Summer 2019 Project 1: Martingale

From Quantitative Analysis Software Courses

Contents

Revisions

This assignment is subject to change up until 3 weeks prior to the due date. We do not anticipate changes; any changes will be logged in this section.

Overview

The purpose of this assignment is to get you started programming in Python right away and to help provide you some initial feel for risk, probability and "betting." Purchasing a stock is, after all, a bet that the stock will increase in value.

In this project you will evaluate the actual betting strategy that Professor Balch uses at roulette when he goes to Las Vegas. Here it is:

- episode_winnings = $0
- while episode_winnings < $80:
  - won = False
  - bet_amount = $1
  - while not won
    - wager bet_amount on black
    - won = result of roulette wheel spin
    - if won == True:
      - episode_winnings = episode_winnings + bet_amount
    - else:
      - episode_winnings = episode_winnings - bet_amount
      - bet_amount = bet_amount * 2

Here are some details regarding how roulette betting works: Betting on black (or red) is considered an "even money" bet. That means that if you bet N chips and win, you keep your N chips and you win another N chips. If you bet N chips and you lose then those N chips are lost. The odds of winning or losing depend on whether you're betting at an American wheel or a European wheel. For this project we will be assuming an **American wheel**. You can learn more about roulette and betting here: https://en.wikipedia.org/wiki/Roulette

Tasks

Set up your development environment First, if you haven't yet set up your software environment, follow the instructions here: ML4T_Software_Setup. The base directory structure is used for all projects in the class, including supporting data and software are will be set up correctly when you follow those instructions.

Get the template code for this project

This project is available here: File:19spring martingale.zip. Download and extract its contents into the base directory (ML4T_2019Spring). Once you've done this, you should see the following directory structure:

- `ML4T_2019Spring/`: Root directory for course
  - `data/`: Location of data
  - `grading/`: Grading libraries used by the individual grading scripts for each assignment.
  - `util.py`: Common utility library. This is the **only** allowed way to read in stock data. Only use the API methods provided here to read in stock data. Do NOT modify this file. For grading, we will use our own unmodified version.
  - README.md
  - `martingale/`: Root directory for this project
    - `martingale.py`: Main project file to use as a template for your code.

You should change only `martingale.py`. All of your code should be in that one file. Do not create additional files. It should always remain in and run from the directory `ML4T_2019Spring/martingale/`. Leave the copyright information at the top intact.

Insert your GT User ID and GT ID number

Revise the code functions `author()` and `gtid()` to correctly include your GT User ID and 9 digit GT ID respectively. Your GT User ID should be something like `tbalch78` and your GTID is a 9 digit number. You should also update this information the comments section at the top.

Build a simple gambling simulator

Revise the code in `martingale.py` to simulate 1000 successive bets on spins of the roulette wheel using the betting scheme outlined above. You should test for the results of the betting events by making successive calls to the `get_spin_result(win_prob)` function. Note that you'll have to update the `win_prob` parameter according to the correct probability of winning. You can figure that out by thinking about how roulette works (see wikipedia link above).

Track your winnings by storing them in a numpy array. You might call that array `winnings` where `winnings[0]` should be set to 0 (just before the first spin). `winnings[1]` should reflect the total winnings after the first spin and so on. For a particular episode if you ever hit $80 in winnings, stop betting and just fill the data forward with the value 80.

Experiment 1: Explore the strategy and make some charts

Now we want you to run some experiments to determine how well the betting strategy works. The approach we're going to take is called Monte Carlo simulation where the idea is to run a simulator over and over again with randomized inputs and to assess the results in aggregate. Skip to the "report" section below to which specific properties of the strategy we want you to evaluate.

For the following charts, and for all charts in this class you should use python's matplotlib library. Your submitted project should include all of the code necessary to generate the charts listed in your report. You should configure your code to write the figures to .png files. Do not allow your code to create a window that displays images. If it does you will receive a penalty.

- Figure 1: Run your simple simulator 10 times and track the winnings, starting from 0 each time. Plot all 10 runs on one chart using `matplotlib` functions. The horizontal (X) axis should range from 0 to 300, the vertical (Y) axis should range from -256 to +100. Note that we will not be surprised if some of the plot lines are not visible because they exceed the vertical or horizontal scales.
- Figure 2: Run your simple simulator 1000 times. Plot the mean value of winnings for each spin using the same axis bounds as Figure 1. Add an additional line above and below the mean at mean+standard deviation, and mean-standard deviation of the winnings at each point.
- Figure 3: Use the same data you used for Figure 2, but plot the median instead of the mean. Be sure to include the standard deviation lines above and below the median as well.

For all of the above charts and experiments, if and when the target $80 winnings is reached, stop betting and allow the $80 value to persist from spin to spin.

Experiment 2: A more realistic gambling simulator

You may have noticed that the strategy actually works pretty well, maybe better than you expected. One reason for this is that we were allowing the gambler to use an unlimited bank roll. In this experiment we're going to make things more realistic by giving the gambler a $256 bank roll. If he or she runs out of money, bzzt, that's it. Repeat the experiments above with this new condition. Note that once the player has lost all of their money (i.e., episode_winnings reaches -256) stop betting and fill that number (-256) forward. An important corner case to be sure you handle is the situation where the next bet should be $N, but you only have $M (where M<N). Make sure you only bet $M. Here are the two charts to create:

- Figure 4: Run your realistic simulator 1000 times. Plot the mean value of winnings for each spin using the same axis bounds as Figure 1. Add an additional line above and below the mean at mean+standard deviation, and mean-standard deviation of the winnings at each point.
- Figure 5: Use the same data you used for Figure 4, but use the median instead of the mean. Be sure to include the standard deviation lines above and below the median as well.

Contents of the report

Please address each of these points/questions in your report, to be submitted as `report.pdf`

1. In Experiment 1, estimate the probability of winning $80 within 1000 sequential bets. Explain your reasoning.
2. In Experiment 1, what is the estimated expected value of our winnings after 1000 sequential bets? Explain your reasoning. Go here to learn about expected value: https://en.wikipedia.org/wiki/Expected_value
3. In Experiment 1, does the standard deviation reach a maximum value then stabilize or converge as the number of sequential bets increases? Explain why it does (or does not).
4. In Experiment 2, estimate the probability of winning $80 within 1000 sequential bets. Explain your reasoning using the experiment. (not based on plots)
5. In Experiment 2, what is the estimated expected value of our winnings after 1000 sequential bets? Explain your reasoning. (not based on plots)
6. In Experiment 2, does the standard deviation reach a maximum value then stabilize or converge as the number of sequential bets increases? Explain why it does (or does not).
7. Include figures 1 through 5.

What to turn in

Submit the following files (only) via Canvas before the deadline:

- Your report as `report.pdf`
- Your code as `martingale.py`

Do not submit any other files. Note that your charts should be included in the report, not submitted as separate files. Also note that if we run your submitted code, it should generate all 5 figures as png files. Not submitting a report will result in a 0 for the assignment.

Rubric

Report

- Are the questions answered correctly? (Up to -5 points for each incorrect answer)
- Is the reasoning for each question correct and supported by the evidence? (Up to -5 points for each if incorrect)
- Are each of the charts provided and correct and include labeled axis and legend? (Up to -8 points for each if incorrect)

Code

- Does the code run without crashing? (-10 points if not)
- Does the code generate appropriate charts written to png files? (-10 points each up to a max of -20 if not)
- Does the implemented code reflect the project requirements? (Up to -10 points if not)

Required, Allowed & Prohibited

Required:

- Your project must be coded in Python 2.7.x.
- This requirement is not enforced for this first project, but will be for future projects. ~~Your code must run on one of the university-provided computers (e.g. buffet02.ee.gatech.edu).~~

Allowed:

- Your code may use standard Python libraries (except os).
- You may use the NumPy, SciPy, matplotlib and Pandas libraries. Be sure you are using the correct versions.
- Code provided by the instructor, or allowed by the instructor to be shared.

Prohibited:

- Any use of global variables.
- Any libraries not listed in the "allowed" section above.
- Use of Python's os module.
- Any code you did not write yourself (except for the 5 line rule in the "allowed" section).
- Knights who say "neeee."

Legacy versions