Stochastic Simulation Lab #5 report

Shitong CHAI

Contents

Chapter 1

Theoretical analysis

Brownian motion is formally described by the Wiener process. To theoretically prove that the experiment settings of the Brownian agent is a Brownian motion, I will give the definition of Lévy characterisation at first.

Definition 1.0.1. *(Lévy characterisation of Wiener process[1]) Wiener process is an almost surely continuous martingale with $W_0 = 0$ and quadratic variation $[W_t, W_t] = Ct$, where C is a constant.*

In the experiment setting, I let the probability of the Brownian agent to move to the adjacent point (Including the current position of the agent) to be equal. Say the random variable describing the position of an agent at time point t is (X_t, Y_t), then we have

$$P(X_{t+1} - X_t = i) = \frac{1}{3},\ i \in \{-1, 0, 1\}$$
$$P(Y_{t+1} - Y_t = i) = \frac{1}{3},\ i \in \{-1, 0, 1\}$$

Then the quadratic variations[3] of the coordinates are

$$[X_t, X_t] = \sum_{s=0}^{t-1} ||X_{t+1} - X_t||^2 = \frac{2}{3}\sum_{s=0}^{t-1} 1 = \frac{2}{3}t$$
$$[Y_t, Y_t] = \sum_{s=0}^{t-1} ||Y_{t+1} - Y_t||^2 = \frac{2}{3}\sum_{s=0}^{t-1} 1 = \frac{2}{3}t$$

Which satisfies the definition of 2-dimension Wiener process.

Actually, the process determined in the experiment settings is a random walk because the movement of an agent is discrete. But according to Donsker Theorem[3], we have

$$\left(\frac{1}{\sigma\sqrt{n}}\left(\sum_{k=1}^{[nt]} U_k + (nt - [nt])U_{[nt]+1}\right)\right)_{0\le t\le 1} \underset{n\to\infty}{\Longrightarrow} (B_t)_{0\le t\le 1}$$

which means the random walk will converge to Brownian movement.

As the time increase, the random walk will be more and more like a Brownian movement, so the probability distribution[2] of the Brownian agent is

$$P(r) = \frac{2r}{N}e^{-\frac{r^2}{N}}$$

So we have the upper bound[2] of the resource foraged by the Brownian agent

$$L \le r^{\frac{4}{3}}.$$

which means the increasing speed of resource foraged by a Brownian agent will never be greater than the increasing speed of $r^{\frac{4}{3}}$, where r is the distance between the agent and its initial position.

Chapter 2

Experiment Setting and Result

I created an Agent class to provide basic attributes shared by all the agents and some basic methods and two other classes inheriting from it. The two classes Lawnmower and Brownian have different move method and have different type but share the same methods inherited from Agent class. The class diagram is shown in Figure 2.1. The

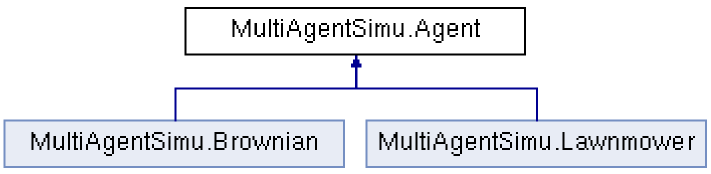

Figure 2.1: Class Diagram

movement of Lawnmower is line by line, so I increment the y coordinate of Lawnmower in its move method whenever there is a function call to it. And the movement of Brownian is random walking discussed in the first chapter.

Every Agent has an attribute *count* to record the number of resource foraged by it, and it will be collected at the end of the simulation.

The simulation of one time step is by calling the function one_step_simu, it will take care of all the forage and movements the agents.

The current status of the resource of every point is recorded in the variable space, all the agents will be stored in the variable agent_list.

The pseudo-random number generator (PRNG) used in the experiment is the default pseudo-random number generator in Python 3.7.4 64 bit build on win32, the Mersenne Twister generator. The documentation on the default random number generator, the Mersenne Twister generator is here: Documentation of random library from Python 3.7.4.

The seed in this experiment is initialized manually in the beginning and fixed throughout all the remaining experiments to avoid repeating patterns.

The space variable is created by the numerical computing library of Python, which is optimized by MKL compilers.

The first experiment is a small scale test on the code to debug. The space in this experiment is 10×10 and there are two agents, one Lawnmower and one Brownian. The Lawnmower is place at the upper left corner of the space in the beginning and the Brownian is placed at the point $x = 7, y = 7$. The maximum time span is 30.

With the experiment setting in the above, we have the resulting position of agents and remaining resources shown in the command line output in Figure 2.2.

```
.        .       .       .       .       .       .       .       .       .
.        .       .       .       .       .       .       .       .       .
.        .       .       .       .       .       .       .       .       .
L0       X       X       X       X       X       X       X       X       X
X        X       X       X       X       X       X       X       X       X
X        X       X       X       X       X       X       X       X       X
X        X       X       X       X       X       X       .       X       .
X        X       X       X       X       X       X       .       .       .
X        X       .       B1      X       .       .       .       .       X
X        .       X       X       .       .       X       .       .       .
```

Figure 2.2: Command line output

The second experiment is performed on a 100×100 2-dimensional point space. There are still two agents foraging for resources in this space. The Lawnmower agent is place on the upper left point of the space, which is the point $(0, 0)$ in the beginning. The second agent, the Brownian agent, is placed at the point $(30, 30)$ when the experiment begins, which is the time point 0.

The maximum time span of this experiment is 1000000, but since the total amount of available resource in the beginning is $100 \times 100 = 10000$, and there are two agents consuming the resources, this time span is not likely to be valid because the experiment will end when all the resources are consumed.

We are concerned about how much resources are consumed by each agent when all the resources are consumed. So I recorded the amount of resources consumed by each agent at any time point when the experiment hasn't stop. The resulting figure is shown in Figure 2.3.

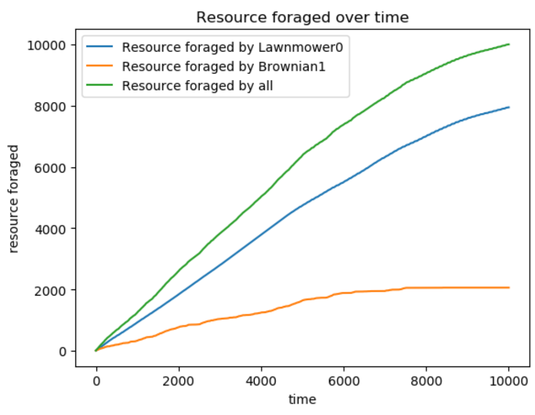

Figure 2.3: Resource consumed by each agent with respect to time

The third experiment is concerned about the confidence interval. All the experiment settings are the same in the second experiment but the random seed is not reset, so the experiment results will not repeat but have fluctuations.

I run 100 independent experiments with the same configurations as the second experiment and recorded all the resulting amount of resources consumed by each agent at time point 4000. With the independent experiments I have, I will compute a confidence interval with 95% confidence. So the confidence interval will become narrower and narrower with more and more independent experiments. The variations of confidence intervals are shown in Figure 2.4

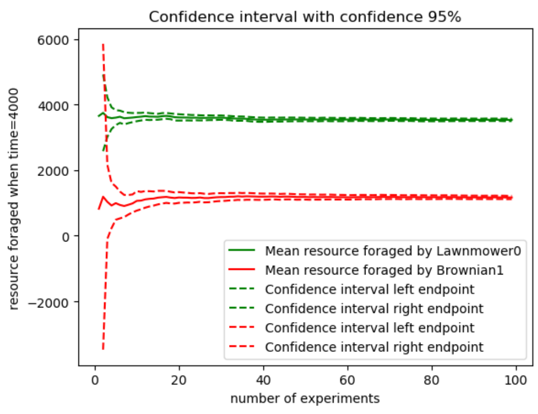

Figure 2.4: Confidence interval with respect to number of independent experiments

The last experiment is performed on a 1000×1000 2-dimensional point space. And the number of agents is different now. There are 4 agents. The first one, a Lawnmower, is placed randomly in the space in the beginning, instead of on the upper left corner. All the other three agents are Brownian agents, they are also placed randomly in the space in the beginning. With the bigger space size and different setting for agents, the results are a little different. See Figure 2.5.

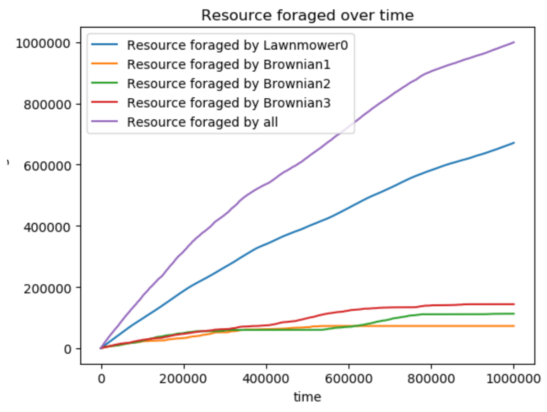

Figure 2.5: Resource foraged by each agent with respect to time

Notice that all the experiments are performed without reinitialization of pseudo-random number generator, which means the experiment results are not likely to repeat itself and be invalid.

Because of the inefficiency of Python interpreter, the agent simulation will become much slower than in C. But with Numpy as the numerical computing engine, the speed will not be a big problem.

Another problem with respect to Python is the Global Interpreter Lock(GIL). GIL is a interpreter level design to reduce parallel complexity and unsafe multi-thread. GIL will restrict Python programs to be run on only one thread, even if on a multi-core processor. This will hugely increase the speed of single-threaded programs but prevent programmer from exploiting the power of parallelism. However, if the numerical simulation problem can be vectorized and translated into linear algebra, the Numpy library will compute the result very quickly and in parallel with its C library. I believe this simulation problem can be translated into vector algebra problems but it will take a lot of time (even more than the time reduced by optimization), so I provide only the code without vectorization.

Chapter 3

Conclusion

As stated in Chapter 1, the resources foraged by a Brownian agent (if there is no other agent and the space is infinitely large) is propotional to $r^{4/3}$, where r is the average distance from the current position of Brownian agent to its initial position. However, this is not true in this simulation setting, because other agents will interfere with the Brownian agent and reduce the number of resource foraged and there is an upper bound of the distance between Brownian and its initial position. But if the time is limited, say less than 1000 (the total simulation time is 10000), and the average distance of Brownian agent from its initial position is less than the radius of the space, that is 50, then the theorem holds. This is supported by Figure 3.1

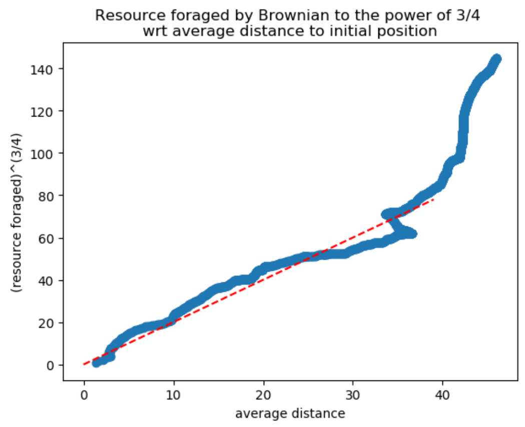

Figure 3.1: Resource foraged by Brownian agent (to the power of $\frac{3}{4}$) wrt average distance to initial position

where $L^{\frac{3}{4}} = 2r$ approximately holds for average distance less than 50 in the second experiment setting (100×100, 2 agents). This does not hold when the average distance approaches its limit 50, though.

If we study the general experiment result, we will find that the Lawnmower agent is almost always more competitive than Brownian agent when the size of space is small. When the size of space increases, the percentage of resources foraged by Brownian becomes stable around 20% with respect to all the resource foraged by the Brownian agent and one Lawnmower, which is shown in Figure 3.2.

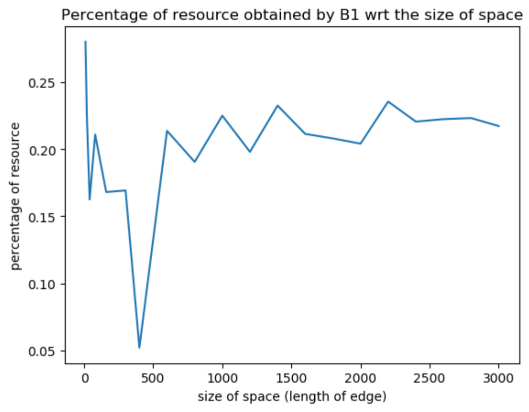

Figure 3.2: percentage of resource foraged by Brownian wrt the size of space

To be precise, the confidence interval of percentage foraged by Brownian when the size of space is larger than 600 is [20.8%, 22.3%] with 95% confidence.

All the comments in my Python code were written in the format required by Doxygen and the generated documentation has been uploaded to my website, this is the link: Documentation on MultiAgentSimu

Bibliography

[1] Wikipedia contributors. Brownian motion — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Brownian_motion&oldid=931386448`, 2019. [Online; accessed 5-January-2020].

[2] Wikipedia contributors. Random walk — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Random_walk&oldid=920819201`, 2019. [Online; accessed 5-January-2020].

[3] David Williams. *Probability with martingales.* Cambridge university press, 1991.