

EE6150: Stochastic Modelling and Queuing Theory

Python Assignment

Amizhthni P R K, EE21B015

April 19, 2024

Contents

1	Problem Statement	1
1.1	Tennis Match Analysis with Service Protocols	1
1.2	Function Signature	1
1.3	Constraints	1
1.4	Example taken here	1
2	Approach 1 (Simpler) - Without considering Deuce condition	2
2.1	Rules of Tennis Game considered	2
2.2	Algorithm	2
2.2.1	Match Simulation	2
2.2.2	Analytic Calculation	2
2.3	Code	3
2.4	Results	9
3	Approach 2 (More Complex) - Considering Deuce condition	13
3.1	Rules of Tennis Game considered	13
3.2	Algorithm	13
3.3	Code	13
3.4	Results	16
3.5	Analytical Deuce Solution	22

1 Problem Statement

1.1 Tennis Match Analysis with Service Protocols

In a tennis match between players A and B, each rally that begins with a serve by player A is won by player A with probability p_a , and each rally that begins with a serve by player B is won by player A with probability p_b . The winner of the rally earns a point and becomes the server of the next rally. Player A serves first. You are tasked with analyzing tennis matches between players A and B based on two different service protocols:

- 1 **Winner Serves Protocol:** The winner of each rally serves for the next rally.
- 2 **Alternating Service Protocol:** The service alternates between players A and B after each rally.

Write a function to simulate tennis matches under both service protocols and analyze the outcomes. The function should accept the following parameters:

- **p_a _values:** A list of probabilities of player A winning a rally when serving. Each element represents a different value of p_a to be tested.
- **p_b :** The probability of player A winning a rally when not serving (i.e., player B serving).
- **N:** The number of matches to simulate for each value of p_a .

1.2 Function Signature

```
def tennis_match_analysis_with_graph(pa_values: List[float], pb: float, n: int) -> None:  
    pass
```

1.3 Constraints

- $0 \leq p_a, p_b \leq 1$
- $n \geq 1$

1.4 Example taken here

- p_a _values = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
- p_b = one from [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
- n = 1000
- tennis_match_analysis_with_graph(p_a _values, p_b , n)

2 Approach 1 (Simpler) - Without considering Deuce condition

2.1 Rules of Tennis Game considered

- We play n number of matches as specified in the question.
- Whoever wins **5 rallies in a match first** is declared the winner (0, 15, 30, 40, game). There is no notion of number of sets won here, it is simply a number of rallies that lead to a winner of a match.
- We write the simulation for both the specified protocols

2.2 Algorithm

The algorithm is divided into two subparts.

2.2.1 Match Simulation

- The matches for alternating serves and winner serves are simulated separately.
- Both the algorithms run until either A or B reaches a score of 5 points.
- The winner of each rally is calculated from p_a and p_b using *np.random.choice()*.
- Each match is played n times and number of wins of each player is counted and plotted.

2.2.2 Analytic Calculation

- We start the analysis by calculating the probability of A winning each rally.
- For alternating serves it's straightforward, just p_a and p_b alternatively.
- For winner serves the probability of A winning a rally is calculated as follows :
 - The probability of A winning first rally $P(A_1)$ is set as p_a since A serves first.
 - The second rally $P(A_2)$ has two cases namely, A winning and A losing.
 - If A wins the first round probability of A winning second round becomes p_a .
 - If A loses, the probability becomes p_b .
 - Hence we get the probability of A winning second round as

$$P(A_2) = P(A_1) * p_a + (1 - P(A_1)) * p_b \quad (1)$$

extending this,

$$P(A_n) = P(A_{n-1}) * p_a + (1 - P(A_{n-1})) * p_b \quad (2)$$

- The round ends after a maximum of 9 rallies.
- The probabilities calculated above are then used to calculate the probability of A winning a match $P(A)$.
- This is calculated as

$$P(A) = P(X \geq 5) = P(5) + P(6) + P(7) + P(8) + P(9) \quad (3)$$

where, $P(X)$ denotes the probability of A winning X rallies.

- This $P(A)$ is multiplied by n to get expected number of matches A wins in n matches.

The results are then plotted in separate graphs to compare them.

2.3 Code

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from typing import List
4
5
6 def winner_serves_match(pa, pb):
7     player_serving = 'A' # Player A serves first
8     score_A = 0
9     score_B = 0
10
11     while score_A < 5 and score_B < 5:
12         if player_serving == 'A':
13             if not np.random.choice(2,1,p=[pa,1-pa])[0]: #A wins if this
14                 ↪ outputs 0
15                 score_A += 1 #same player repeats
16             else:
17                 score_B += 1 #score update
18                 player_serving = 'B' #swap for the winner
19         else:
20             if not np.random.choice(2,1,p=[pb,1-pb])[0]: #B wins if this
21                 ↪ outputs 1
22                 score_A += 1
23                 player_serving = 'A' #swap for the winner
24             else:
25                 score_B += 1 #score update #same player repeats
26     ret = 'A'
27     if(score_B>score_A):
28         ret='B'
29     return ret
30
31 def alternating_service_match(pa, pb):
32     score_A = 0
33     score_B = 0
34     player_serving = 'A' # Player A serves first
35
36     while score_A < 5 and score_B < 5:
37         if player_serving == 'A':
38             if not np.random.choice(2,1,p=[pa,1-pa])[0]:
39                 score_A += 1
40             else:
41                 score_B += 1 #score update
42         else:
43             if not np.random.choice(2,1,p=[pb,1-pb])[0]:
44                 score_A += 1
45             else:
46                 score_B += 1 #score update
```

```

47     if player_serving == 'A': #swap players
48         player_serving = 'B'
49     elif player_serving == 'B':
50         player_serving = 'A'
51
52     ret = 'A'
53     if(score_B>score_A):
54         ret='B'
55     return ret
56
57 def simulate_matches(pa_values, pb, n):
58     winner_serves_wins = []
59     alternating_service_wins = []
60
61     for pa in pa_values:
62         winner_serves_wins_count = 0
63         alternating_service_wins_count = 0
64         for _ in range(n):
65             winner = winner_serves_match(pa, pb)
66             if winner == 'A':
67                 winner_serves_wins_count += 1
68
69             winner = alternating_service_match(pa, pb)
70             if winner == 'A':
71                 alternating_service_wins_count += 1
72
73         winner_serves_wins.append(winner_serves_wins_count)
74         alternating_service_wins.append(alternating_service_wins_count)
75
76     return winner_serves_wins, alternating_service_wins
77
78 def tennis_match_analysis_with_graph(pa_values: List[float], pb: float, n:
↪ int):
79     winner_serves_wins, alternating_service_wins =
↪     simulate_matches(pa_values, pb, n)
80
81     plt.plot(pa_values, winner_serves_wins, label="Winner Serves
↪ Protocol",marker='o')
82     plt.plot(pa_values, alternating_service_wins, label="Alternating
↪ Service Protocol",marker='o')
83     plt.xticks(pa_values)
84     plt.xlabel('Probability of A winning when A is the server (pa)')
85     plt.ylabel('Number of Wins for A')
86     plt.title('Simulation of Tennis for pb={}'.format(pb))
87     plt.legend()
88     plt.savefig('ig{}.png'.format(pb))
89     plt.show()
90
91 def dp_simul_winner_serve(pa,pb):
92     pw=np.zeros(10);

```

```

93 pw[0]=pa
94 for round in range(1,10):
95     pw[round]=pw[round-1]*pa+(1-pw[round-1])*pb #dp array
96     ↪ initialisation
97
98 p9=0
99 for i in range(9): #calculating p5, , A loses 4 matches
100     p9t=(1-pw[i])
101     for j in range(i,9):
102         if(j!=i):
103             p9t1=p9t*(1-pw[j])
104             for k in range(j,9):
105                 if(k!=i and k!=j):
106                     p9t2=p9t1*(1-pw[k])
107                     for l in range(k,9):
108                         if(l!=i and l!=j and l!=k):
109                             p9t3=p9t2*(1-pw[l])
110                             for m in range(9):
111                                 if(m!=i and m!=j and m!=k and m!=l):
112                                     p9t3*=pw[m]
113                             p9+=p9t3
114
115 for i in range(9): #calculating p6, A loses 3 matches
116     p9t=(1-pw[i])
117     for j in range(i,9):
118         if(j!=i):
119             p9t1=p9t*(1-pw[j])
120             for k in range(j,9):
121                 if(k!=i and k!=j):
122                     p9t2=p9t1*(1-pw[k])
123                     for l in range(9):
124                         if(l!=i and l!=j and l!=k):
125                             p9t2*=(pw[l])
126                     p9+=p9t2
127
128 for i in range(9): #calculating p7, A loses 2 matches
129     p9t=(1-pw[i])
130     for j in range(i,9):
131         if(j!=i):
132             p9t1=p9t*(1-pw[j])
133             for k in range(9):
134                 if(k!=i and k!=j):
135                     p9t1*=(pw[k])
136             p9+=p9t1
137
138 for i in range(9): #calculating p8, A loses only 1 match
139     p9t=(1-pw[i])
140     for j in range(9):
141         if(j!=i):
142             p9t*=(pw[j])

```

```

142         p9+=p9t
143
144     p9t=1    #calculating p9, A wins all 9 matches
145     for i in range(9):
146         p9t*=(pw[i])
147     p9+=p9t
148
149     #Care has been taken to ensure there are no overlaps between the 5
150     ↪ probabilities calculated
151     p_final = (p9)
152     # print(p_final)
153     # print(pw)
154     return p_final
155
156 def dp_simul_alternate_serve(pa,pb):
157     pw=np.zeros(10);
158     for round in range(0,10):
159         if(round%2):
160             pw[round]=pb
161         else:
162             pw[round]=pa
163
164
165     p9=0
166     for i in range(9):
167         p9t=(1-pw[i])
168         for j in range(i,9):
169             if(j!=i):
170                 p9t1=p9t*(1-pw[j])
171                 for k in range(j,9):
172                     if(k!=i and k!=j):
173                         p9t2=p9t1*(1-pw[k])
174                         for l in range(k,9):
175                             if(l!=i and l!=j and l!=k):
176                                 p9t3=p9t2*(1-pw[l])
177                                 for m in range(9):
178                                     if(m!=i and m!=j and m!=k and m!=l):
179                                         p9t3*=pw[m]
180                                 p9+=p9t3
181
182     for i in range(9):
183         p9t=(1-pw[i])
184         for j in range(i,9):
185             if(j!=i):
186                 p9t1=p9t*(1-pw[j])
187                 for k in range(j,9):
188                     if(k!=i and k!=j):
189                         p9t2=p9t1*(1-pw[k])
190                         for l in range(9):

```



```

191         if(l!=i and l!=j and l!=k):
192             p9t2*=(pw[l])
193         p9+=p9t2
194
195     for i in range(9):
196         p9t=(1-pw[i])
197         for j in range(i,9):
198             if(j!=i):
199                 p9t1=p9t*(1-pw[j])
200                 for k in range(9):
201                     if(k!=i and k!=j):
202                         p9t1*=(pw[k])
203                 p9+=p9t1
204
205     for i in range(9):
206         p9t=(1-pw[i])
207         for j in range(9):
208             if(j!=i):
209                 p9t*=(pw[j])
210         p9+=p9t
211
212     p9t=1
213     for i in range(9):
214         p9t*=(pw[i])
215     p9+=p9t
216
217
218     p_final = (p9)
219     # print(p_final)
220     # print(pw)
221     return p_final
222
223 def final_dp(pa_values,pb,n):
224     winner_serves_wins,alternating_service_wins=[],[]
225     for pa in pa_values:
226         winner_serves_wins.append(dp_simul_winner_serve(pa,pb)*n)
227         alternating_service_wins.append(dp_simul_alterate_serve(pa,pb)*n)
228
229     plt.plot(pa_values, winner_serves_wins, label="Winner Serves
230             ↪ Protocol",marker='o')
231     plt.plot(pa_values, alternating_service_wins, label="Alternating
232             ↪ Service Protocol",marker='o')
233     plt.xticks(pa_values)
234     plt.xlabel('Probability of A winning when A is the server (pa)')
235     plt.ylabel('Number of Wins for A')
236     plt.title('Analytical solution for pb={}'.format(pb))
237     plt.legend()
238     plt.savefig('imag{}.png'.format(pb))
239     plt.show()

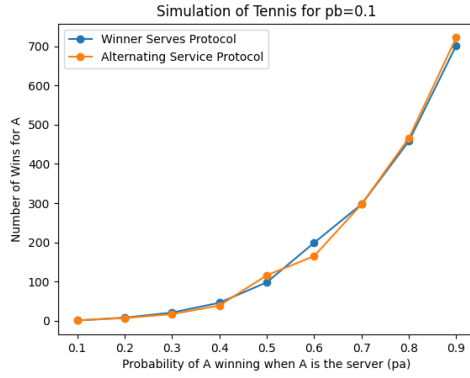
```

239

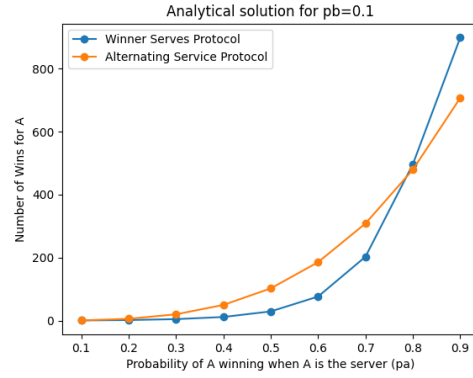
240

Listing 1: Code snippet1-for plotting Simulation and calculated number of wins for A

2.4 Results

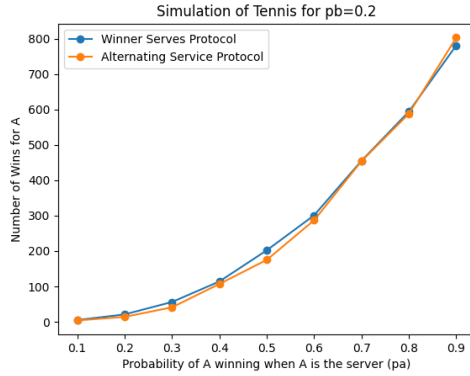


(a) Simulated wins for A using Random module

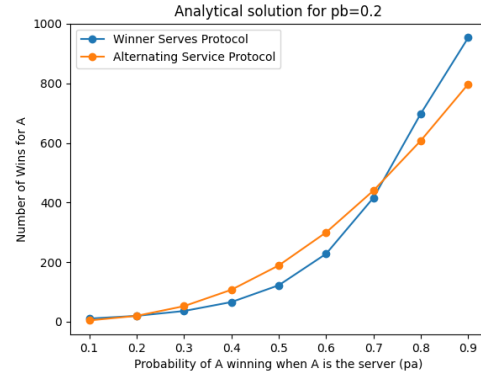


(b) Expected number of wins of A as calculated by us

Figure 1: $p_b = 0.1$ for 10 p_a values



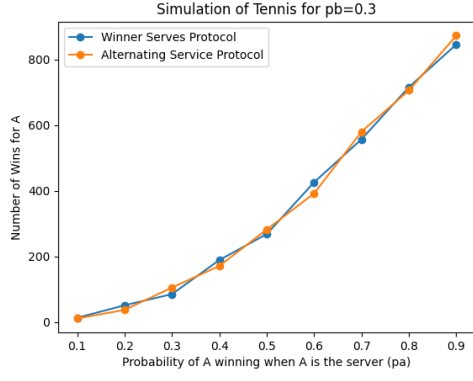
(a) Simulated wins for A using Random module



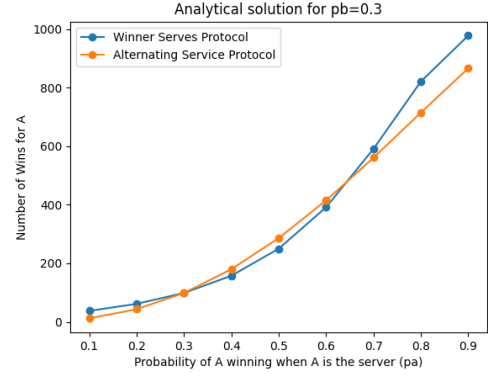
(b) Expected number of wins of A as calculated by us

Figure 2: $p_b = 0.2$ for 10 p_a values

- We see that our simulation graphs match with the analytical graphs in all cases, especially so for the alternative serving protocol.
- For the $p_a=p_b=0.5$ case both results give us 500 as the answer, which is intuitively correct.
- The nature of the parabola inverts after $p_b=0.5$ and we get higher number of wins for lower p_a values. We also get a straight line for $p_b=0.5$ as expected.

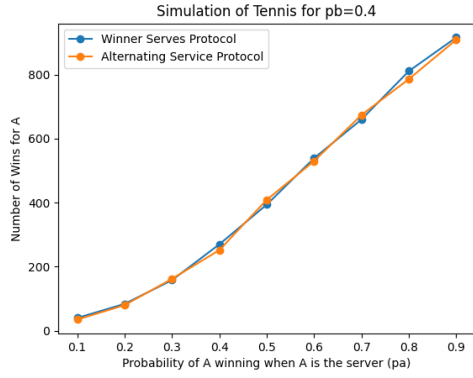


(a) Simulated wins for A using Random module

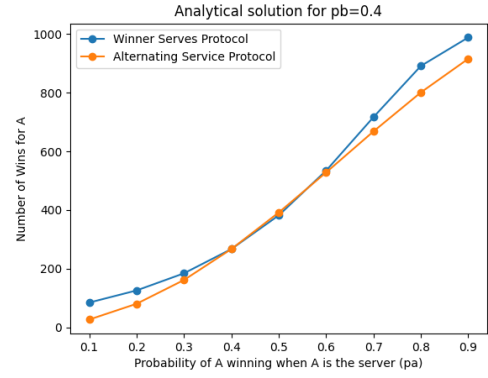


(b) Expected number of wins of A as calculated by us

Figure 3: $p_b = 0.3$ for 10 p_a values

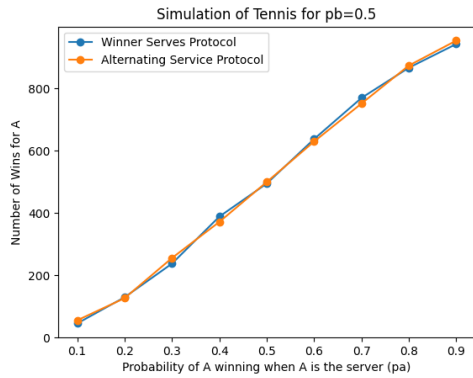


(a) Simulated wins for A using Random module

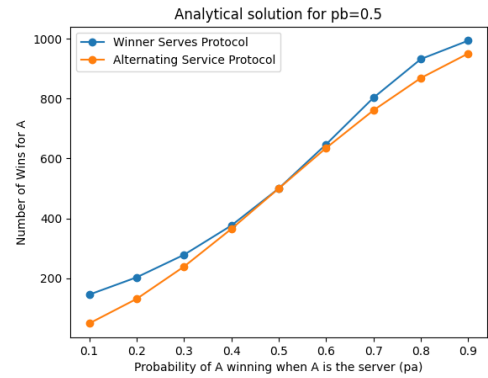


(b) Expected number of wins of A as calculated by us

Figure 4: $p_b = 0.4$ for 10 p_a values

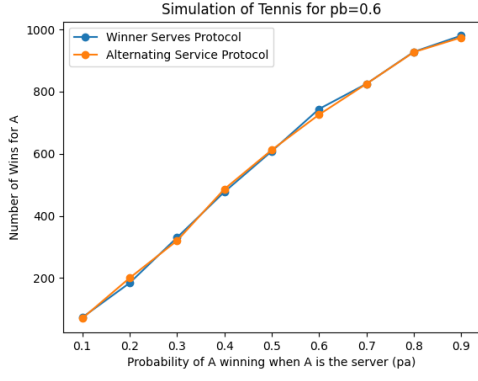


(a) Simulated wins for A using Random module

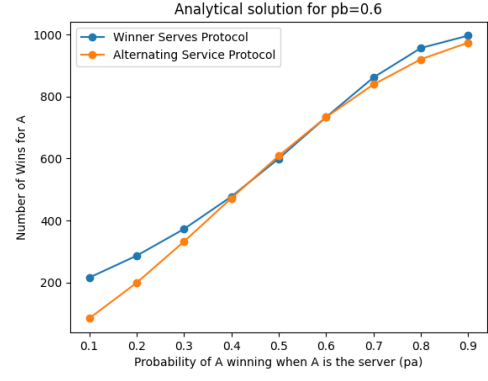


(b) Expected number of wins of A as calculated by us

Figure 5: $p_b = 0.5$ for 10 p_a values

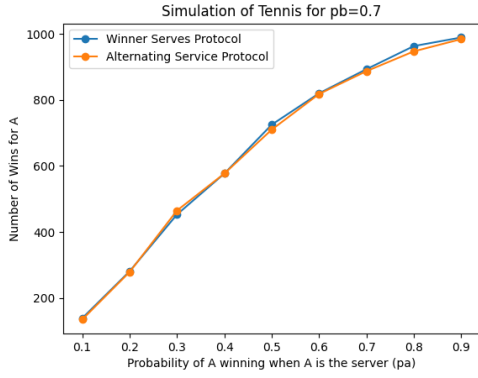


(a) Simulated wins for A using Random module

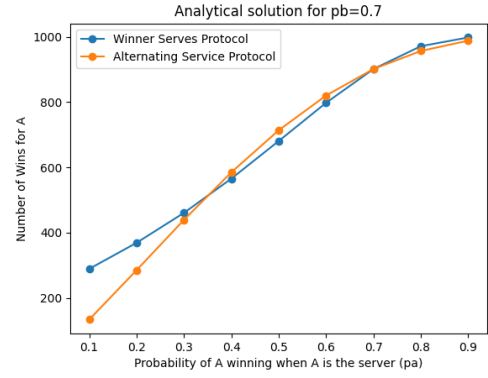


(b) Expected number of wins of A as calculated by us

Figure 6: $p_b = 0.6$ for 10 p_a values

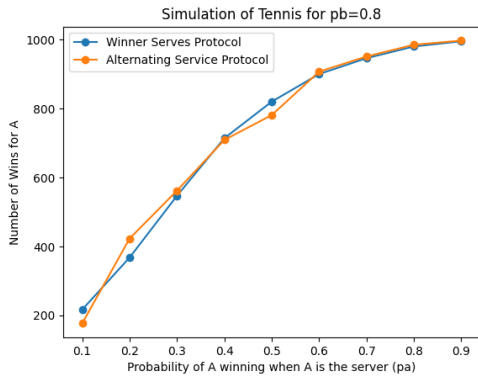


(a) Simulated wins for A using Random module

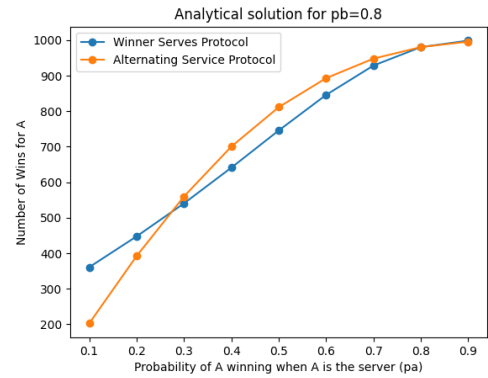


(b) Expected number of wins of A as calculated by us

Figure 7: $p_b = 0.7$ for 10 p_a values

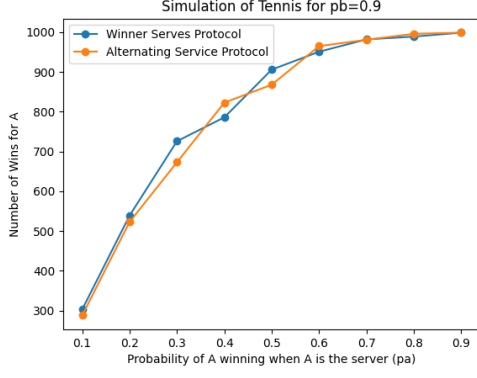


(a) Simulated wins for A using Random module

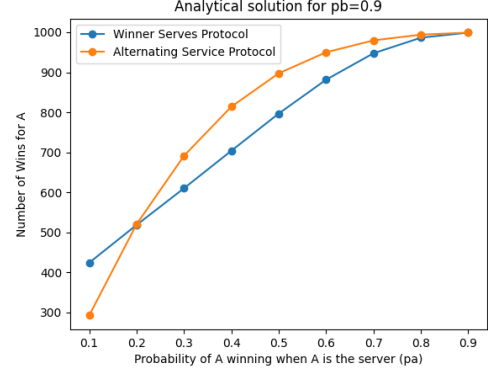


(b) Expected number of wins of A as calculated by us

Figure 8: $p_b = 0.8$ for 10 p_a values

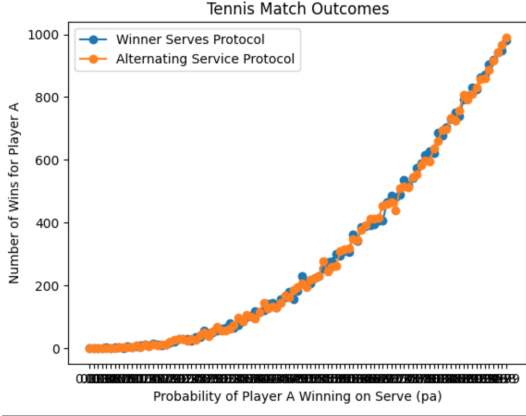


(a) Simulated wins for A using Random module

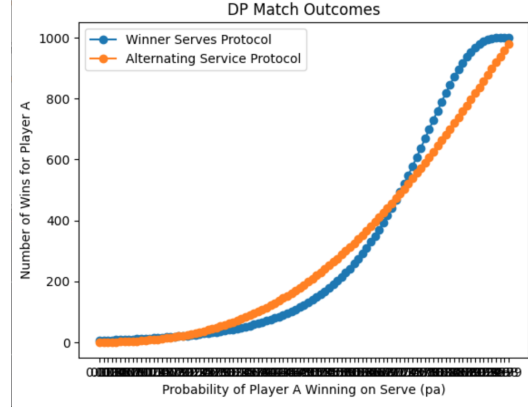


(b) Expected number of wins of A as calculated by us

Figure 9: $p_b = 0.9$ for 10 p_a values

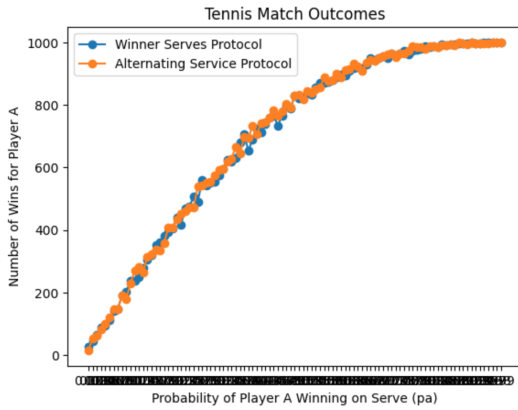


(a) Simulated wins for A using Random module

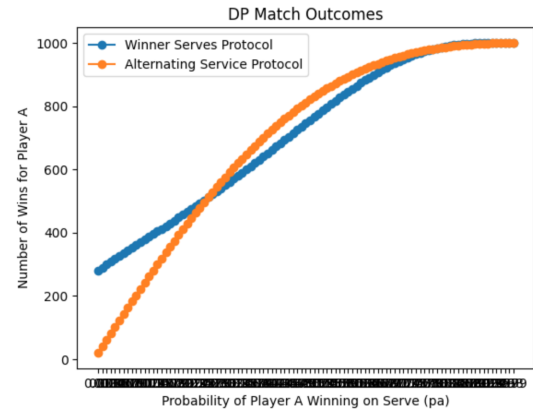


(b) Expected number of wins of A as calculated by us

Figure 10: $p_b = 0.2$ for 100 uniformly distributed p_a values



(a) Simulated wins for A using Random module



(b) Expected number of wins of A as calculated by us

Figure 11: $p_b = 0.8$ for 100 uniformly distributed p_a values. Note that both these parabolas are inverted due to change in p_b value. The expected wins graph is clearly the denoised or smooth version of the simulation. This noise can be accounted for by the random module

3 Approach 2 (More Complex) - Considering Deuce condition

3.1 Rules of Tennis Game considered

- We play n number of matches as specified in the question.
- Whoever wins **5 rallies in a match first** is declared the winner (0, 15, 30, 40, game).
- Additionally, when A and B reach the same score (40-40), a situation called the "DEUCE", one of them is expected to establish a two point lead in order to win. This code takes into account this additional complexity.
- We write the simulation for both the specified protocols.

3.2 Algorithm

- The simulation is mostly similar to the simulation for the game without deuces with one crucial difference.
- There is a additional condition which keeps the loop running while the score difference is less than 2.
- To prevent infinite looping the code breaks and outputs the higher scoring player when their scores reach 500, but the probability of this happening is infinitesimally small.
- The analytical solution is presented in the form of a **Markov chain diagram** in section 3.5.

3.3 Code

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4
5 def winner_serves_match(pa, pb):
6     player_serving = 'A' # Player A serves first
7     score_A = 0
8     score_B = 0
9
10    while (score_A < 5 and score_B < 5) or abs(score_A - score_B)<2: #the
11        ↪ second condition here checks for deuce
12        if player_serving == 'A':
13            if not np.random.choice(2,1,p=[pa,1-pa])[0]: #we choose 0 with
14                ↪ pa,
15                score_A += 1 #score update
16            else:
17                score_B += 1
18                player_serving = 'B' #swap for winner
19        else:
20            if not np.random.choice(2,1,p=[pb,1-pb])[0]:
21                score_A += 1
22                player_serving = 'A'
23            else:
```

```

22         score_B += 1 #score update
23
24     ret = 'A'
25     if(score_B>score_A):
26         ret='B'
27
28     if(score_A>500):
29         return ret
30     return ret
31
32
33 def alternating_service_match(pa, pb):
34     score_A = 0
35     score_B = 0
36     player_serving = 'A' # Player A serves first
37
38     while (score_A < 5 and score_B < 5) or abs(score_A-score_B)<2:
39         if player_serving == 'A':
40             if not np.random.choice(2,1,p=[pa,1-pa])[0]:
41                 score_A += 1
42             else:
43                 score_B += 1 #score update
44         else:
45             if not np.random.choice(2,1,p=[pb,1-pb])[0]:
46                 score_A += 1
47             else:
48                 score_B += 1 #score update
49
50         if player_serving == 'A': #Swapping after every match
51             player_serving = 'B'
52         elif player_serving == 'B':
53             player_serving = 'A'
54         '''if(score_A>10):
55             print("Score A : ",score_A)
56             print("Score B : ",score_B)'''
57     ret = 'A'
58     if(score_B>score_A):
59         ret='B'
60
61     if(score_A>500):
62         return ret
63     return ret
64
65 def simulate_matches(pa_values, pb, n):
66     winner_serves_wins = []
67     alternating_service_wins = []
68
69     for pa in pa_values:
70         winner_serves_wins_count = 0
71         alternating_service_wins_count = 0

```



```

72     for _ in range(n):
73         winner = winner_serves_match(pa, pb)
74         if winner == 'A':
75             winner_serves_wins_count += 1
76
77         winner = alternating_service_match(pa, pb)
78         if winner == 'A':
79             alternating_service_wins_count += 1
80     '''if(score_A>20):
81         print("Score A : ",score_A)
82         print("Score B : ",score_B)'''
83     winner_serves_wins.append(winner_serves_wins_count)
84     alternating_service_wins.append(alternating_service_wins_count)
85
86     return winner_serves_wins, alternating_service_wins
87
88 def tennis_match_analysis_with_graph(pa_values, pb, n):
89     winner_serves_wins, alternating_service_wins =
90     ↪ simulate_matches(pa_values, pb, n)
91
92     plt.plot(pa_values, winner_serves_wins, label="Winner Serves
93     ↪ Protocol",marker='o')
94     plt.plot(pa_values, alternating_service_wins, label="Alternating
95     ↪ Service Protocol",marker='o')
96     plt.xticks(pa_values)
97     plt.xlabel('Probability of A winning when A is the server (pa)')
98     plt.ylabel('Number of Wins for A')
99     plt.title('Simulation of Tennis for pb={}'.format(pb))
100     plt.legend()
101     plt.savefig('o{}.png'.format(pb))
102     plt.show()

```

Listing 2: Code snippet2-for plotting just the simulations for the Deuce condition.

3.4 Results

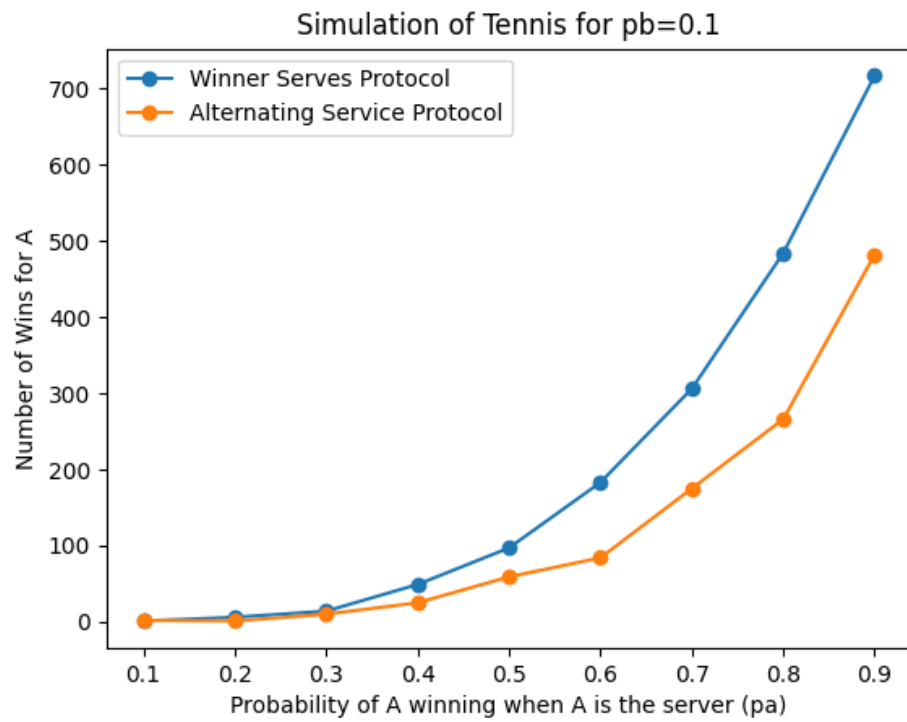


Figure 12: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.1$

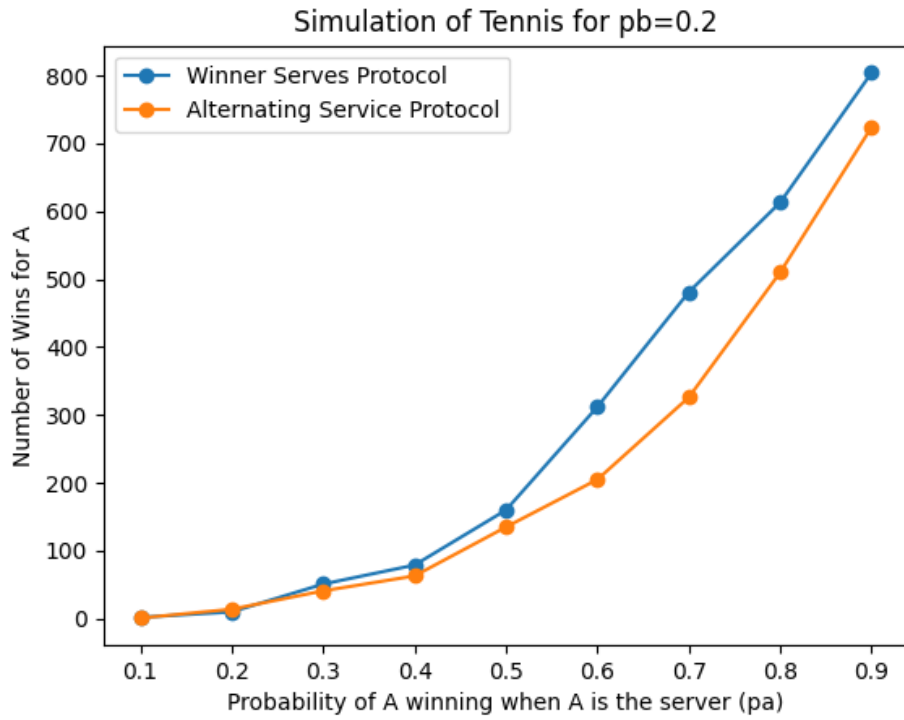


Figure 13: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.2$

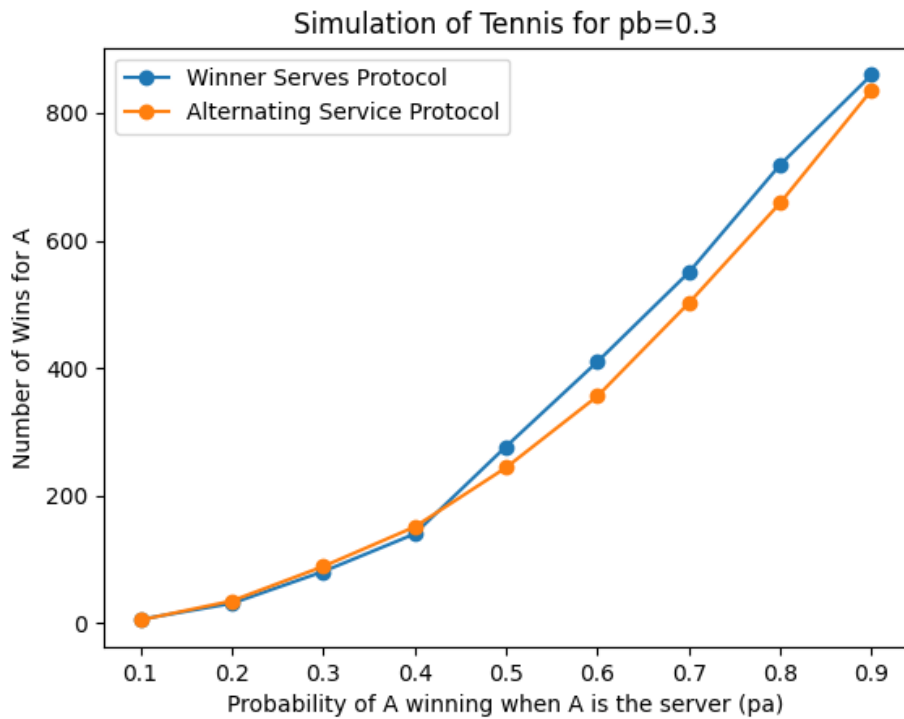


Figure 14: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.3$

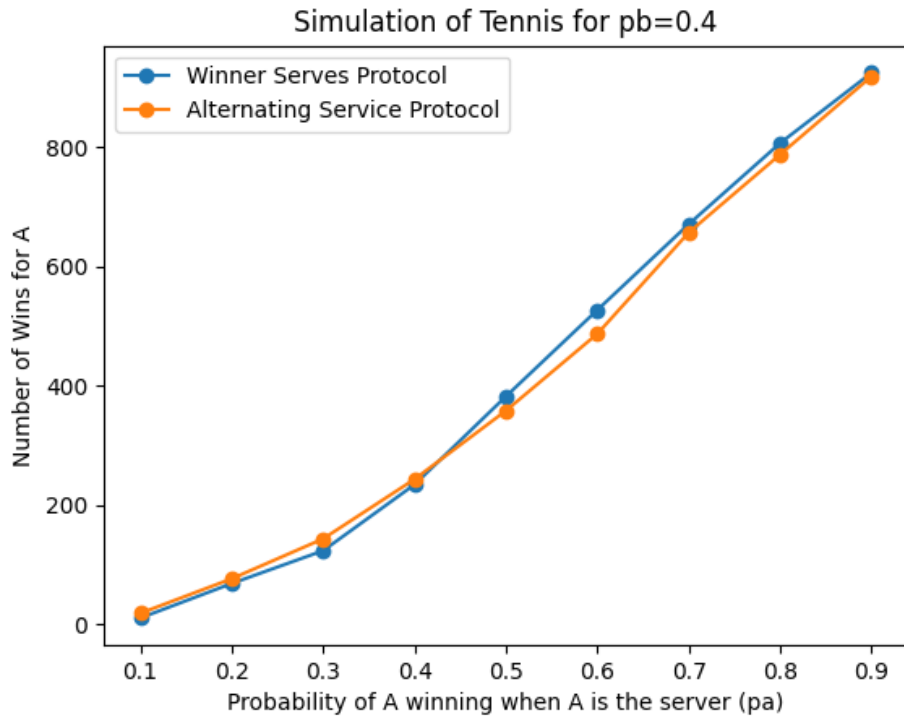


Figure 15: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.4$

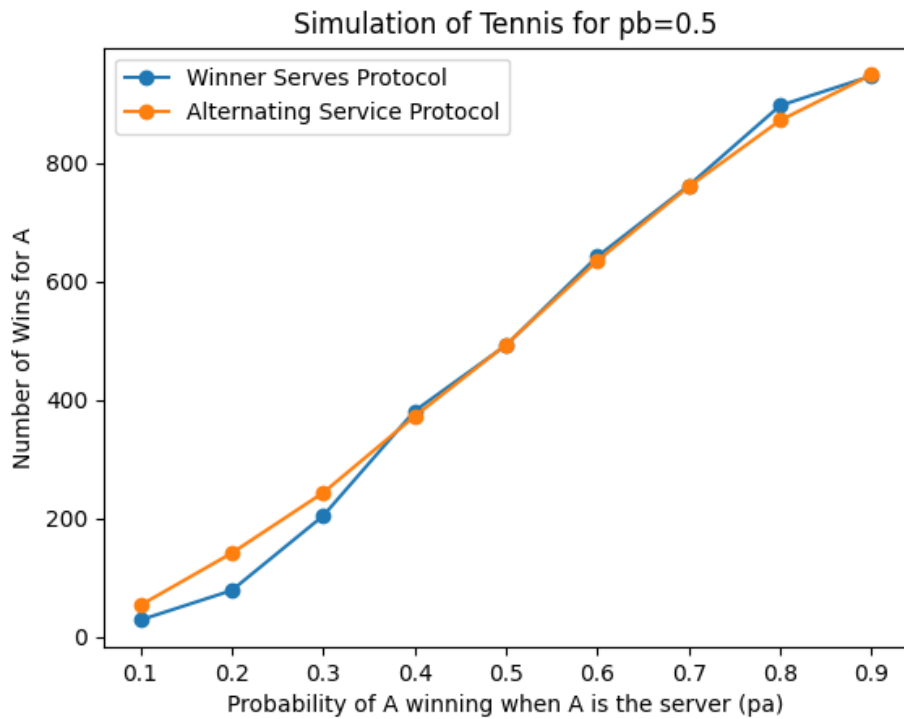


Figure 16: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.5$

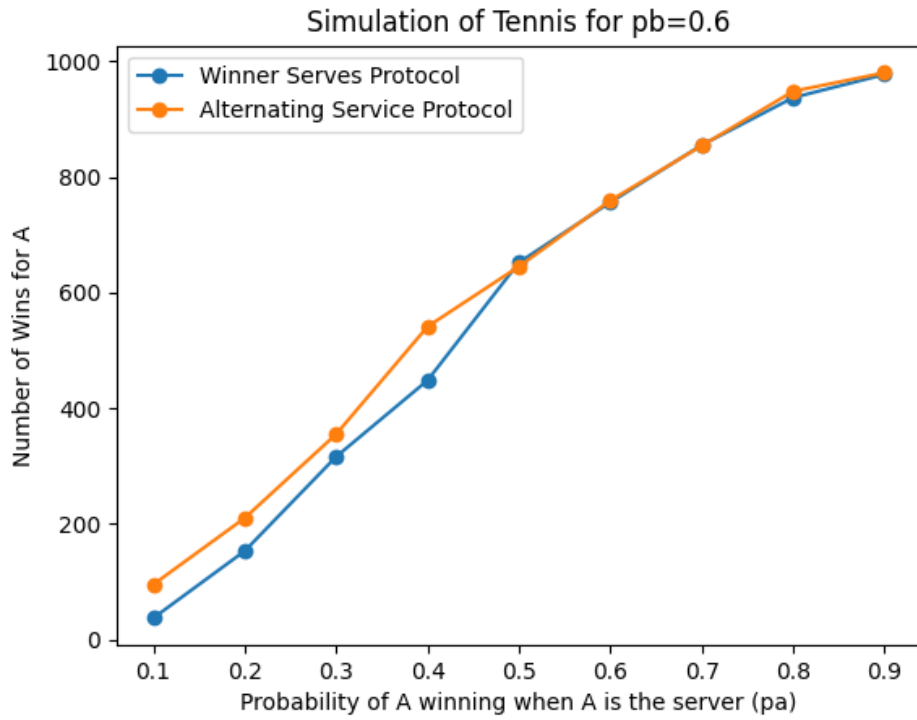


Figure 17: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.6$

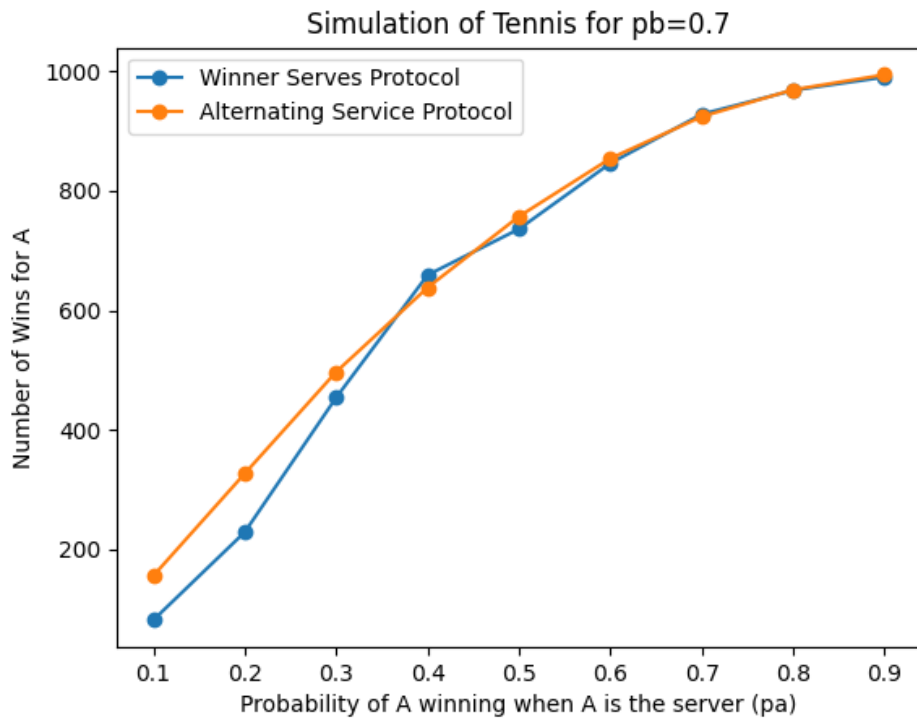


Figure 18: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.7$

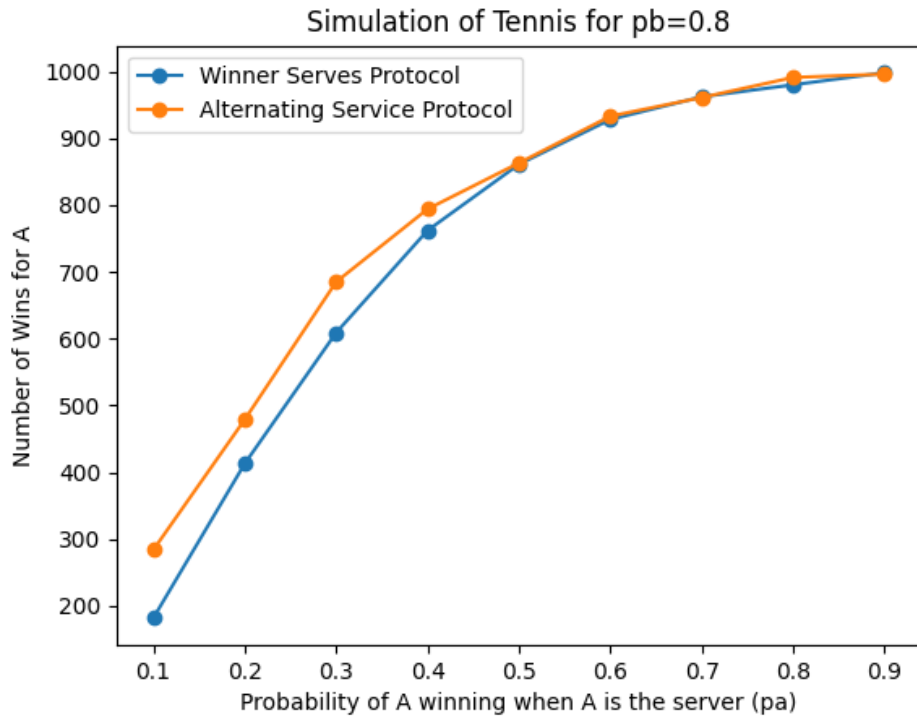


Figure 19: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.8$

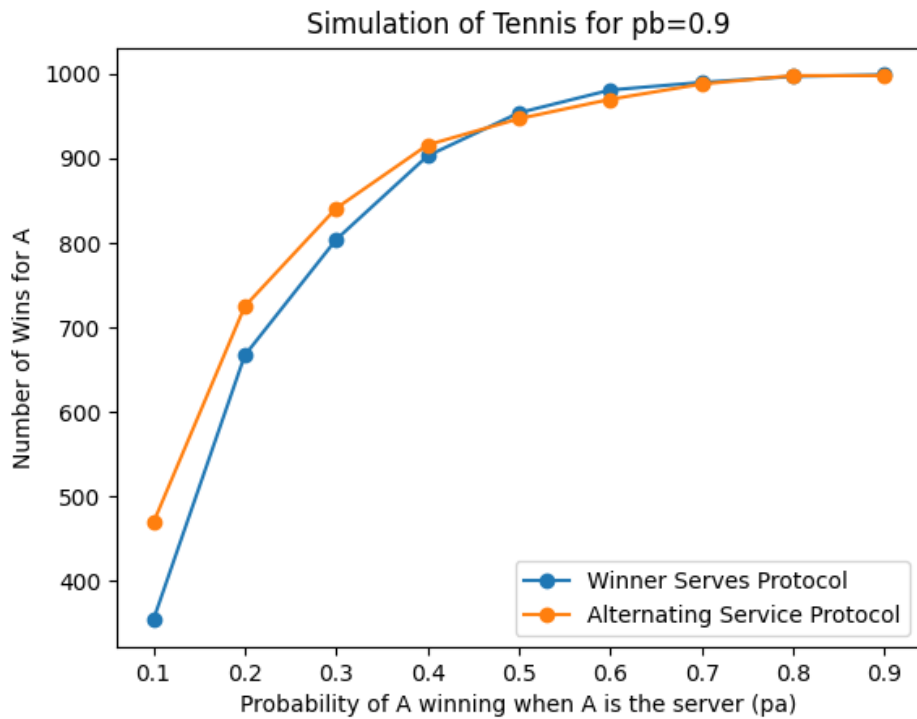


Figure 20: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.9$

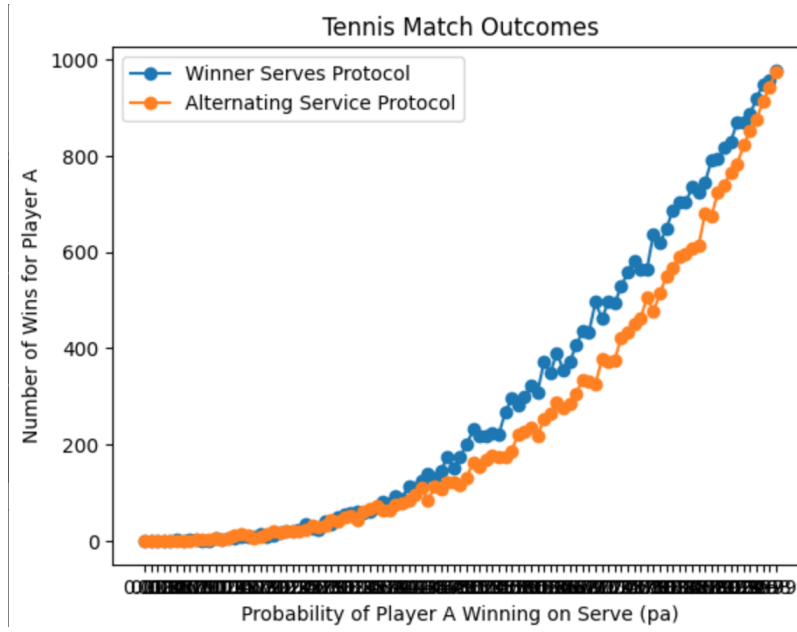


Figure 21: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.2$, but for 100 uniformly spaced values of p_a .

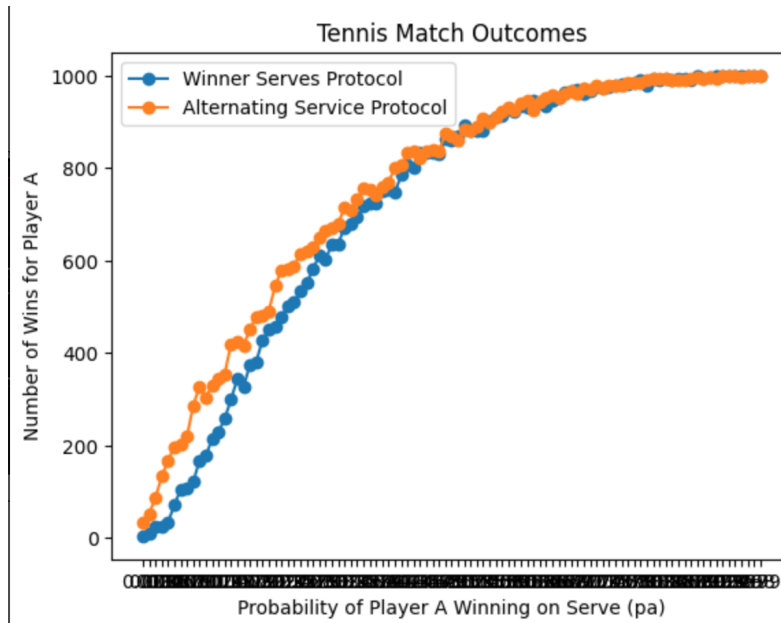


Figure 22: The simulated number of winnings for A, considering Deuce in the matches for $p_b=0.8$, but for 100 uniformly spaced values of p_a . Note that the shape of the parabola inverts compared to the previous parabola for $p_b=0.2$.

3.5 Analytical Deuce Solution

The analytical solution to the game is presented in the form of a Markov chain diagram in an attempt to form a solution.

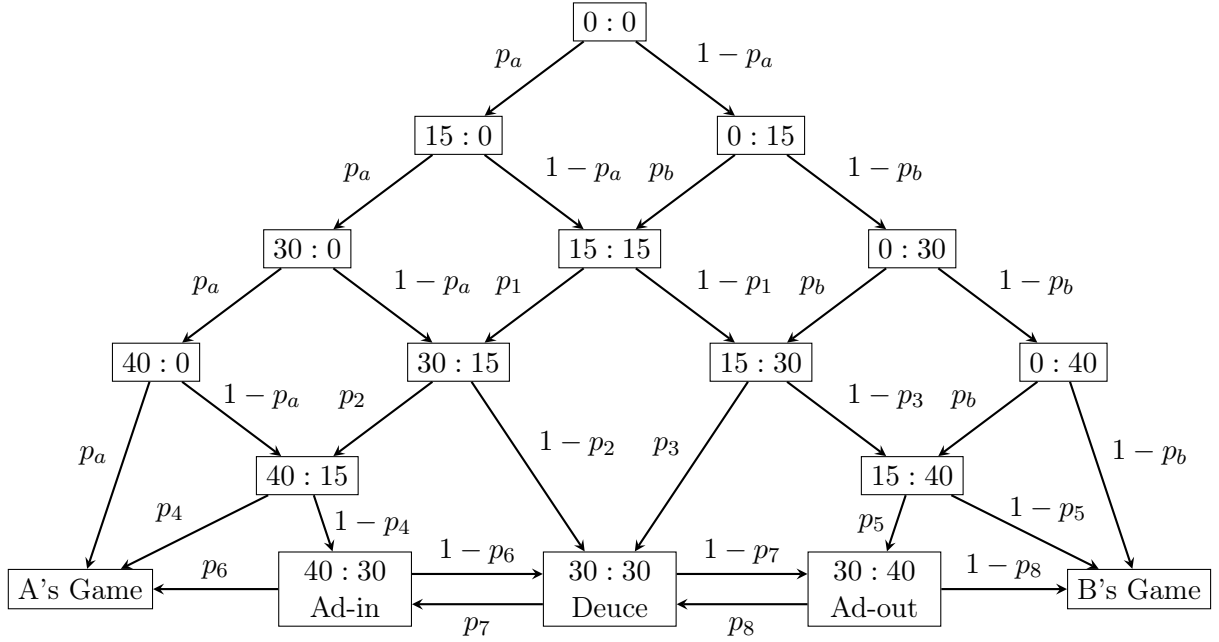


Figure 23: Winner Serves Game

In the figure,

$$p_1 = (1 - p_a) * (p_b) + (p_b) * (p_a) \quad p_2 = (1 - p_a) * (p_b) + (p_1) * (p_a)$$

$$p_3 = (1 - p_1) * (p_b) + (p_b) * (p_a) \quad p_4 = (1 - p_a) * (p_b) + (p_2) * (p_a)$$

$$p_5 = (1 - p_3) * (p_b) + (p_b) * (p_a) \quad p_6 = (1 - p_4) * (p_b) + (p_7) * (p_a)$$

$$p_7 = (1 - p_2) * (p_b) + (p_6) * (p_b) + (p_8) * (p_a) + (p_3) * (p_a)$$

$$p_8 = (1 - p_7) * (p_b) + (p_5) * (p_a)$$

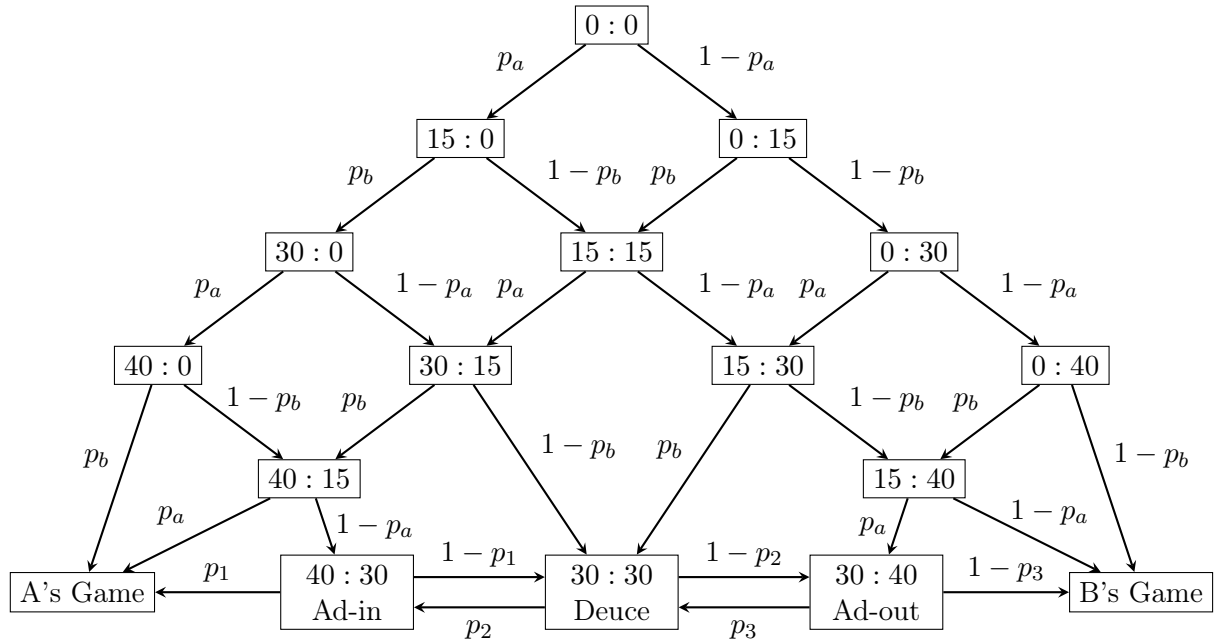


Figure 24: Alternate Serves Game

In the figure,

$$p_1 = (1 - p_a) * (p_b) + (p_2) * (p_b) \quad p_3 = (1 - p_2) * (p_b) + (p_b) * (p_a)$$

$$p_2 = (1 - p_1) * (p_a) + (1 - p_b) * (p_a) + (p_b) * (p_a) + (p_3) * (p_a)$$

- The above Markov chains can be solved by taking the value at [0:0] to be 1 and applying the transitions consecutively to get the final probability of A winning the game.
- A sample [half-solved chain](#) is shown below to illustrate the method to solve them.
- The empty cells can be filled by applying the corresponding transitions to the states and summing them up.
- This helps us find the probability of reaching all of the states in the bottom row of the markov chain. We model a random walk for the 5 states in the bottom of the Markov Chain. We then multiply the probabilities of reaching one of the 4 states (Advantage in, out, Deuce and B's game) and the probability of reaching A's game from each of these 4 states (from the random walk earlier) and add these products to get the final probability of reaching A's game. This calculation gets messy because of the fact that we have different success and failure probabilities at each stage, and ends up with us dealing with messy polynomials. I however tried this for $p_a=p_b=0.5$ and it matched with the simulation.

Github Link of Assignment: <https://github.com/hizhitman/EE6150-Assignment>

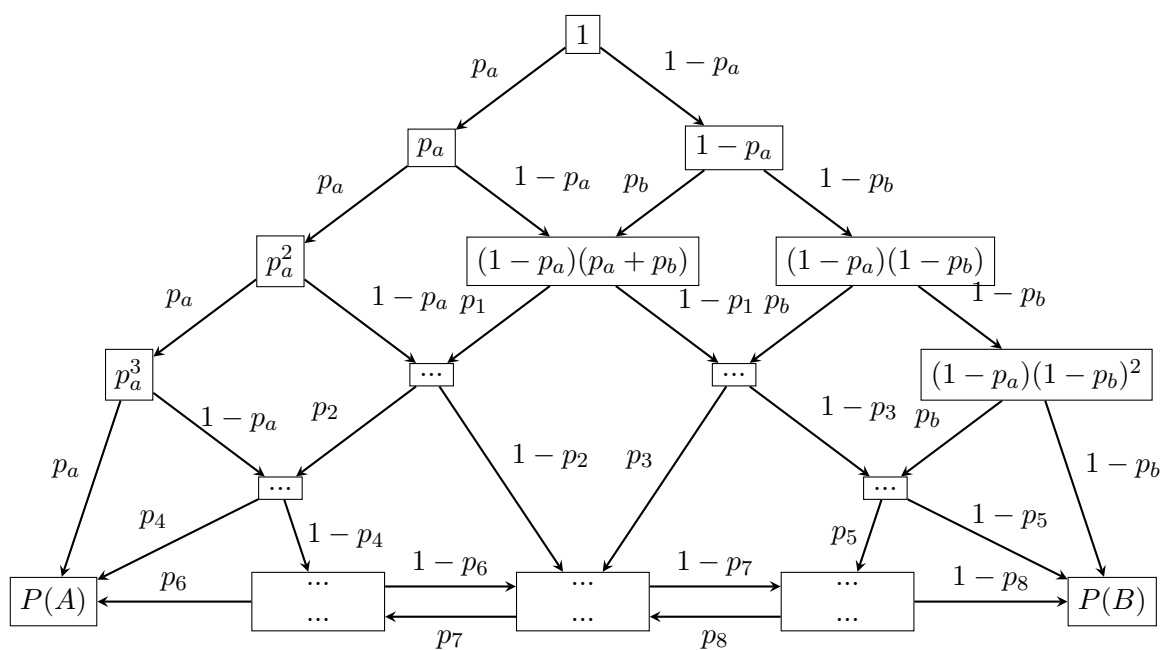


Figure 25: Sample Solution