Harvard John A. Paulson School of Engineering and Applied Sciences

Michael P. Brenner and Chris Gumb

Homework 2

Random Walks: From Basketball scores to Stock prices

Issued:	September 30, 2025
Due:	October 17, 2025

Information

This assignment is an opportunity to apply the modeling and software engineering skills you've been developing. You may work in groups of up to three. We are confident in your ability to tackle these interesting problems.

Repository and Collaboration: For groups, one member's repository (named after their NetID) will serve as the central submission repo. This member must add their partners as collaborators. All group members should then clone and work from this central repository.

We strongly encourage you to use feature branches for individual problems or experiments (e.g., `hw2-problem1`) and merge them into a primary `hw2` branch as you complete them.

Submission: Your submission has two required components. First, you must open a pull request to merge your `hw2` branch into `main`; do not merge this PR until after grading. Second, one group member must upload a single `.zip` archive to Gradescope, adding all other members to the submission. The final problem of this assignment provides detailed instructions for creating this archive.

For a complete guide on the required Git workflow, please refer to the tutorial at https://harvard-am215.github.io/2025-public/pages/tutorials.html.

Problem 1: Basketball Scoring as a Random Walk

This problem explores modeling basketball scoring as a random walk process.

Write your code in the subdirectory `homework/hw2/submission/P1`. You should write a primary script, named `P1.py`, to generate the answers for all questions. All class and function definitions should be placed in a separate Python module(s), not in the script itself. This separation of logic from the main script is a key principle of good software design.

a) Using data from the 2014-15 NBA season, calculate the probability per unit time (15 seconds in this case, as the mean possession time in NBA games) of the *Golden State Warriors* team scoring $n = 0, 1, 2, 3, 4, \ldots$ points. Divide all games into windows of 15 seconds, then count how many points were scored by the GSW in each window, and calculate the probabilities for each possible scoring outcome based on your recorded data. The logic for this calculation should be encapsulated in a function within your helper module(s), which is then called by your main `P1.py` script. ***Hint:*** *Use the Python `nba_api` package demonstrated in the lecture 6 notebook to acquire the data.*

b) Using the computed probabilities, simulate a random walk representing the score progression of the team in a basketball game. Generate a sufficient number of simulations to produce a representative score distribution. Plot the score distribution after 12, 24, 36, and 48 minutes of play, save it as `P1b.png`. ***Hint:*** *A few hundred simulations should be sufficient to generate a smooth distribution.*

c) Derive the associated continuum equation for this process. Your answer should be in the form of a differential equation. Save your derivation as `P1c.pdf`.

Hint: Follow the derivation of the diffusion equation explained in class. Your starting point should have the form:

$$P(s, t+1) = \sum_{n=0}^{N} p_n P(s-n, t) \tag{1}$$

where $P(s,t)$ is the probability of having score s at time t, and p_n is the probability of scoring n points in one time step (with $n = 0, 1, 2, 3, 4, ...$ representing points scored by the team). $N = \max(n)$ will be the highest score you registered in 1a.

d) Using your derived equation, calculate and plot the expected score distribution after 12, 24, 36, and 48 minutes of play. Assume both teams start at 0 points, that is $s(t = 0) = \delta(s)$. Save the resulting plot as `P1d.png`.

e) Compare the theoretical distributions to the ones you generated in 1b and answer the following:

i) How does the variance of the score difference grow with time? Is this consistent with a standard random walk? Write your answers into `P1e_i.txt`.

ii) What are the limitations of this model? What aspects of real basketball games are not captured by this simple random walk approach? Write your answers into `P1e_ii.txt`.

f) Take a screenshot of the graphical commit history after finishing this problem and save as `P1f.png`. Your screenshot should clearly show the history of your `hw2` branch, including any feature branches you merged into it. You can generate this view using a command like:

```
$ git log --oneline --graph --all
```

Extra Credit Problem: *ELO Scores and Random Walks*

This problem explores the connection between ELO scoring systems and random walk models in sports. Consider two teams, A and B, whose performance in a match can be modeled as independent random walks. Let μ_A and μ_B be the mean performance of teams A and B respectively, and σ_A^2 and σ_B^2 their variances.

Place all files for this problem in the `homework/hw2/submission/ExtraCredit` directory. The deliverables are:

- A PDF document named `ExtraCredit.pdf` containing all written answers, derivations, and discussion. This document should also include the plots from parts (4) and (6).
- A Python script named `make_plots.py` that generates the required plots. You may use helper modules if you wish.
- The plot images, saved as `EC_plot1.png` and `EC_plot2.png`.

1. Show that the distribution of the score difference $\Delta s = s_B - s_A$ follows a random walk with mean $\tilde{\mu} = \mu_B - \mu_A$ and variance $\tilde{\sigma} = \sigma_A^2 + \sigma_B^2$.
2. A basketball match runs for a specified time T. Argue about why the distribution of Δs at the end of the match follows a Gaussian distribution. What are its mean and variance?
3. The probability of team A winning can be expressed as $P(A \text{ wins}) = P(\Delta s < 0)$. Using results from the previous questions, express this probability in terms of the cumulative distribution function of a standard normal distribution.
4. Plot this probability as a function of $\tilde{\mu} = \mu_B - \mu_A$, assuming $\sigma_A^2 = \sigma_B^2 = 1$. What shape does this curve take?
5. Recall that in the ELO rating system, the probability of team A (with rating R_A) winning against team B (with rating R_B) is given by:

$$P(A \text{ wins}) = \frac{1}{1 + 10^{(R_B - R_A)/400}} \tag{2}$$

   Compare this to your plot from part (4). What is the relationship between the ELO rating difference and the mean performance difference in the random walk model?
6. Now consider the case where the random walks of the two teams have standard deviations $\sigma_A = 1$ and $\sigma_B = 3$. Plot the probability of team A winning as a function of $\tilde{\mu} = \mu_B - \mu_A$. How does this curve differ from the one in part (4)? And from the ELO score formula in part (5)? ***Hint:*** *use the analytic formula based on the cumulative standard normal distribution you worked out before.*
7. Under what conditions would the ELO system and the random walk model give approximately equivalent results? Discuss any assumptions required.

Problem 2: Geometric Brownian Motion

We discussed the geometric Brownian motion as a model for stock prices.

For this problem, write your code in the subdirectory `homework/hw2/submission/P2`. Use as many branches as you want.

a) Pick your favorite stock (or if you don't have a favorite stock, choose a random stock) and import it using the Yahoo Finance Python API (introduced in the lecture notebook). Use it to make a plot of the stock price variation over the time period 2020-2023. Define a plotting function that accepts the stock data. Create and apply a decorator to this function to handle plot aesthetics, such as adding a title and axis labels. Save the script as `P2_a.py` and the plots as `P2_a.png`.

b) Geometric Brownian motion has two parameters, μ and σ. Find the values of these parameters that are consistent with the stock price variation over the given time period. Encapsulate the code into a class. The `fit` method should estimate the parameters μ and σ from the data. The `predict` method should use these parameters to generate a new simulated price path over a specified time horizon. Inside the same script, define a `main` function where you create train/test splits from your data, and demonstrate the usage of your class. The script must print the estimated values to the console. Save it as `P2_b.py`.

c) In a new script, `P2_c.py`, import the class you developed in the previous step. Use it to simulate 100 instances of geometric Brownian motion for this stock. Decorate another plot function to plot all simulations on the same plot, with a small value of alpha for transparency, and save the plot as `P2_c.png`. Are your plots consistent with the changes observed in the stock price? Write your answers in `P2_c.txt`.

d) Package the geometric Brownian motion simulator you developed in the previous parts into a Python package. Your package should be structured according to modern Python packaging standards (e.g., using a `pyproject.toml` file). Upload your package to the TestPyPI server. As confirmation, take a screenshot of your package's project page on TestPyPI and save it as `P2_d.png`. ***Hint:*** *Ensure your package has a unique name on TestPyPI, for example by including your NetID.*

Problem 3: Kaggle Competition: Portfolio Optimization

The data for this problem, consisting of prices for 10 stocks from 2010-2022, as well as an example submission file, is provided on the competition website. Carry out a Markowitz-style portfolio optimization to find the optimal portfolio going forward. Your answer must be a vector denoting the proportion of each stock. Upload your weights as your entry to the Kaggle competition website (Note: The final link will be announced on Canvas).

Please be careful when formulating your entry. We have generated these stock prices synthetically, and this is **not** as straightforward as the initial lecture notebook exercise. Our advice is to carry out a train/dev split of the data to validate your results yourself before submitting to the competition.

Place all code for this problem in the `homework/hw2/submission/P3` subdirectory. While the specific file structure is flexible, your solution must demonstrate best practices discussed in class, such as object-oriented programming, the Python data model, and the use of submodules where appropriate.

Data Handling: Download the data from the Kaggle competition page. Create a directory named `data` inside `homework/hw2/submission/P3` and place the downloaded files there. Your `.gitignore` file should be configured to exclude this `data` directory. Your `docker_run.sh` script should then mount this local data directory into the container so that your Python code can access it (e.g., using the `-v` flag with `docker run`).

To ensure reproducibility, your submission must include:

- A script named `docker_build.sh` to build a Docker container with all necessary dependencies.
- A script named `docker_run.sh` that executes the code within the container to generate the portfolio weights for your Kaggle submission.

The Python code executed by your Docker container should generate a submission file named `submission.csv` containing the final weights vector. Remember to commit frequently and document your code. Use as many branches as you want, as they will help you track different models and experiments.

Problem 4: Submission Packaging

To ensure a clean and consistent submission, you will create a reusable shell script that packages all your deliverables into a single `.zip` archive for Gradescope.

a) First, create the following two text files and place them directly inside your `homework/hw2/submissio` directory:

- `collaborators.txt`: List the NetID of each group member, one per line.
- `pr_link.txt`: Paste the full URL to your group's pull request for this homework.

b) Next, create a directory named `scripts` in the root of your repository if it does not already exist. Inside this directory, create a shell script named `create_submission.sh`. This script will be designed to be reusable for future assignments.

When run from the root of your repository, the script must:

1. Accept a single command-line argument, which is the name of the homework directory (e.g., `hw2`).
2. Begin with a robust shebang, such as `#!/usr/bin/env bash`.
3. Using the argument, determine the path to the source directory (e.g., `homework/hw2/submission`) and the desired output path for the archive (e.g., `homework/hw2/submission.zip`).
4. Create a zip archive at the specified output path. **When unzipped, this archive must produce a single directory named** `submission` containing all your deliverable files.
5. The zip process must explicitly exclude common temporary files and directories, such as `__pycache__`, `*.pyc`, and `.DS_Store`.

The final `submission.zip` file is what you will upload to Gradescope. This generated archive should not be committed to your repository; consider adding `*.zip` to your `.gitignore` file. ***Hint:*** *The `zip` command is useful here. To create an archive with the correct internal structure, a common strategy is for the script to first change into the directory containing the `submission` folder (e.g., `homework/hw2`) before running the `zip` command. This ensures the resulting archive contains a single top-level `submission` directory. Your script must be runnable from the repository root, like so: `./scripts/create_submission.sh hw2`.*