a2 = a * v1
                    r1 = sqrt(a1/np.pi)
                    r2 = sqrt(a2/np.pi)
                    particulas.at[i, 'r'] = r1
                    particulas = particulas.append
                        ({'x': px, 'y': py, 'dx': vx1, 'dy':
                            vy1, 'r': r2, 'v': 1, 'a': 1}, ignore_index=True)
```

We can observe the results of the simulation by compiling all the .dat files into one gif and observing its behaviour. We can find this gif in the repository of E. "edgardjfc" [1].

References

[1] E. Edgardjfc. *GitHub,P9*. URL: https://github.com/edgardjfc/Simulacion-Nano-2022/tree/main/P9.

[2] J. FeroxDeitas. *GitHub,P9*. URL: https://github.com/FeroxDeitas/Simulacion-Nano/tree/main/Tareas/P9.

[3] E. Schaeffer. *GitHub,Particles*. URL: https://github.com/satuelisa/Simulation/tree/master/Particles.