ps3

March 6, 2023

1. Helen Xiao, Alex Cai, Iñaki Arango
2. Total: 9 (Theory: 6, Computational: 3)
3. https://michaelbigelow.com/post/polya-urn-simulation/
4. I have not shared my code with anyone and have not used anyone else's code.

[54]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import yfinance as yfin
import math
import matplotlib_inline.backend_inline
import statsmodels.api as sm
from patsy import dmatrices
import random

matplotlib_inline.backend_inline.set_matplotlib_formats('pdf', 'png')

plt.rcParams['savefig.dpi'] = 75

plt.rcParams['figure.autolayout'] = False
plt.rcParams['figure.figsize'] = 10, 6
plt.rcParams['axes.labelsize'] = 18
plt.rcParams['axes.titlesize'] = 20
plt.rcParams['font.size'] = 16
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['lines.markersize'] = 8
plt.rcParams['legend.fontsize'] = 14

plt.rcParams['text.usetex'] = True
plt.rcParams['font.family'] = "serif"
plt.rcParams['font.serif'] = "cm"
```

[55]:
```
# 1.1 SPY vs AAPL CAPM Model

data = yfin.download(tickers=['SPY', 'AAPL'], start='2010-01-01',␣
↪end='2022-01-01', interval='1d')
data = data['Adj Close'].pct_change().fillna(method='bfill')
```

```
y, X = dmatrices(f"AAPL ~ SPY", data=data, return_type="dataframe")

model = sm.OLS(y, X)
results = model.fit()

results.params
```

```
[*********************100%***********************]  2 of 2 completed
```

[55]:
```
Intercept    0.000584
SPY          1.093151
dtype: float64
```

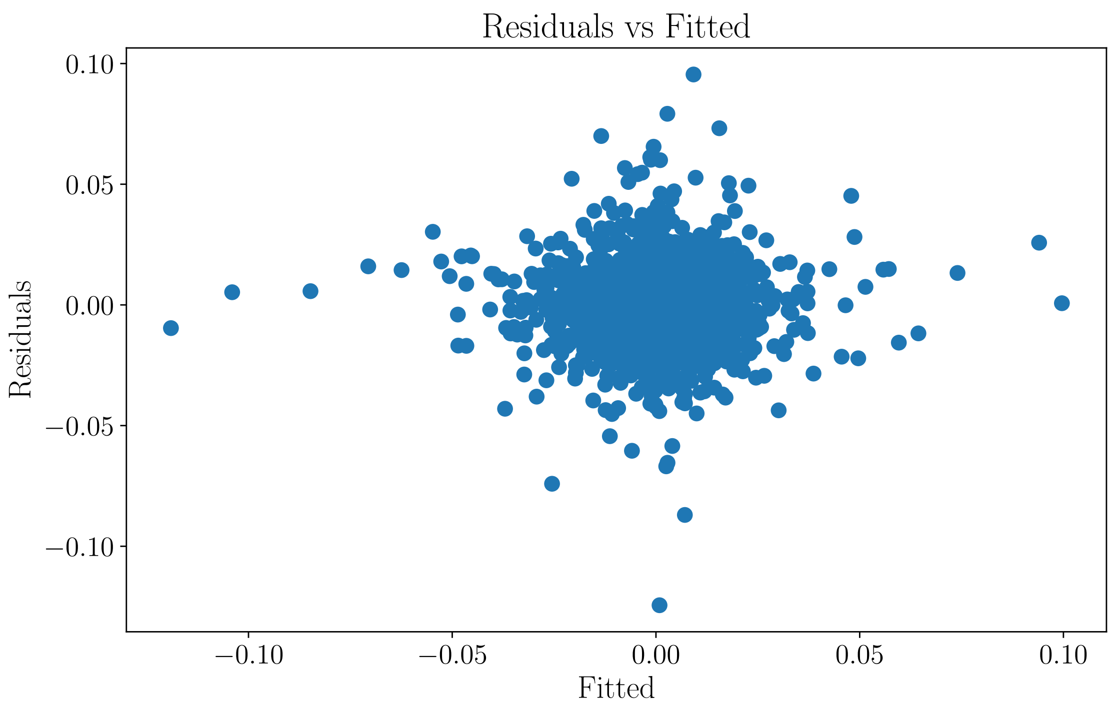
Assuming the risk free rate is zero, the CAPM model between SPY and AAPL can be fitted using $r_{aapl} = \alpha + \beta r_{spy}$. Plugging in our α and β values from above, we get the regression line $r_{aapl} = 0.000584 + 1.093151 r_{spy}$.

[56]:
```
# 1.2 Residuals vs. Fitted

fittedValues = results.fittedvalues
residuals = y['AAPL'] - fittedValues

plt.scatter(fittedValues, residuals)
plt.title('Residuals vs Fitted')
plt.xlabel('Fitted')
plt.ylabel('Residuals')
plt.show
```

[56]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[57]:
```
# 1.2 Residual Sum of Squares RSS

RSS = sum(np.square(residuals))

RSS
```

[57]:
```
0.5316488696383749
```

[58]:
```
# 1.3 Null Regression Model

X = np.ones(len(y), dtype = int)

null_model = sm.OLS(y, X)
null_results = null_model.fit()

null_results.params
```

[58]:
```
const    0.00125
dtype: float64
```

[59]:
```
# 1.3 Alpha in Null Regression Model

np.mean(y)
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
```

```
packages/numpy/core/fromnumeric.py:3462: FutureWarning: In a future version,
DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame.
To retain the old behavior, use 'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

[59]:
```
AAPL    0.00125
dtype: float64
```

As we can see above, the estimated coefficient α in the regression output is equal to the mean of AAPL returns.

[60]:
```python
# 1.4 R Squared Using Formula

null_fittedValues = null_results.fittedvalues

null_residuals = y['AAPL'] - null_fittedValues

null_RSS = sum(np.square(null_residuals))

r_squared = 1 - (RSS/null_RSS)

r_squared
```

[60]:
```
0.4358922873119354
```

[61]:
```python
# 1.4 R Squared

results.rsquared
```

[61]:
```
0.4358922873119355
```

1.5

In order for R^2 to be between 0 and 1, $\frac{RSS_M}{RSS_0}$ must also be between 0 and 1. We know this is true since 1. both RSS_M and RSS_0 are nonnegative as they are the sum of squares and 2. $RSS_M \leq RSS_0$ as the CAPM model for RSS_M minimizes residuals whereas the null regression model for RSS_0 simply finds residuals by subtracting observed values by their mean

PROBLEM 2

(1) $E[M_{n+1} | F_n] = E[M_n U_{n+1} | F_n]$
$= M_n E[U_{n+1} | F_n]$
$= M_n E[U_{n+1}]$
$= M_n$ ✓

(2) $Z_i \sim N(0,1) \rightarrow \sum_{i=1}^{n} Z_i \sim N(0,n)$

$E[M_n] = 0$ since this is the 3rd moment of a normal r.v.

$$E[M_{n+1}|F_n] = E\left[\left(\sum_{i=1}^{n+1} Z_i\right)^3 \Big| F_n\right]$$
$$= E\left[\left(\sum_{i=1}^{n} Z_i + Z_{n+1}\right)^3 \Big| F_n\right]$$
$$= E\left[\left(\sum_{i=1}^{n} Z_i\right)^3 + 3\left(\sum_{i=1}^{n} Z_i\right)^2 (Z_{n+1}) + 3\left(\sum_{i=1}^{n} Z_i\right)(Z_{n+1})^2 + (Z_{n+1})^3 \Big| F_n\right]$$
$$= E[M_n|F_n] + 3 \cdot E\left[\left(\sum_{i=1}^{n} Z_i\right)^2 \Big| F_n\right] \cdot \underbrace{E[Z_{n+1}]}_{0} + 3 \cdot E\left[\sum_{i=1}^{n} Z_i \Big| F_n\right] \cdot \underbrace{E[(Z_{n+1})^2]}_{1} + \underbrace{E[(Z_{n+1})^3]}_{0}$$
$$= M_n + 3 M_n^{1/3}$$

↳ not a martingale

(3) total # of balls in urn at time $n = n+2$

let $X_i = \begin{cases} 1 & \text{if the ball picked if red} \\ 0 & \text{otherwise} \end{cases}$

$R_n = \sum_{i=1}^{n} X_i$

# of red balls at time $n = R_n + 1$

HENCE $f_n = \frac{R_n + 1}{n+2}$

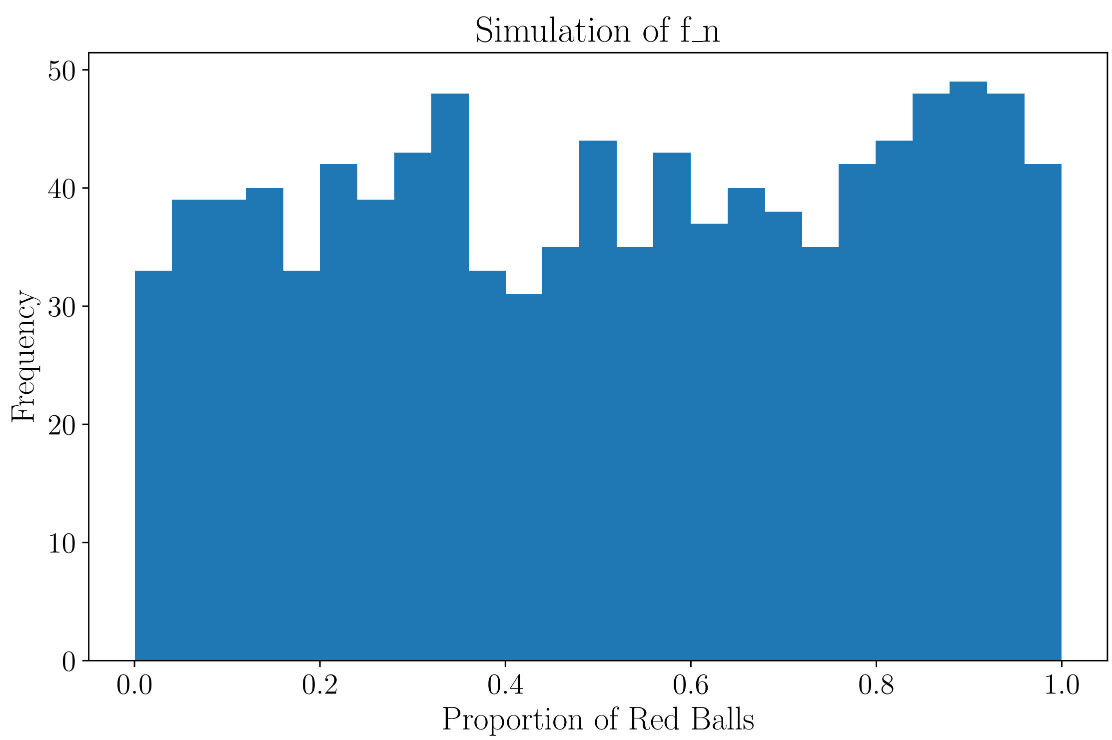
$$E[f_{n+1}|F_n] = E\left[\frac{R_{n+1}+1}{n+3} \Big| F_n\right]$$
$$= E\left[\frac{\sum_{i=1}^{n} X_i + X_{n+1} + 1}{n+3} \Big| F_n\right] \qquad \bigstar = X_{n+1} = 1\left(\frac{R_n+1}{n+2}\right) + 0\left(1 - \frac{R_n+1}{n+2}\right)$$
$$= E\left[\frac{R_n + \frac{R_n+1}{n+2} + 1}{n+3} \Big| F_n\right]$$
$$= E\left[\frac{nR_n + 3R_n + n + 3}{(n+2)(n+3)} \Big| F_n\right]$$
$$= E\left[\frac{\cancel{(n+3)}(R_n+1)}{(n+2)\cancel{(n+3)}} \Big| F_n\right]$$
$$= \frac{R_n+1}{n+2}$$
$$= f_n \checkmark$$

[62]:
```python
# 2.3 Simulate Distribution of f_n

def frac_of_red(n):
    red = 1
    green = 1
    for trial in range(n):
        pick = np.random.choice(['red', 'green'], p = [red/(red + green), green/
 ↪(red + green)])
        if pick == 'red':
            red += 1
        else:
            green += 1
    return (red/(red + green))

proportion_red = [frac_of_red(10000) for trial in range(1000)]

plt.hist(proportion_red, bins=25)
plt.title('Simulation of f_n')
plt.xlabel('Proportion of Red Balls')
plt.ylabel('Frequency')
plt.show()
```



My guess for the distribution of f_n is uniform as we can see in the histogram above that every proportion of red balls has about the same probability as n heads towards infinity.

(4) (a) $E\left[\frac{Z_{n+1}}{\mu^{n+1}} \middle| F_n\right] = E\left[\frac{\sum_{i=1}^{Z_n} \varepsilon_i^{n+1}}{\mu^{n+1}} \middle| F_n\right]$

$= \frac{Z_n \mu}{\mu^{n+1}}$

$= \frac{Z_n}{\mu^n}$ ✓

(b) $E\left[\frac{Z_n}{\mu^n}\right] = E\left[\frac{Z_0}{\mu^0}\right] = 1$ by properties of martingales

$E[Z_n] = \mu^n$

$P(Z_n > 0) = P(Z_n \geq 1) \leq \frac{E[Z_n]}{1}$ by markov's inequality

$= P(Z_n \geq 1) \leq \mu^n$

↳ as $n \to \infty$, $\mu^n \to 0$ and thus $P(Z_n > 0) \to 0$

(c) radioactive decay

↳ Let Z_n be the # of decays in each half-life

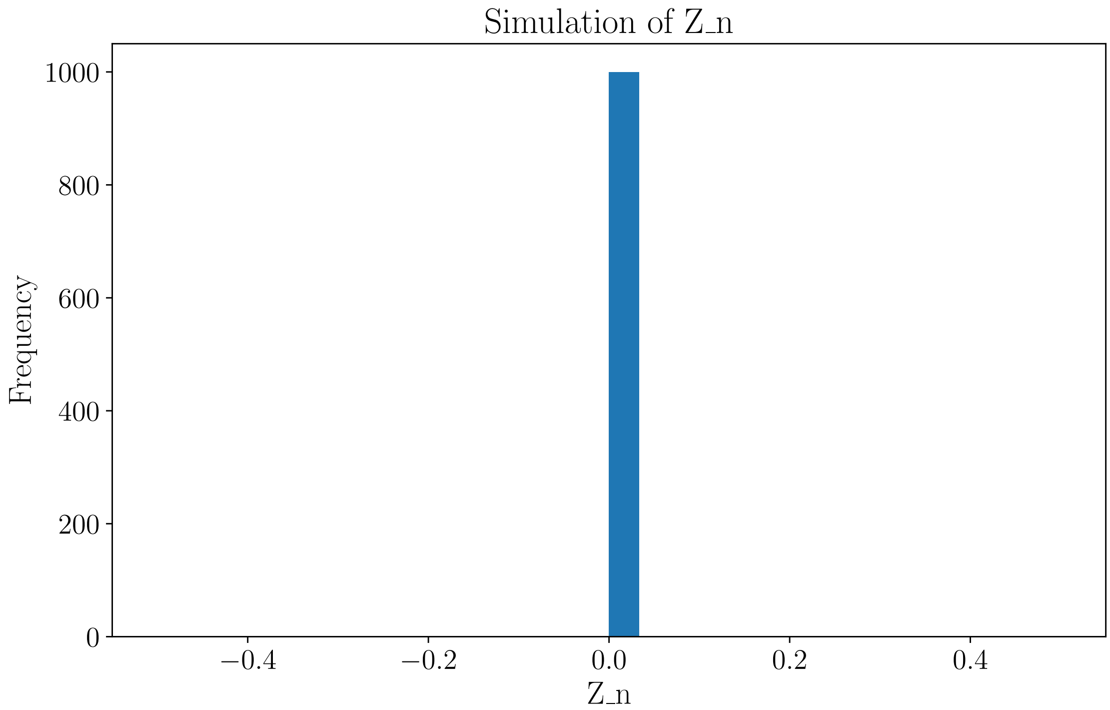
↳ as n (# of half-lives) $\to \infty$, the number of nuclei decays $\to 0$ and the probability for a single nucleus to decay $\to 0$

[63]:
```python
# 2.4 (d) Simulate Z_n

def simulations():
    z_previous = 1
    for i in range(10000):
        z = 0
        for j in range(z_previous):
            epsilon = np.random.poisson(1)
            z += epsilon
        z_previous = z
    return z_previous

Z = [simulations() for trial in range(1000)]

plt.hist(Z, bins=30)
plt.title('Simulation of Z_n')
plt.xlabel('Z_n')
plt.ylabel('Frequency')
plt.show()
```



When n goes to infinity, we can see that Z_n goes to 0. This makes sense because Z_n is the summation of poissons which is poisson.

(5) $$E[Z_{n+1} \mid F_n] = E\left[\sum_{k=1}^{n+1} A_k (M_k - M_{k-1}) \mid F_n\right]$$

$$= Z_n + E[A_{n+1}(M_{n+1} - M_{n+2}) \mid F_n]$$

$$= Z_n + A_{n+1} \underbrace{E[M_{n+1} - M_{n+2} \mid F_n]}_{\substack{\text{martingale difference} \\ = 0}}$$

$$= Z_n \checkmark$$

PROBLEM 3

(1) $E[M_{n+1} | F_n] = E[(\frac{q}{p})^{S_{n+1}} | F_n]$

$= E[M_n (\frac{q}{p})^{X_{n+1}} | F_n]$

$= M_n E[(\frac{q}{p})^{X_{n+1}}]$

$= M_n (p(\frac{q}{p})^1 + q(\frac{q}{p})^{-1})$ $\qquad \frac{p(1-p)}{p} + \frac{p(1-p)}{1-p} = \frac{p(1-p)^2 + p^2(1-p)}{p(1-p)} = 1-p+p = 1$

$= M_n$ ✓

(2) $E[M_T] = E[M_0] = 1$ by Doob

$$M_T = \begin{cases} A & \text{wp } \alpha \\ -B & \text{wp } 1-\alpha \end{cases}$$

$$E[M_T] = \alpha \left(\frac{q}{p}\right)^A + (1-\alpha)\left(\frac{q}{p}\right)^{-B} = 1$$

$$\Leftrightarrow \alpha \left[\left(\frac{q}{p}\right)^A - \left(\frac{q}{p}\right)^{-B}\right] = 1 - \left(\frac{q}{p}\right)^{-B}$$

$$\Leftrightarrow \boxed{\alpha = \frac{1-\left(\frac{p}{q}\right)^B}{\left(\frac{q}{p}\right)^A - \left(\frac{p}{q}\right)^B}}$$

[51]:
```
# 3.3 Game Simulation
probabilities = [0.5, 0.495, 0.490, 0.480, 0.470]

def random_walk(p):
    prob = [p, 1 - p]

    money_won = 0
    money_lost = 0
    duration = 0

    while money_won < 100 and money_lost < 100:
        x = np.random.random(1)
        if x < prob[0]:
            money_won += 1
        else:
            money_lost += 1
        duration += 1

    return money_won, money_lost, duration

def simulation(p, trials):
    num_wins = 0
    for trial in range(trials):
```

```python
        if random_walk(p)[0] == 100:
            num_wins += 1
    prob_win = num_wins/trials
    durations = [random_walk(p)[2] for trial in range(trials)]
    ave_duration = sum(durations)/trials

    return(prob_win, ave_duration)

stats = [simulation(p, 1000) for p in probabilities]

stats
```

[51]:
```
[(0.491, 188.847),
 (0.418, 188.554),
 (0.375, 188.176),
 (0.282, 186.834),
 (0.197, 185.501)]
```

p	0.5	0.495	0.490	0.480	0.470
Probability to win \$100 before losing \$100	0.491	0.418	0.375	0.282	0.197
Average duration of the game	188.847	188.554	188.176	186.834	185.501

3.4

I would bet \$100 on the first bet. As we can see in the table above by continously betting \$1 until we win \$100, the probability of winning \$100 before losing \$100 decreases exponentially for marginal decreases in probabilty p. Further, the probability of winning \$100 by betting \$1 is lower in all 5 cases than just winning in one trial. In other words, $p >$ P(winning \$100 before losing \$100) for all values of p.

[53]:
```python
# 3.5 Stopping Time Simulation

def stopping_time(p):
    prob = [p, 1 - p]

    money_won = 0
    duration = 0

    while money_won < 1:
        x = np.random.random(1)
        if x < prob[0]:
            money_won += 1
        duration += 1

    return duration
```

```
durations = [stopping_time(0.5) for trial in range(1000)]

np.mean(durations)
```

[53]: 1.981

τ is a stopping time since it is a fixed time value where it is possible to determine if it has been reached at any realization. The computed estimate for the expectation of τ is 1.981.