Simulation Challenge

Generative Models and Monte Carlo Simulation

Simulation Challenge - Monte Carlo Analysis

Challenge Overview

Your Mission (along with your Cursor AI partner): Create a comprehensive Quarto document that simulates an investment strategy (or possibly two strategies), analyzes the simulation results, and presents your ability to simulate a scenario and present counter-intuitve results in a compelling way.

The Investment Game

Original Game Strategy

Example 0.1. Imagine you are offered the following game and given a $1,000 budget in a special account to play the game: I will flip a coin, and if it comes up heads, we increase your account's balance by 50%; if it comes up tails, we reduce your account's balance by 40%. We are not only doing this once, but we will do it once per year until you turn 55. When you turn 55, you will receive the balance in your account.

Generative DAG Model for the Investment Game

Challenge Requirements

Minimum Requirements for Any Points on Challenge

1. **Create a Quarto Document:** Write a concise quarto markdown file that includes a narrative of what you are doing along with the requested code, results, and visualizations of your simulations.
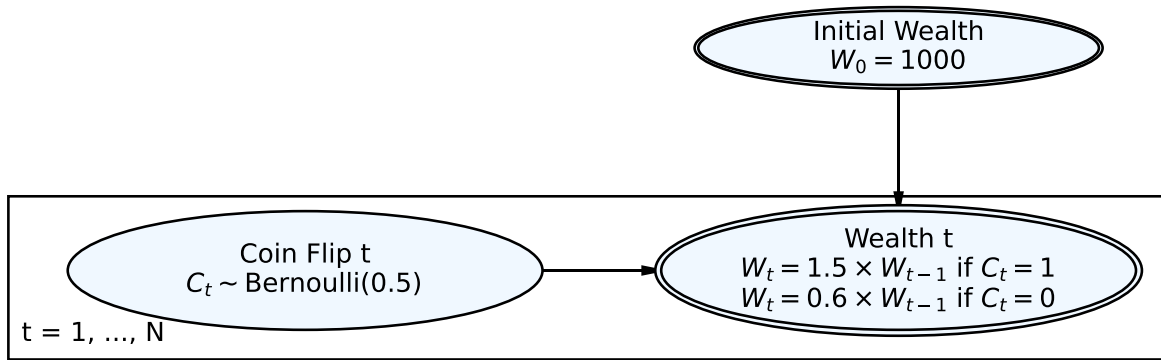
Figure 1: Generative DAG model for the investment game showing how wealth evolves over time through coin flips

2. **Render to HTML:** You must render the quarto markdown file to HTML.

3. **GitHub Repository:** The rendered HTML must be uploaded to a new GitHub repository called "simulationChallenge" in your Github account.

4. **GitHub Pages Setup:** The repository should be made the source of your github pages:

   - Go to your repository settings (click the "Settings" tab in your GitHub repository)
   - Scroll down to the "Pages" section in the left sidebar
   - Under "Source", select "Deploy from a branch"
   - Choose "main" branch and "/ (root)" folder
   - Click "Save"
   - Your site will be available at: `https://[your-username].github.io/simulationChallenge/`
   - **Note:** It may take a few minutes for the site to become available after enabling Pages

Grading Rubric

Questions to Answer for 75% Grade on Challenge

1. **Expected Value Analysis:** What is the "expected value" of your account balance after 1 coin flip for the original game?

2. **Expectation vs. Reality:** Is the expected value positive or negative? Do you expect your account to be worth more or less than $1,000 based on this result?

3. **Single Simulation:** Run one simulation showing the dynamics of your account balance over time. Make an object-oriented matplotlib OR ggplot2 plot showing your simulated account balance over time (i.e. as you age). Comment on the results, are you happy?

Questions to Answer for 85% Grade on Challenge

4. **Multiple Simulations:** Run 100 simulations showing the dynamics of your account balance over time. Make an object-oriented matplotlib OR ggplot2 plot showing a probability distribution of account balance at age 55. Comment on the results, are you happy? Why or why not?

Questions to Answer for 95% Grade on Challenge

5. **Probability Analysis:** Based on the 100 simulations above, what is the probability that your account balance will be greater than $1,000 at age 55?

Questions to Answer for 100% Grade on Challenge

6. **Strategy Comparison:** Run 100 simulations for the modified game strategy shown below in Example 0.2. What is the probability that your account balance will be greater than $10,000 at age 55? Is this probability higher or lower than the probability in the original game?

Modified Game Strategy

Example 0.2. Imagine you are offered the following game and given a $1,000 budget in a special account to play the game: I will flip a coin, and if it comes up heads, we increase your bet by 50%; if it comes up tails, we reduce your bet by 40%. You must bet exactly 50% of your current account balance on each flip, and this 50% is locked in for each round. We are not only doing this once, but we will do it once per year until you turn 55. When you turn 55, you will receive the balance in your account.

Technical Implementation Preferences

Setting Up Your Analysis

For R Users:

- Use `tidyverse` for data manipulation
- Use `ggplot2` for visualizations
- Use `set.seed()` for reproducible results

For Python Users:

- Use `numpy` for numerical operations

- Use `pandas` for data manipulation
- Use `matplotlib` (object-oriented)
- Use `np.random.seed()` for reproducible results

Visualization Preferences

- **Professional Styling:** Use consistent colors, clear labels, readable fonts, and informative titles

Submission Checklist

Minimum Requirements (Required for Any Points):

☐ Quarto document created with clear narrative
☐ Document rendered to HTML successfully
☐ Repository "simulationChallenge" created
☐ HTML files uploaded to repository
☐ GitHub Pages enabled and working
☐ Site accessible at `https://[your-username].github.io/simulationChallenge/`

75% Grade Requirements:

☐ Expected value calculations shown (Question 1)
☐ Expectation vs. reality analysis (Question 2)
☐ Single simulation with time series plot (Question 3)
☐ Clear interpretation of single simulation results

85% Grade Requirements:

☐ 100 simulations with distribution analysis (Question 4)
☐ Probability distribution plot of final account balances
☐ Clear interpretation of multiple simulation results

95% Grade Requirements:

☐ Probability calculations for original strategy (Question 5)
☐ Analysis of probability that balance > $1,000 at age 55

100% Grade Requirements:

☐ 100 simulations for modified strategy (Question 6)
☐ Probability calculations for modified strategy
☐ Comparative analysis between both strategies
☐ Analysis of probability that balance > $10,000 at age 55

Code Quality (All Grades):

☐ Reproducible results (seeds set)
☐ Clean, well-commented code
☐ Appropriate use of functions and loops
☐ Professional visualization styling

Resources

- **Quarto Markdown:** quarto.org/docs/authoring/markdown-basics.html
- **Quarto Documentation:** quarto.org/docs
- **R for Data Science:** r4ds.had.co.nz
- **Python Data Science Handbook:** jakevdp.github.io/PythonDataScienceHandbook

Getting Started Tips

> **ℹ** Navy SEALs Motto
>
> "Slow is Smooth and Smooth is Fast"
>
> *Take your time to understand the simulation mechanics, plan your approach carefully, and execute with precision. Rushing through this challenge will only lead to errors and confusion.*

- **Browse Essential Simulation Concepts:** This section will give you a good understanding of the concepts you need to know to complete the challenge.
- **Start Simple:** Begin with a single simulation to understand the mechanics
- **Document Everything:** Explain your reasoning and interpret your results
- **Forgetting to Set Seeds:** Always set random seeds for reproducible results
- **Total time to complete:** ~3-4 hours for the 100% grade
- **Good luck, and remember simulation will steer you right even when intuition will steer you wrong!**

> **⚠** Important: Save Your Work Frequently!
>
> **Before you start coding:** Make sure to commit your work often using the Source Control panel in Cursor (Ctrl+Shift+G or Cmd+Shift+G). This prevents the AI from overwriting your progress and ensures you don't lose your work.
> **Commit after each major step:**
>
> - After completing each simulation example

- After finishing each challenge question
- Before asking the AI for help with new code

How to commit:

1. Open Source Control panel (Ctrl+Shift+G)
2. Stage your changes (+ button)
3. Write a descriptive commit message
4. Click the checkmark to commit

Remember: Frequent commits are your safety net!

Essential Simulation Concepts

Before diving into the challenge, let's review the key simulation concepts you'll need. These examples will prepare you for the investment game analysis.

1. Simple Simulation: Coin Flip Game

Let's start with a basic coin flip simulation to understand the mechanics:

R

```r
suppressPackageStartupMessages(library(tidyverse))

# Set seed for reproducibility
set.seed(123)

# Number of simulations
n_sims <- 10

# Create simulation data
sim_data <- tibble(
  sim_num = 1:n_sims,
  coin_flip = rbinom(n = n_sims, size = 1, prob = 0.5)
) %>%
  mutate(
    winnings = if_else(coin_flip == 1, 100, -100)
  )
```

```
# Display results
sim_data
```

```
# A tibble: 10 x 3
   sim_num coin_flip winnings
     <int>     <int>    <dbl>
 1       1         0     -100
 2       2         1      100
 3       3         0     -100
 4       4         1      100
 5       5         1      100
 6       6         0     -100
 7       7         1      100
 8       8         1      100
 9       9         1      100
10      10         0     -100
```

Python

```python
import numpy as np
import pandas as pd

# Set seed for reproducibility
np.random.seed(123)

# Number of simulations
n_sims = 10

# Step 1: Draw coin flips (stochastic node)
X = np.random.binomial(n=1, p=0.5, size=n_sims)

# Step 2: Compute winnings (deterministic node)
W = np.where(X == 1, 100, -100)

# Combine into data frame
sim_data = pd.DataFrame({
    'sim_num': range(1, n_sims + 1),
    'coin_flip': X,
    'winnings': W
})
```

```
# Display results
sim_data
```

```
   sim_num  coin_flip  winnings
0        1          1       100
1        2          0      -100
2        3          0      -100
3        4          1       100
4        5          1       100
5        6          0      -100
6        7          1       100
7        8          1       100
8        9          0      -100
9       10          0      -100
```

2. Time-Series Simulation: Account Balance Over Time

Now let's simulate how an account balance changes over multiple periods:

R

```r
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(ggplot2))

# Set seed for reproducibility
set.seed(456)

# Parameters
initial_balance <- 1000
n_periods <- 10
n_sims <- 1   # Start with one simulation

# Simulate one path
simulate_path <- function(initial, periods) {
  balance <- initial
  path <- numeric(periods + 1)
  path[1] <- initial

  for (i in 1:periods) {
```

```r
    coin_flip <- rbinom(1, 1, 0.5)
    if (coin_flip == 1) {
      balance <- balance + 100
    } else {
      balance <- balance - 100
    }
    path[i + 1] <- balance
  }

  return(path)
}

# Run simulation
time_series_data <- tibble(
  period = 0:n_periods,
  balance = simulate_path(initial_balance, n_periods)
)

# Create time series plot
ggplot(time_series_data, aes(x = period, y = balance)) +
  geom_line(color = "cadetblue", linewidth = 1.2) +
  geom_point(color = "darkblue", size = 2) +
  labs(
    title = "Account Balance Over Time",
    subtitle = "Single Simulation Path",
    x = "Period",
    y = "Account Balance ($)",
    caption = "Heads: +$100$, Tails: -$100$"
  ) +
  scale_y_continuous(labels = scales::dollar_format()) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12, color = "gray50")
  )
```
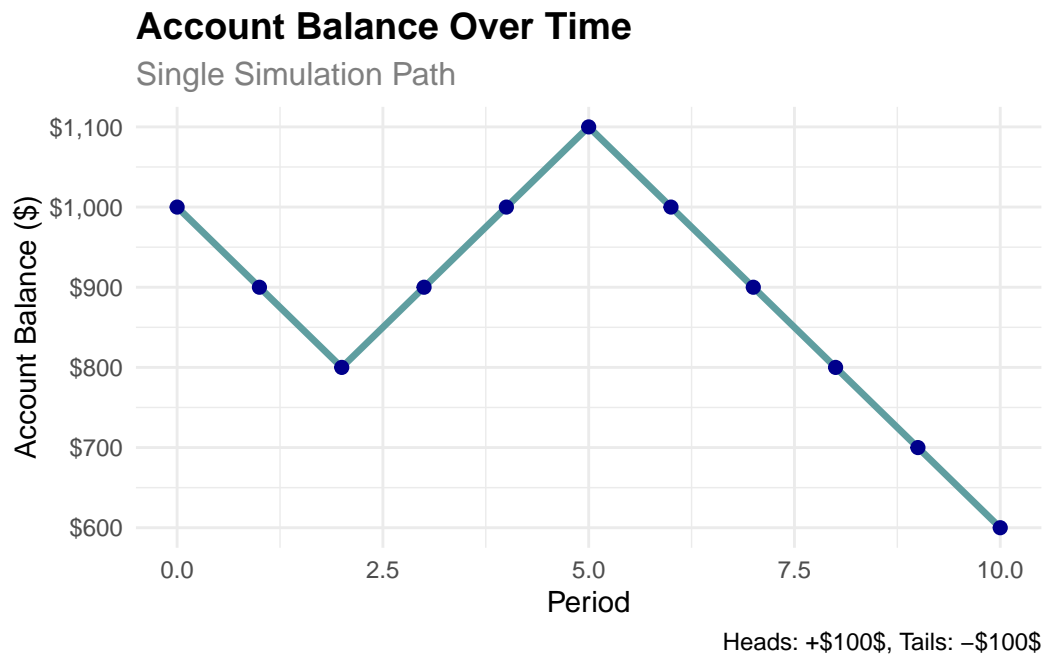
Figure 2: R time-series simulation of account balance

```
# Show the data
time_series_data
```

```
# A tibble: 11 x 2
   period balance
    <int>   <dbl>
 1      0    1000
 2      1     900
 3      2     800
 4      3     900
 5      4    1000
 6      5    1100
 7      6    1000
 8      7     900
 9      8     800
10      9     700
11     10     600
```

Python

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(456)

# Parameters
initial_balance = 1000
n_periods = 10
n_sims = 1  # Start with one simulation

# Simulate one path
def simulate_path(initial, periods):
    balance = initial
    path = [initial]

    for i in range(periods):
        coin_flip = np.random.binomial(1, 0.5)
        if coin_flip == 1:
            balance = balance + 100  # $100 gain
        else:
            balance = balance - 100  # $100 loss
        path.append(balance)

    return path

# Run simulation
time_series_data = pd.DataFrame({
    'period': range(n_periods + 1),
    'balance': simulate_path(initial_balance, n_periods)
})

# Create time series plot
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(time_series_data['period'], time_series_data['balance'],
        color='cadetblue', linewidth=2, marker='o', markersize=6)
ax.set_title('Account Balance Over Time\nSingle Simulation Path',
             fontsize=14, fontweight='bold')
ax.set_xlabel('Period', fontsize=12)
```

11

```
ax.set_ylabel('Account Balance ($)', fontsize=12)
ax.grid(True, alpha=0.3)
ax.set_ylim(0, max(time_series_data['balance']) * 1.1)
```

(0.0, 1540.0000000000002)

```
# Format y-axis as currency
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x:,.0f}'))

plt.tight_layout()
plt.show()
```
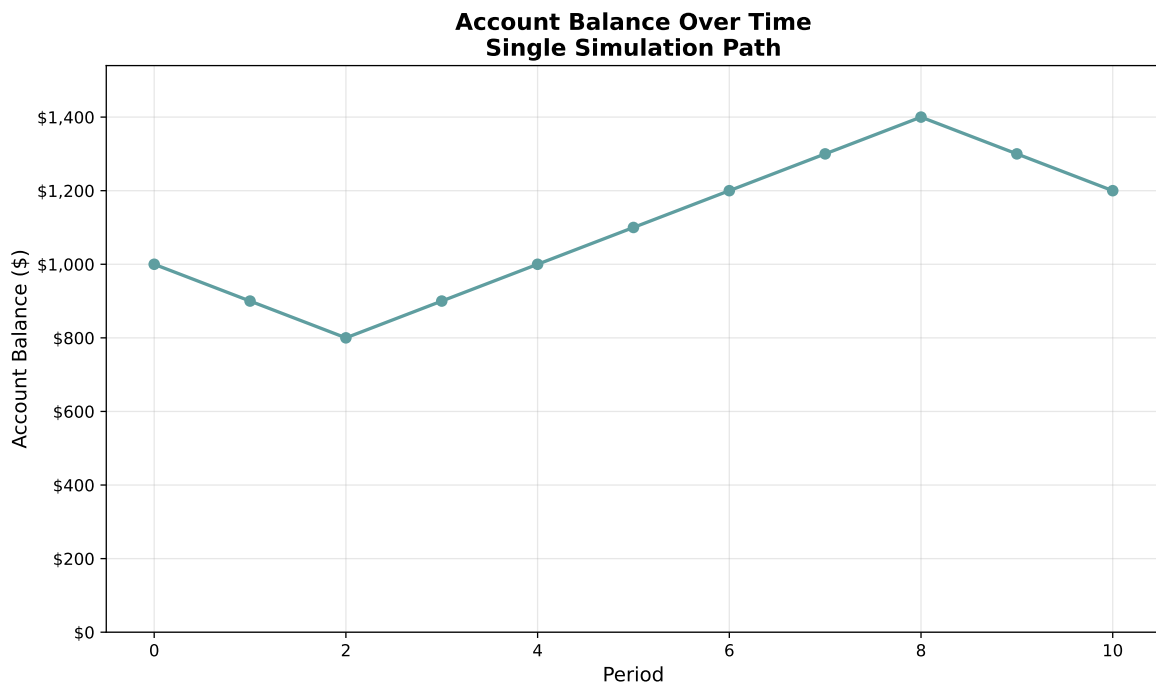

Figure 3: Python time-series simulation of account balance

```
# Show the data
print("Time Series Data:")
```

Time Series Data:

```
print(time_series_data)
```

```
     period  balance
0         0     1000
1         1      900
2         2      800
3         3      900
4         4     1000
5         5     1100
6         6     1200
7         7     1300
8         8     1400
9         9     1300
10       10     1200
```

3. Probability Distribution: Final Balance Distribution

Let's see what the distribution of final balances looks like across many simulations:

R

```
suppressPackageStartupMessages(library(tidyverse))
suppressPackageStartupMessages(library(ggplot2))

# Set seed for reproducibility
set.seed(789)

# Parameters
initial_balance <- 1000
n_periods <- 10
n_sims <- 100  # Multiple simulations

# Simulate multiple paths
simulate_final_balance <- function(initial, periods) {
  balance <- initial
  for (i in 1:periods) {
    coin_flip <- rbinom(1, 1, 0.5)
    if (coin_flip == 1) {
      balance <- balance + 100  # $100 gain
    } else {
```

```r
      balance <- balance - 100  # $100 loss
    }
  }
  return(balance)
}

# Run multiple simulations
final_balances <- replicate(n_sims, simulate_final_balance(initial_balance, n_periods))

# Create data frame
distribution_data <- tibble(
  sim_num = 1:n_sims,
  final_balance = final_balances
)

# Create histogram
ggplot(distribution_data, aes(x = final_balance)) +
  geom_histogram(bins = 20, fill = "plum", alpha = 0.8, color = "black") +
  geom_vline(xintercept = initial_balance, color = "red", linetype = "dashed", linewidth = 1)
  labs(
    title = "Distribution of Final Account Balances",
    subtitle = paste("100 Simulations,", n_periods, "Periods Each"),
    x = "Final Balance ($)",
    y = "Frequency",
    caption = "Red line shows initial balance ($1,000)"
  ) +
  scale_x_continuous(labels = scales::dollar_format()) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12, color = "gray50")
  )
```

Distribution of Final Account Balances

100 Simulations, 10 Periods Each

Red line shows initial balance ($1,000)
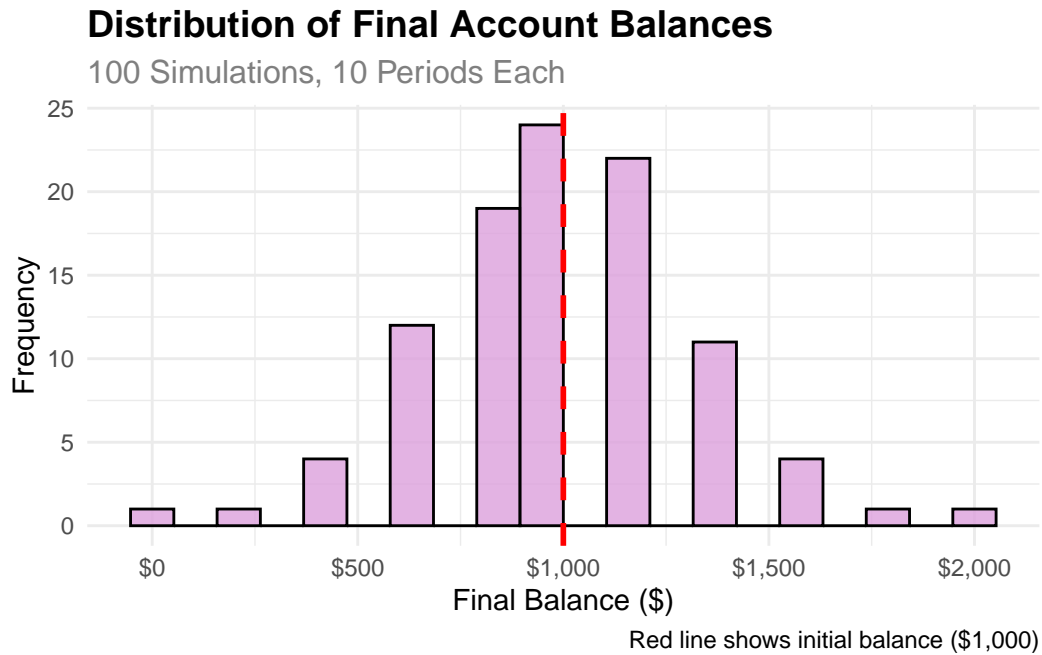
Figure 4: R probability distribution of final balances

```
# Summary statistics
summary_stats <- distribution_data %>%
  summarise(
    mean_balance = mean(final_balance),
    median_balance = median(final_balance),
    prob_above_initial = mean(final_balance > initial_balance)
  )

print("Summary Statistics:")
```

```
[1] "Summary Statistics:"
```

```
summary_stats
```

```
# A tibble: 1 x 3
  mean_balance median_balance prob_above_initial
         <dbl>          <dbl>              <dbl>
1         1002           1000               0.39
```

Python

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(789)

# Parameters
initial_balance = 1000
n_periods = 10
n_sims = 100  # Multiple simulations

# Simulate multiple paths
def simulate_final_balance(initial, periods):
    balance = initial
    for i in range(periods):
        coin_flip = np.random.binomial(1, 0.5)
        if coin_flip == 1:
            balance = balance + 100  # $100 gain
        else:
            balance = balance - 100  # $100 loss
    return balance

# Run multiple simulations
final_balances = [simulate_final_balance(initial_balance, n_periods) for _ in range(n_sims)]

# Create data frame
distribution_data = pd.DataFrame({
    'sim_num': range(1, n_sims + 1),
    'final_balance': final_balances
})

# Create histogram
fig, ax = plt.subplots(figsize=(10, 6))
ax.hist(distribution_data['final_balance'], bins=20, color='plum', alpha=0.8, edgecolor='bla
ax.axvline(initial_balance, color='red', linestyle='--', linewidth=2, label='Initial Balance
ax.set_title(f'Distribution of Final Account Balances\n100 Simulations, {n_periods} Periods
              fontsize=14, fontweight='bold')
ax.set_xlabel('Final Balance ($)', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
```

```
ax.legend()
ax.grid(True, alpha=0.3)

# Format x-axis as currency
ax.xaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x:,.0f}'))

plt.tight_layout()
plt.show()
```
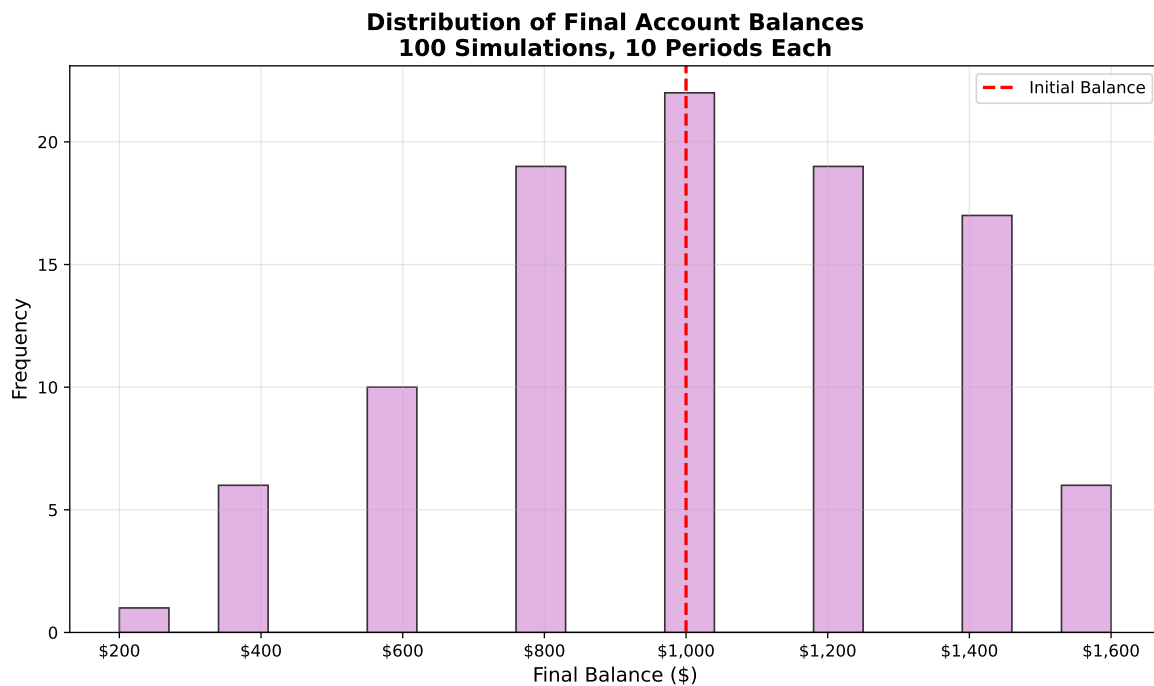

Figure 5: Python probability distribution of final balances

```
# Summary statistics
mean_balance = distribution_data['final_balance'].mean()
median_balance = distribution_data['final_balance'].median()
prob_above_initial = (distribution_data['final_balance'] > initial_balance).mean()

print("Summary Statistics:")
```

```
Summary Statistics:
```

```python
print(f"Mean balance: ${mean_balance:,.2f}")
```

Mean balance: $1,020.00

```python
print(f"Median balance: ${median_balance:,.2f}")
```

Median balance: $1,000.00

```python
print(f"Probability above initial: {prob_above_initial:.3f}")
```

Probability above initial: 0.420