Simple Random Walk

Prompt

Original Prompt can be found here. A copy of the prompt along with the completed exercise can be found under /Applications.

Summary

Two equally matched opponents are competing in a game in which changes in score occur often and in one point increments. (Imagine a basketball game in which every basket counts only one point.) We'll use simulation to investigate the following questions.

1. Which is more likely: that one team leads for most of the game, or that the lead tends to change frequently over the course of the game?
2. When would you expect the largest lead (or deficit) to occur — near the beginning, the end, or in the middle of the game? (If the largest lead (or deficit) is attained at several points in the game, when you do expect it to first occur?)
3. When would you expect the last tie to occur — near the beginning, the end, or in the middle of the game?

Hypothesis

1. I would think if the opponents are competitive, then that would lead to the frequent change of leads.
2. I think they could happen at any time. I don't think there would be a skew to one point of the game or another.
3. Again if it is competitive, I would think that the last tie would occur at the end of the game.

$$A = \text{Team A score} \qquad B = \text{Team B score} \qquad n = \text{Total scores}$$
$$X_n = A - B \qquad \text{After first n scores}$$
$$X_0 = 0$$
$$2n = \text{steps}$$
$$T = \text{Last tie} \qquad L = \text{Total time A leads} \qquad M = \text{First time max differential} \qquad M_A = \text{First time max A lead}$$

Application

Write your own code to conduct and run a simulation to approximate the distribution of each of $T/(2n)$, $L/(2n)$, and $M/(2n)$ for $n = 100$. Summarize the results with appropriate plots and summary statistics, and describe the distributions.

Using the Symbulate package and referencing their documentation.

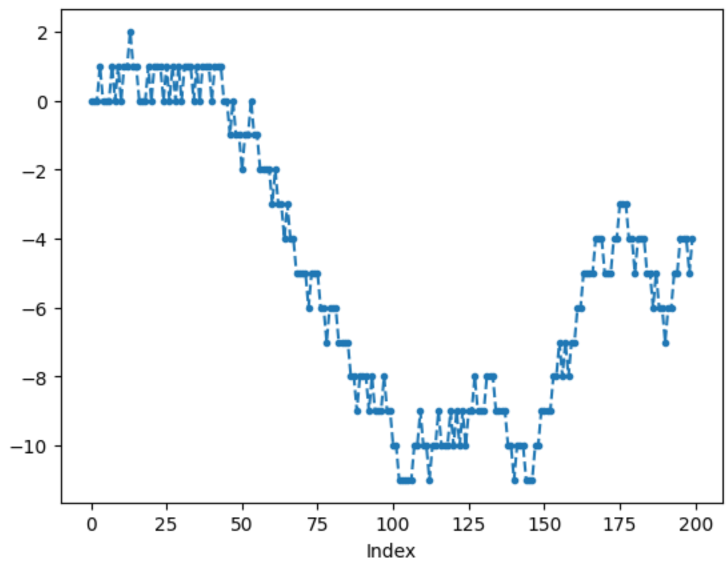
```
pip install symbulate
```

```
Requirement already satisfied: symbulate in /usr/local/lib/python3.10/dist-packages (0.5.7)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from symbulate) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from symbulate) (1.11.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from symbulate) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (4.47.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->symbulate) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->symbulate) (1
```

```
from symbulate import *
%matplotlib inline
```

First lets look at a single simulation and resulting path.

```
#Setting up based on prompt
n = 100
steps = 2*n

#Following and modifying Random Processes section from symbulate documentation
P = Bernoulli(0.5)**steps
Z = RV(P)
A = RandomProcess(P, Naturals())
B = RandomProcess(P, Naturals())

A[0] =  0
B[0] =  0
for i in range(steps):
    if i%2 == 0:
      A[i+1] = A[i] + Z[i+1]  # Has possession - Possibility to score
      B[i+1] = B[i]           # Defense!
    else:
      A[i+1] = A[i]           # Defense!
      B[i+1] = B[i] + Z[i+1]  # Has possession - Possibility to score

#create process to show differential
X = A - B




#To see A score over time
#A.sim(1).plot(alpha = 1, tmin = 0, tmax = 100)


#To see B score over time
#B.sim(1).plot(alpha = 1, tmin = 0, tmax = 100)


#To see A - B differential over time
games = X.sim(1)
games.plot(alpha = 1, tmin = 0, tmax = 200)
```




```
def last_tie(x):
    last_tie_index = 0
    for i in range(steps):
        if x[i] == 0:
            last_tie_index = i + 1
    return last_tie_index

tie = games.apply(last_tie).get(0)
msg = "The last tie was at step {}, {:.2f}% through the game.".format(tie, (tie/steps)*100)
print(msg)
```

```
The last tie was at step 54, 27.00% through the game.
```

```
def time_lead_A(x):
    time_lead = 0
    for i in range(steps):
        if x[i] > 0:
            time_lead += 1
    return time_lead

lead = games.apply(time_lead_A).get(0)
msg = "The total time team A held the lead was {} steps, {:.2f}% of the game.".format(lead,(lead/steps)*100)
print(msg)
```

```
The total time team A held the lead was 25 steps, 12.50% of the game.
```

```
def max(x):
    max_num = 0
    for i in range(steps):
        if abs(x[i]) > max_num:
              max_num = abs(x[i])
    return max_num

def maxa(x):
    max_num = 0
    for i in range(steps):
        if x[i] > max_num:
              max_num = x[i]
    return max_num

# diff_max = games.apply(max).get(0)
# msg = "The greatest differential in the game was {} points.".format(diff_max)
# print(msg)
```

```
def maxi(x):
    max_num = max(x)
    maxi = 0
    for i in range(steps):
      if abs(x[i]) == max_num:
            maxi = i + 1
            return maxi

diff_max = games.apply(max).get(0)
diff_maxi = games.apply(maxi).get(0)
msg = "The greatest differential in the game of {} occured at step {}, {:.2f}% through the game.".format(diff_max, diff_maxi, (diff_maxi/ste
print(msg)
```

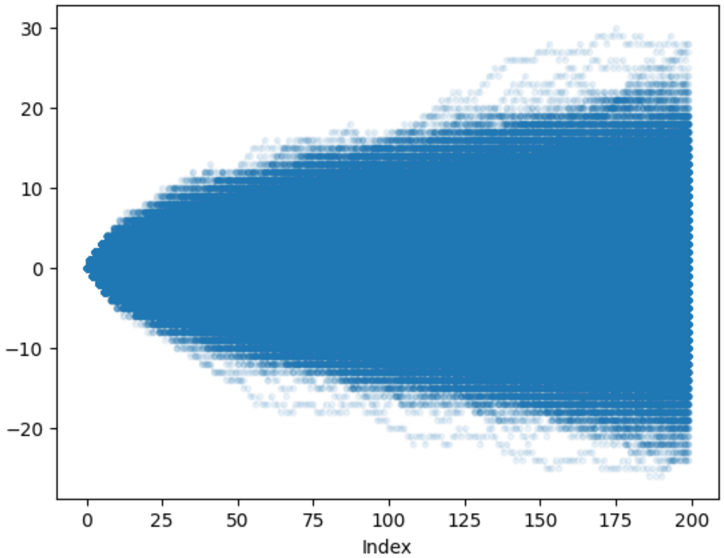
```
The greatest differential in the game of 11 occured at step 103, 51.50% through the game.
```

```
def maxia(x):
    max_num = maxa(x)
    maxi = 0
    for i in range(steps):
      if x[i] == max_num:
            maxi = i + 1
            return maxi

diff_max_a = games.apply(maxa).get(0)
diff_maxi_a = games.apply(maxia).get(0)
msg = "The largest A lead differential in the game of {} occured at step {}, {:.2f}% through the game.".format(diff_max_a, diff_maxi_a, (dif
print(msg)
```

```
The largest A lead differential in the game of 2 occured at step 14, 7.00% through the game.
```

Now lets increase the amount of paths and find the distributions of T, L, and M values.

```
#To see multiple A - B differentials over time
games = X.sim(10000)
games.plot(tmin = 0, tmax = 200)
```

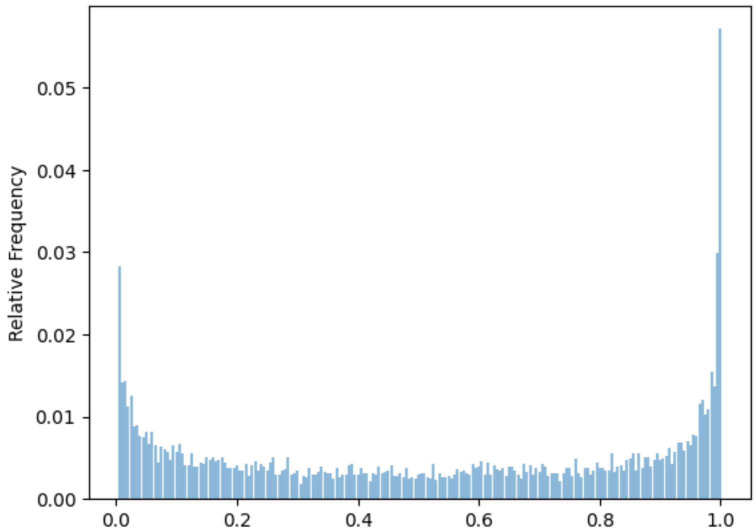


Last Ties

```
#find last ties of the games
tie = games.apply(last_tie)

#normalize the ties to % progression of game
tie = tie/200

#tabulate distribution
tie.tabulate(normalize=True)
```

Value	Relative Frequency
0.005	0.0283
0.01	0.0142
0.015	0.0144
0.02	0.0113
0.025	0.0126
0.03	0.0088
0.035	0.009
0.04	0.0076
0.045	0.0075
0.05	0.0081
0.055	0.0067
0.06	0.0082
0.065	0.0066
0.07	0.0044
0.075	0.0063
0.08	0.0061
0.085	0.0058
0.09	0.0047
0.095	0.0066
...	...
1.0	0.0571
Total	**1.0000000000000002**

```
#plot distribution
tie.plot()
```

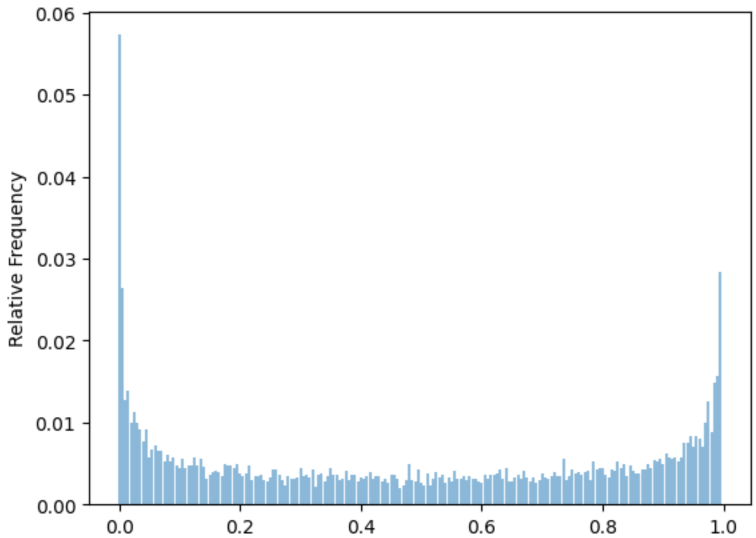


Time A Lead

```
#find total time A lead
lead = games.apply(time_lead_A)

#normalize the lead to % progression of game
lead = lead/200

#tabulate distribution
lead.tabulate(normalize=True)
```

Value	Relative Frequency
0.0	0.0573
0.005	0.0264
0.01	0.0128
0.015	0.0139
0.02	0.0099
0.025	0.0112
0.03	0.0099
0.035	0.0092
0.04	0.0077
0.045	0.0092
0.05	0.0058
0.055	0.0068
0.06	0.0072
0.065	0.0066
0.07	0.0065
0.075	0.0052
0.08	0.0061
0.085	0.0053
0.09	0.0058
...	...
0.995	0.0283
Total	**1.0**

```
#plot distribution
lead.plot()
```

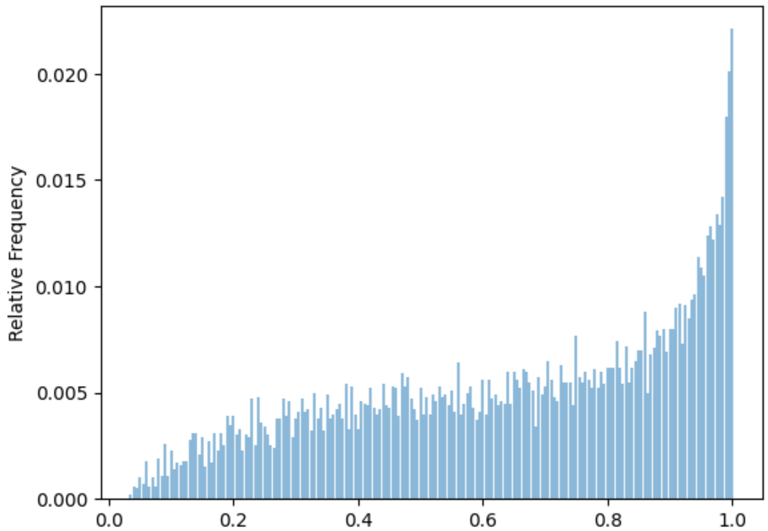


First Max Lead

```
#find max and first time it occurs in game
maxdi = games.apply(maxi)

#normalize the ties to % progression of game
maxdi = maxdi/200

#tabulate distribution
maxdi.tabulate(normalize=True)
```

Value	Relative Frequency
0.035	0.0002
0.04	0.0006
0.045	0.0005
0.05	0.001
0.055	0.0007
0.06	0.0018
0.065	0.0006
0.07	0.001
0.075	0.0006
0.08	0.0019
0.085	0.0011
0.09	0.0026
0.095	0.0011
0.1	0.0023
0.105	0.0014
0.11	0.0017
0.115	0.0016
0.12	0.0018
0.125	0.0018
...	...
1.0	0.0221
Total	**0.9999999999999998**

```
#plot distribution
maxdi.plot()
```

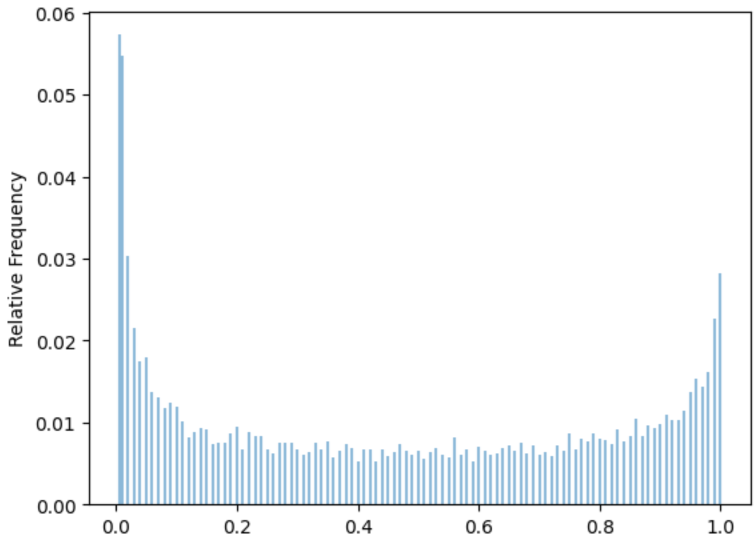



```
#find A's max and first time it occurs in game
maxdia = games.apply(maxia)

#normalize the ties to % progression of game
maxdia = maxdia/200

#tabulate distribution
maxdia.tabulate(normalize=True)
```

Value	Relative Frequency
0.005	0.0573
0.01	0.0548
0.02	0.0304
0.03	0.0215
0.04	0.0174
0.05	0.018
0.06	0.0138
0.07	0.013
0.08	0.0118
0.09	0.0124
0.1	0.0119
0.11	0.0101
0.12	0.0082
0.13	0.0088
0.14	0.0093
0.15	0.0091
0.16	0.0074
0.17	0.0075
0.18	0.0076
...	...
1.0	0.0282
Total	**0.9999999999999999**

```
#plot distribution
maxdia.plot()
```


```
#plot distribution
maxdia.plot()
```



Analysis and Conclusion

Consider the three questions at the start of this page; what do your simulation results suggest? Write a brief report summarizing your results and conclusions.

To reiterate the questions:

1. Which is more likely: that one team leads for most of the game, or that the lead tends to change frequently over the course of the game?
2. When would you expect the largest lead (or deficit) to occur — near the beginning, the end, or in the middle of the game? (If the largest lead (or deficit) is attained at several points in the game, when you do expect it to first occur?)
3. When would you expect the last tie to occur — near the beginning, the end, or in the middle of the game?

1. After conducting the simulation, I see now how even if the teams are competitive, there are situtations where a team can take the lead early and from then on the score differential is relatively constant but that entire time one team hold the lead. We can also see from the plot showing the many possible paths that the middle section of the paraboloid is the darkest which indicates to me most paths flow through or cross there many times. Therefore I think the probability of frequent lead changes or team leads for most of the game is more equally distributed, even if they are competitive.