Probabilities on Latin Squares

by

Anna Long

Submitted in partial fulfillment of the

requirements for Departmental Honors in

the Department of Mathematics

Texas Christian University

Fort Worth, Texas

May 8, 2023

PROBABILITIES ON LATIN SQUARES

Project Approved:

Supervising Professor: Drew Tomlin, PhD

Department of Mathematics

Efton Park, PhD

Department of Mathematics

Liran Ma, PhD

Department of Computer Science

Abstract

A Latin square is an $n \times n$ square that contains n different symbols, often numbers that are arranged so that each symbol appears exactly once in each row and column. In this project, we look at the probability of a random arrangement of symbols being a Latin square. I start with n number of n symbols; for example a 3×3 square will contain the numbers $1, 1, 1, 2, 2, 2, 3, 3, 3$ in a random assortment. Using counting methods and statistical estimation through Python, we discover the proportion of Latin squares to total squares.

Acknowledgments

I want to thank my supervising professor, Dr. Drew Tomlin, for her time spent guiding my research. I would not have been able to complete this project without her insights.

I want to thank Dr. Efton Park for serving on my committee and being a resource for both Dr. Tomlin and me.

Finally, I want to thank Dr. Liran Ma for giving me the foundational skills in Python to complete this project and for serving on my committee.

Reference Guide

1 Definitions

1.1 Latin square

A Latin square is an $n \times n$ square that contains n different symbols that are arranged so that each symbol appears exactly once in every row and every column. Latin squares that are reflections or rotations of each other are considered different squares for my purposes.

1.2 Central limit theorem

The central limit theorem states that given sufficiently large random samples from a population, the average of the sample proportions will yield a normal distribution that estimates the population proportion. Here $np \geq 10$ defines sufficiently large, with n being the sample size and p being the proportion. The random samples must be independent of each other. I am taking a sample proportion of Latin squares to total squares from each random sample of squares, and the mean of these proportions should be the population proportion of Latin squares to random squares. Each new square does not rely on the outcome of the previous squares, so every square selection is independent.

2 Estimation through Python

In order to check my calculations, I used Python to simulate random squares of sizes 2×2, 3×3, and 4×4. By listing the valid numbers that would be used in a square of a certain size, randomizing, and reordering into a square shape, I created random squares that I could test for the criteria that would make it a Latin square. To check for uniqueness of numbers in each row and column, I used lists and sets. In Python, a list can contain any elements, whereas a set can only contain unique elements. With this information, I turned each row and column of the squares into a list and a set and then compared the lengths of each. If the list and the set had the same length, then I knew that that row or column contained only unique values. If every row and column passed this test, then I knew the square must be a Latin square. Counting the squares that met the criteria and the total randomly generated squares, I estimated the proportion of Latin squares to total squares for the three different sizes of squares. By simulating this process hundreds of times, I visualized this estimation through histograms with a mean that is approximately equal to the calculated proportion. Using the central limit theorem and solving $np \geq 10$ for n, where p is my calculated proportion, I found the number of times I would have to randomly generate squares to get an accurate estimation of the proportion of Latin squares to random squares. The mean of these samples is my estimate of the proportion of Latin squares.

2.1 2x2 squares

The 2×2 case of Latin squares is the simplest variety of Latin squares. In my code, I define the symbols and randomly select the values of the square in Listing 1.

```
values = [1,1,2,2]
        a = rd.choice(values)
        values.remove(a)
        b = rd.choice(values)
        values.remove(b)

        c = rd.choice(values)
        values.remove(c)
        d = rd.choice(values)

        ltSq = np.array([[a,b],[c,d]])
```

I checked the square to see if it fit the definition of a Latin square and kept track of the counts. There were two different ways that I checked for Latin squares for the 2×2 case. The first was by simply checking to see if the elements in each row and each column were not equal as seen below in Listing 2.

```
    ltSq = np.array([[a,b],[c,d]])

    if a!=b and a!=c and d!=c and d!=b:
            count +=1
        i+=1
print(count/i)
```

For the second way, I checked for uniqueness of elements within rows and columns through my list/set method mentioned before, as seen in Listing 1.

```
row1 = [a,b]
            srow1 = set(row1)
            row2 = [c,d]
            srow2 = set(row2)
            col1 = [a,c]
            scol1 = set(col1)
            col2 = [b,d]
            scol2 = set(col2)

            if len(row1) != len(srow1):
                i+=1
                #end row 1 check
            elif len(row2) != len(srow2):
                i+=1
            elif len(col1) != len(scol1):
                i+=1
                    #end col 1 check
            elif len(col2) != len(scol2):
                i+=1
            else:
                count+=1
                i+=1
```

My minimum sample size for this case was $n \geq 34$. So, I repeated to estimate the proportion of Latin squares to total random squares and created the histogram with a mean proportion of 0.33323 and a standard deviation of 0.00461 as you can see in Figure 1.

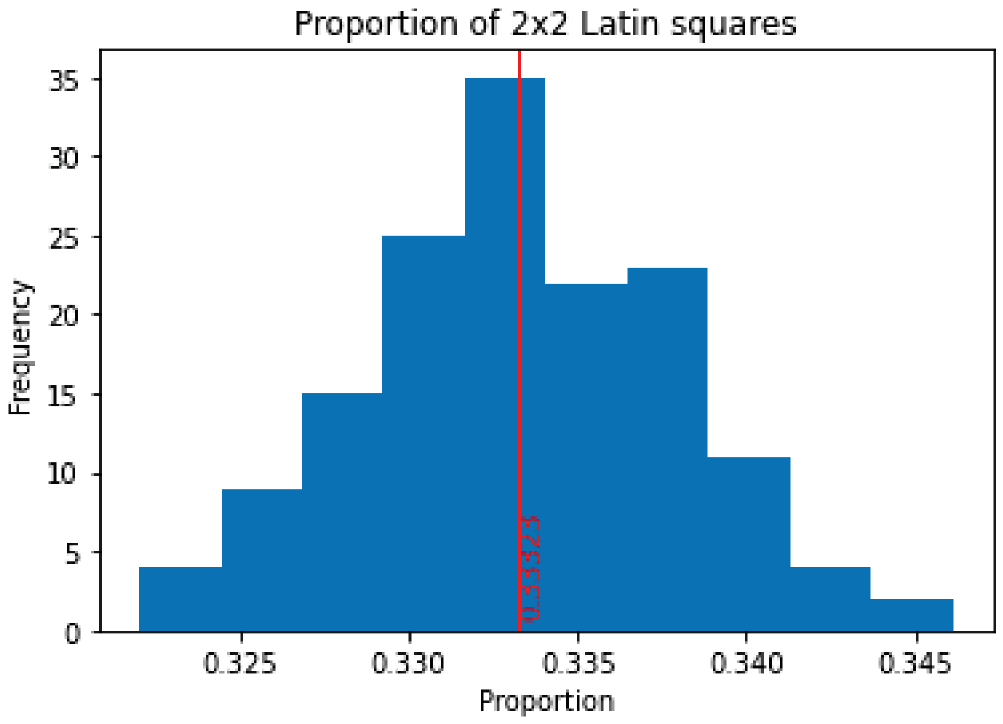

Figure 1: 2×2 Simulation Results

2.2 3x3 squares

I began trying to estimate the proportion of 3×3 Latin squares using the same technique as in Listing 2. I realized that this would require checking uniqueness for all n^2 elements in the square, and that number would quickly get out of hand. This is the point when I began using the list/set method as seen below in Listing 3.

```python
    while i <100000 :#the number of squares i want to make

        values = [1,1,1,2,2,2,3,3,3] #the elligible values for a 3x3 square

        ltSq = np.array([[a,b,c],[d,e,f],[g,h,k]]) #arrange the numbers in a square

        #turn into a set then check the length of the list vs the set
        #since a set cannot have dublicates it checks if the values are unique in each row
    and column

        row1 = [a,b,c]
        srow1 = set(row1)

        row2 = [d,e,f]
        srow2 = set(row2)

        row3 = [g,h,k]
        srow3 = set(row3)

        col1 = [a,d,g]
        scol1 = set(col1)

        col2 = [b,e,h]
        scol2 = set(col2)

        col3 = [c,f,k]
        scol3 = set(col3)

        if len(row1) != len(srow1):
            i+=1
            #end row 1 check
        elif len(row2) != len(srow2):
            i+=1
            #end row 2 check
        elif len(row3) != len(srow3):
            i+=1
            #end row 3 check
        elif len(col1) != len(scol1):
            i+=1
            #end col 1 check
        elif len(col2) != len(scol2):
            i+=1
            #end col 2 check
        elif len(col3) != len(scol3):
            i+=1
            #end col 3 check

        else:
            count+=1
            i+=1
```

Using the central limit theorem, I checked that I used a large enough sample size for an accurate estimate. I used $n = 100,000$ and solving $np \geq 10$ with $p = 0.00714$ from my calculations, I only needed $n \geq 1,400$. Between my sample size and number of samples, I checked $15,000,000$ random squares. I estimated the 3×3 Latin squares proportion to be 0.00715 with a standard deviation of 0.00025, pictured in Figure 2.

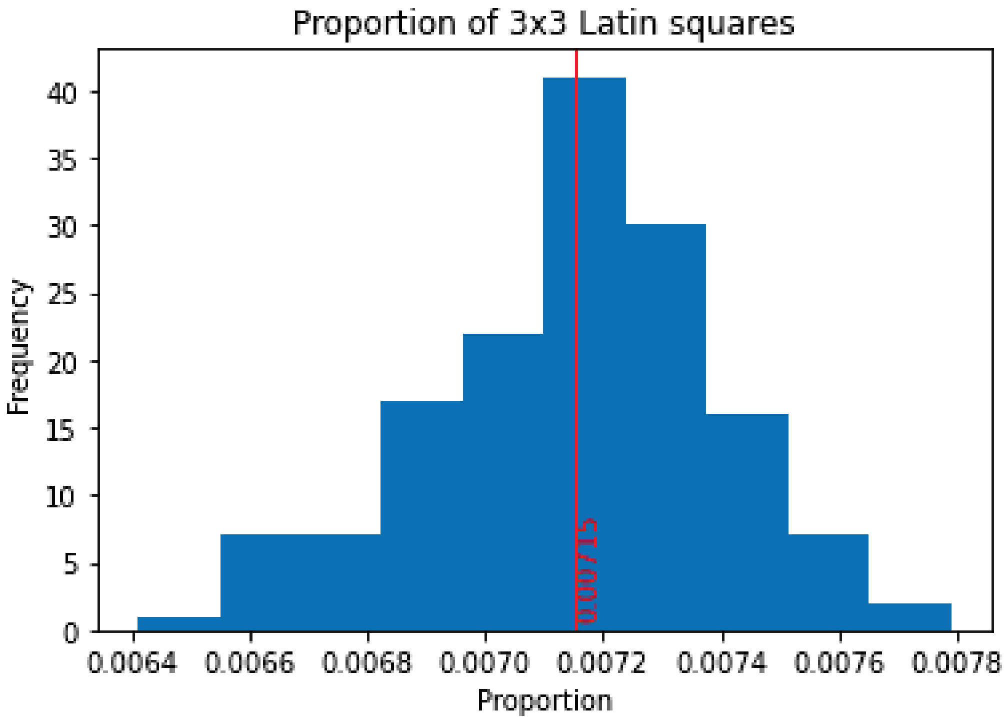

Figure 2: 3×3 Simulation Results

2.3 4x4 squares

Even with my updated technique for checking if a square is a Latin square, the larger square started causing issues for my program. The number of rows and columns that needed checking grew slightly, but the accuracy of my estimation was failing short due to the proportion that I was expecting from my theoretical calculations. Referencing the central limit theorem and solving $np \geq 10$ with $p = 9.06 \times 10^{-6}$, I found $n \geq 1,094,811$ or the size of my samples I would have to randomly generate for 4×4 squares to get an accurate estimation of the proportion of Latin squares to random squares. I selected $n = 1.1$ million, to round up and be sure that I had a large enough sample size. This number was so large that my program would take almost 45 minutes to run. Due to these limitations, 4×4 was the largest size square that I attempted to create and sample in Python. The steps for randomizing this many squares is shown in Listing 4.

```
    while i <1100000: #using 1.1mil squares to find one p hat value

        values = [1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4]

        a = rd.choice(values)
        values.remove(a)
        b = rd.choice(values)
        values.remove(b)
        c = rd.choice(values)
        values.remove(c)
        d = rd.choice(values)
        values.remove(d)

        e = rd.choice(values)
        values.remove(e)
        f = rd.choice(values)
        values.remove(f)
        g = rd.choice(values)
        values.remove(g)
        h = rd.choice(values)
        values.remove(h)

        j = rd.choice(values)
        values.remove(j)
        k = rd.choice(values)
        values.remove(k)
        l = rd.choice(values)
        values.remove(l)
        m = rd.choice(values)
        values.remove(m)

        n = rd.choice(values)
        values.remove(n)
        o = rd.choice(values)
        values.remove(o)
        p = rd.choice(values)
        values.remove(p)
        q = rd.choice(values)

        ltSq = np.array([[a,b,c,d],[e,f,g,h],[j,k,l,m],[n,o,p,q]])

        #turn into a set then check the length of the list vs the set
        row1 = [a,b,c,d]
        srow1 = set(row1)

        row2 = [e,f,g,h]
        srow2 = set(row2)

        row3 = [j,k,l,m]
        srow3 = set(row3)

        row4 = [n,o,p,q]
        srow4 = set(row4)

        col1 = [a,e,j,n]
        scol1 = set(col1)

        col2 = [b,f,k,o]
        scol2 = set(col2)

        col3 = [c,g,l,p]
        scol3 = set(col3)

        col4 = [d,h,m,q]
        scol4 = set(col4)

        if len(row1) != len(srow1):
            i+=1
            #end row 1 check
        elif len(row2) != len(srow2):
            i+=1
            #end row 2 check
        elif len(row3) != len(srow3):
            i+=1
            #end row 3 check
        elif len(row4) != len(srow4):
            i+=1
            #end row 4 check
        elif len(col1) != len(scol1):
            i+=1
            #end col 1 check
        elif len(col2) != len(scol2):
            i+=1
            #end col 2 check
        elif len(col3) != len(scol3):
            i+=1
            #end col 3 check
        elif len(col4) != len(scol4):
            i+=1
            #end col 4 check

        else:
            count+=1
```

```
                i+=1
```

I discovered the estimated proportion of 4×4 Latin squares to be 9.06×10^{-6} with a standard deviation of 2.9×10^{-6}. The average printed on Figure 3 is rounded to 5 decimal places, therefore it rounded up to 1×10^{-5}.

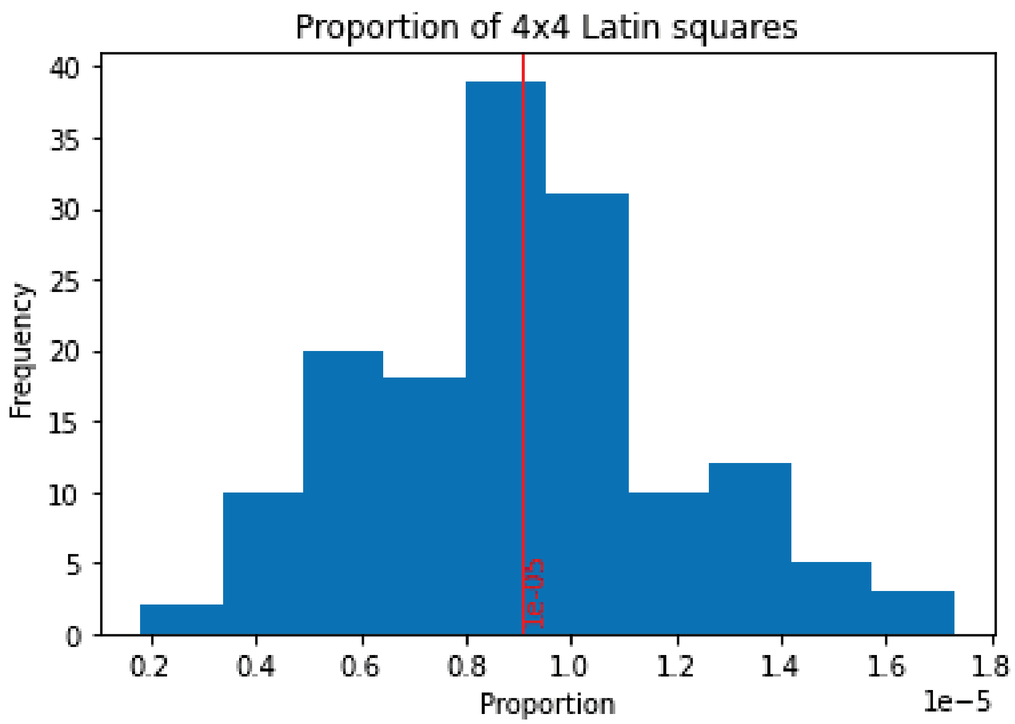

Figure 3: 4×4 Simulation Results

3 Pattern counting

In order to count the Latin squares of various sizes by hand, I counted the different patterns of each number that were possible. Beginning with the small squares and then working my way to larger and larger squares, I counted the patterns of 1's then the patterns of 2's and so on. By placing a number row by row, I counted the different possible options for the number in the next row. I discovered that I was able to begin with a 1 in the top left corner for every square, and then multiply my final number by n to get to total number of Latin squares of size $n \times n$. This yielded the correct number of squares because it's the equivalent of swapping all the 1's and 2's or 1's and 3's until all values have been swapped with the 1's pattern, which would be n swapped patterns.

1	2	3
2	3	1
3	1	2

2	1	3
1	3	2
3	2	1

3	2	1
2	1	3
1	3	2

Figure 4: 3×3 Number Swapping

3.1 Pattern of 1's

Counting the patterns of 1's begins with $\binom{n}{1}$ choices in the first row, then $\binom{n-1}{1}$ for the second row, this continues until $\binom{1}{1}$ for the last row or $\prod_{i=0}^{n} \binom{n-1}{1} = n!$. Only working with squares with a 1 in the top left corner, allowed me to deal with $(n-1)!$ patterns of 1's instead of $n!$ patterns of 1's, which ended up being significantly fewer squares for the larger sizes.

3.2 Pattern of 2's

Counting the patterns of 2's for a fixed pattern of 1's consisted of drawing many examples for each size n and identifying relationships and patterns within the squares. I was again able to use symmetry to my advantage for the larger squares and begin with a 2 placed in the top row then multiplying by $(n-1)$ at the end to account for the choices of that first placement. Finding the number of patterns of 2's for each pattern of 1's within each size square, the counts became apparent. I checked my numbers with different patterns of 1's and with different placements of the first 2 to verify that I was justified in using symmetry to count. I wrote out full equations for different sizes of patterns of 2's, for example counting the 4×4 patterns of 2's for the pattern on 1's on the main diagonal is: $3(3\frac{1}{3} + 2\frac{2}{3}(2\frac{1}{2} + 1\frac{1}{2})) = 9$. Doing this for different patterns of 1's, created different equations, but each pattern of 1's resulted in 9 patterns of 2's. A visual example of my technique is shown in Figure 5:

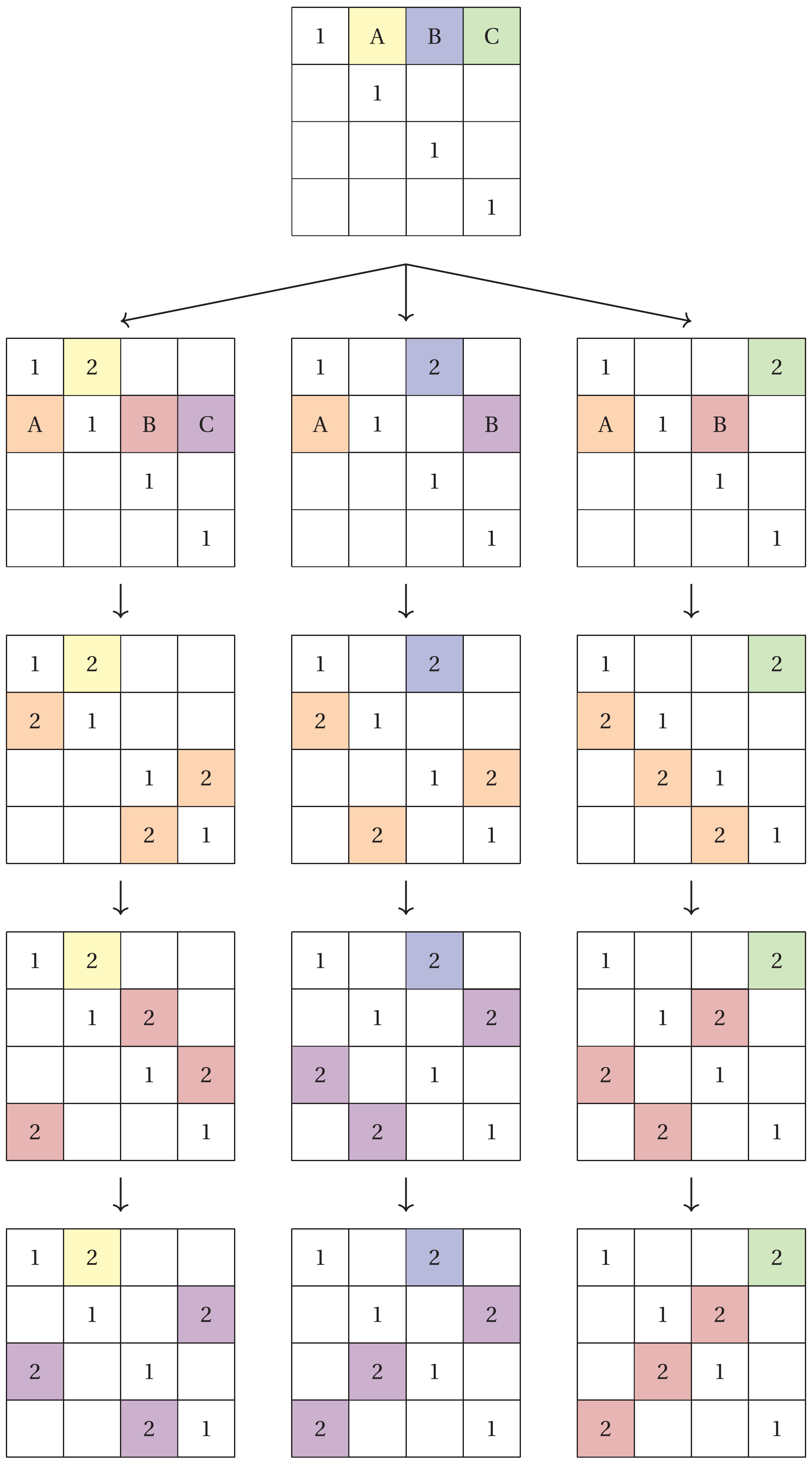

Figure 5: 4×4 Patterns of 2's

I finally noticed a pattern after finding the patterns of 2's for the Latin squares up to 6×6. I began with a 2 placed in the top row along with my selected pattern of 1's. Then, I placed my second 2 and wrote out my expression for the remaining 2s and did this for each placement of the second 2. Some of these expressions were exactly my expressions for some of the patterns of 2's on 4×4 Latin squares. Summing these expressions, I discovered that by adding the previous 2 counts of patterns of 2's and then multiplying by $(n-1)$ I would get the count of patterns of 2's for each pattern of 1's for the current sized square. So with the base cases $t_2 = 1$ and $t_3 = 2$ then, $t_n = (n-1)(t_{n-2} + t_{n-1})$ for $n \geq 4$ where t_n is the number of patterns of 2's for a pattern of 1's for an $n \times n$ Latin square.

3.3 Pattern of 3's

I only needed to count the patterns of 3's for 4×4 and 5×5 Latin squares. These numbers started to differ from the previous pattern counting because different patterns of 1's and 2's resulted in different numbers of patterns of 3's. In the 4×4 case, the number of patterns of 3's was also the number of Latin squares from each pattern of 1's and 2's. I discovered that $\frac{2}{3}$ of the patterns of 1's and 2's created 2 unique Latin squares, one for each placement of the first 3 in the first row. As seen in Table 1, there is no other valid pattern of 3's with the same placement of 3 in the first row.

1	2		3
	1	3	2
2	3	1	
3		2	1

Table 1: 4×4 Pattern of 3's

However, $\frac{1}{3}$ of the patterns of 1's and 2's resulted in 4 unique Latin squares. The four Latin squares from a single pattern of 2's happened when numbers paired up so that the placement of two of the 3s does not affect the placement of the other two 3s. Table 2 shows the possible pairings of 3s where the purple 3s are paired up so that the blue squares represent a Latin square and the red squares represent a different Latin square. The purple 3s could be moved to the gray squares, and the same red and blue options are available for the other 3s. So 2^2 or 4 Latin squares were made by the single pattern of 2's.

1	3	2	▨
🟦	2	🟥	1
🟥	1	🟦	2
2	▨	1	3

Table 2: 4×4 pairing

The 5×5 Latin squares also exhibited a similar pairing phenomenon in the patterns of 3's. Since 5 is not evenly divisible by 2, these patterns could only split up into a pair and a trio. By placing my first 2 in the first row, I was able to work with $\frac{44}{4} = 11$ patterns of 1's and 2's to to calculate the subsequent patterns of 3's. I drew all the patterns of 3's for each of the 11 patterns of 1's and 2's and found that some had 12 patterns of 3's and some had 13 patterns of 3's. The patterns that created only 12 patterns of 3's included all the squares where the pairing occurred in the 1's and 2's, as seen in Table 3. These patterns all created 2 Latin squares when the 4s and 5s were filled in.

1	2	3		
2	1			3
3		1	2	
	3		1	2
		2	3	1

Table 3: 5×5 Pairing in 1's

The squares that had 13 patterns of 3's contained the squares where the pairing occurred in the 3's, 4's, 5's or not at all. This collection of squares resulted in 4 Latin squares $\frac{5}{13}$ of the time, and 2 Latin squares the remaining $\frac{8}{13}$ of the time. Table 4 is an example of a pattern that can create 4 different Latin squares.

1	2		3	
3	1	2		
		1	2	3
	3		1	2
2		3		1

Table 4: 5×5 Pairing in 3's

4 Conclusions

There are 6 possible 2×2 configurations with 2 of those being Latin squares with two different patterns of 1's. For 3×3 squares, the random configurations go up to $1,680$ with only 12 of these being Latin squares, consisting of 2 patterns of 2's for each of the 6 patterns of 1's. There are 576 different 4×4 Latin squares out of the $63,063,000$ random squares that are made up by 24 patterns of 1's, 9 patterns of 2's for each pattern of 1's, and $\frac{1}{3}$ of those resulting in 4 patterns of 3's and the remaining $\frac{2}{3}$ resulting in 2 patterns of 3's. There are $623,360,750,000,000$ random 5×5 squares with $161,280$ possible Latin squares with 120 patterns of 1's, 44 patterns of 2's for each, $\frac{5}{11}$ of those patterns of 2's result in 12 patterns of 3's and $\frac{6}{11}$ result in 13. When the pattern of 2's results in 12 patterns of 3's, then each of those could make 2 Latin squares with the remaining numbers filled in, but the ones that create 13 patterns of 3's then $\frac{5}{13}$ can make 4 possible Latin squares and the remaining ones can only make 2. While it was possible to find exactly how many patterns of 3's there are for each pattern of 1's and 2's, I could not write an independent equation for the patterns of 4's on 5×5 Latin squares because it depends on the specific pattern of 3's. In Table 5, you can multiply the P_n columns across a row to get the total number of Latin squares, except for $n = 5$ where you have to omit the P_3 column because that value is already incorporated in the P_4 value. P_1 refers to the patterns of 1's, etc. and LS refers to Latin squares. The proportions that I refer to throughout are the LS number divided by the total number.

n	P_1	P_2	P_3	P_4	P_5	LS	Total
2	2	1	-	-	-	2	6
3	6	2	1	-	-	12	$1,680$
4	24	9	$\frac{2}{3}2 + \frac{1}{3}4$	1	-	576	63 million
5	120	44	$\frac{5}{11}12 + \frac{6}{11}13$	$\frac{336}{11}$	1	$161,280$	623 trillion

Table 5: Summary Table.

I noticed that beginning with the 4×4 Latin squares, two rows and two columns could pair up at the $(n-2)$ level, and this is what created the 4 resulting Latin squares rather than 2. Bringing this into higher dimensions would allow for even more resulting Latin squares from each pattern of the $(n-2)$ number. A 6×6 Latin square has the ability to have 3 of these pairings, so some patterns of 4's could create 8, or 2^3 Latin squares from a single pattern. Generally, this would be $2^{\lfloor \frac{n}{2} \rfloor}$ where $\lfloor \frac{n}{2} \rfloor$ is the largest integer less than or equal to $\frac{n}{2}$.

5 Future Work

The next steps in this research would be finding the generic formula for counting the patterns of 3's for an $n \times n$ Latin square. I would continue looking at these patterns until a generic formula for counting Latin squares of a certain size is found. I believe that looking at the patterns within Latin squares will tell a lot about how they are formed, such as the pairing technique I mentioned. Given more computing power, I would also estimate these proportions of larger Latin squares through statistical estimation on Python. I might also create a program that can look at the patterns within the Latin squares so that trends might be able to be drawn for certain patterns.

6 Python code

Listing 1: 2×2 Latin Square Code

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Oct  2 17:10:05 2022

@author: annam
"""

import numpy as np
import random as rd
import matplotlib.pyplot as plt
import statistics as st


pHat = 0 #will be used to save each p hat value
pHatList = [] #will be used to save all the p hat values
summ = 0 #will be used to find the mean

for j in range(150): #plotting values

    i = 0 #the total number of squares created
    count = 0 #the count of latin squares
    while i<10000:
        values = [1,1,2,2]
        a = rd.choice(values)
        values.remove(a)
        b = rd.choice(values)
        values.remove(b)

        c = rd.choice(values)
        values.remove(c)
        d = rd.choice(values)

        ltSq = np.array([[a,b],[c,d]])

        row1 = [a,b]
        srow1 = set(row1)
        row2 = [c,d]
        srow2 = set(row2)
        col1 = [a,c]
        scol1 = set(col1)
        col2 = [b,d]
        scol2 = set(col2)

        if len(row1) != len(srow1):
            i+=1
            #end row 1 check
        elif len(row2) != len(srow2):
            i+=1
        elif len(col1) != len(scol1):
            i+=1
                #end col 1 check
        elif len(col2) != len(scol2):
            i+=1
        else:
            count+=1
            i+=1
    pHat = count/i
    pHatList.append(pHat)
    summ+=pHat

average = summ/150
print(average)
std = st.stdev(pHatList)
print(std)

graph = plt.hist(pHatList)
plt.title("Proportion of 2x2 Latin squares")
plt.xlabel("Proportion")
plt.ylabel("Frequency")
plt.axvline(average, color="red", linestyle = "solid", linewidth=1)
plt.text(average, 1.1, round(average, 5), color="red", rotation=90)
plt.show()



#the probabilities are consistent with my calculations of 1/3 being latin squares
```

Listing 2: 2×2 Latin Square Original Method

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  8 15:26:59 2022

@author: annam
"""

import random as rd
import numpy as np


i = 0
count = 0
while i <10000 :

    values = [1,1,2,2]

    a = rd.choice(values)
    values.remove(a)
    b = rd.choice(values)
    values.remove(b)
    c = rd.choice(values)
    values.remove(c)
    d = rd.choice(values)

    ltSq = np.array([[a,b],[c,d]])

    if a!=b and a!=c and d!=c and d!=b:
        count +=1
    i+=1
print(count/i)
```

Listing 3: 3×3 Latin Square Code

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Thu Jan 19 15:02:29 2023

@author: annam
"""


import random as rd
import numpy as np
import matplotlib.pyplot as plt
import statistics as st

pHat = 0 #will be used to save each p hat value
pHatList = [] #will be used to save all the p hat values
summ = 0 #will be used to find the mean

for j in range(150): #plotting values

    i = 0 #the total number of squares created
    count = 0 #the count of latin squares
    while i <100000 :#the number of squares i want to make

        values = [1,1,1,2,2,2,3,3,3] #the elligible values for a 3x3 square

        #select a value and then remove it from the elligible list until all numbers have
    been selected
        a = rd.choice(values)
        values.remove(a)
        b = rd.choice(values)
        values.remove(b)
        c = rd.choice(values)
        values.remove(c)
        d = rd.choice(values)
        values.remove(d)
        e = rd.choice(values)
        values.remove(e)
        f = rd.choice(values)
        values.remove(f)
        g = rd.choice(values)
        values.remove(g)
        h = rd.choice(values)
        values.remove(h)
        k = rd.choice(values)

        ltSq = np.array([[a,b,c],[d,e,f],[g,h,k]]) #arrange the numbers in a square

        #turn into a set then check the length of the list vs the set
        #since a set cannot have dublicates it checks if the values are unique in each row
    and column

        row1 = [a,b,c]
        srow1 = set(row1)

        row2 = [d,e,f]
        srow2 = set(row2)

        row3 = [g,h,k]
        srow3 = set(row3)

        col1 = [a,d,g]
        scol1 = set(col1)

        col2 = [b,e,h]
        scol2 = set(col2)

        col3 = [c,f,k]
        scol3 = set(col3)

        if len(row1) != len(srow1):
            i+=1
            #end row 1 check
        elif len(row2) != len(srow2):
            i+=1
            #end row 2 check
        elif len(row3) != len(srow3):
            i+=1
            #end row 3 check
        elif len(col1) != len(scol1):
            i+=1
            #end col 1 check
        elif len(col2) != len(scol2):
            i+=1
            #end col 2 check
        elif len(col3) != len(scol3):
            i+=1
            #end col 3 check

        else:
            count+=1
            i+=1
    pHat = count/i
    pHatList.append(pHat)
    summ+=pHat

average = summ/150
print(average)
std = st.stdev(pHatList)
print(std)

graph = plt.hist(pHatList)
plt.title("Proportion of 3x3 Latin squares")
plt.xlabel("Proportion")
plt.ylabel("Frequency")
plt.axvline(average, color="red", linestyle = "solid", linewidth=1)
plt.text(average, 1.1, round(average, 5), color="red", rotation=90)
plt.show()
```

Listing 4: 4×4 Latin Square Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 26 11:54:34 2023

@author: annam
"""


import random as rd
import numpy as np
import matplotlib.pyplot as plt
import statistics as st


pHat = 0 #will be used to save each p hat value
pHatList = [] #will be used to save all the p hat values
summ = 0 #will be used to find the mean

for j in range(150): #plotting values
    i = 0
    count = 0

    while i <1100000: #using 1.1mil squares to find one p hat value

        values = [1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4]

        a = rd.choice(values)
        values.remove(a)
        b = rd.choice(values)
        values.remove(b)
        c = rd.choice(values)
        values.remove(c)
        d = rd.choice(values)
        values.remove(d)

        e = rd.choice(values)
        values.remove(e)
        f = rd.choice(values)
        values.remove(f)
        g = rd.choice(values)
        values.remove(g)
        h = rd.choice(values)
        values.remove(h)

        j = rd.choice(values)
        values.remove(j)
        k = rd.choice(values)
        values.remove(k)
        l = rd.choice(values)
        values.remove(l)
        m = rd.choice(values)
        values.remove(m)

        n = rd.choice(values)
        values.remove(n)
        o = rd.choice(values)
        values.remove(o)
        p = rd.choice(values)
        values.remove(p)
        q = rd.choice(values)

        ltSq = np.array([[a,b,c,d],[e,f,g,h],[j,k,l,m],[n,o,p,q]])

        #turn into a set then check the length of the list vs the set
        row1 = [a,b,c,d]
        srow1 = set(row1)

        row2 = [e,f,g,h]
        srow2 = set(row2)

        row3 = [j,k,l,m]
        srow3 = set(row3)

        row4 = [n,o,p,q]
        srow4 = set(row4)

        col1 = [a,e,j,n]
        scol1 = set(col1)

        col2 = [b,f,k,o]
        scol2 = set(col2)

        col3 = [c,g,l,p]
        scol3 = set(col3)

        col4 = [d,h,m,q]
        scol4 = set(col4)

        if len(row1) != len(srow1):
            i+=1
            #end row 1 check
        elif len(row2) != len(srow2):
            i+=1
            #end row 2 check
        elif len(row3) != len(srow3):
            i+=1
            #end row 3 check
        elif len(row4) != len(srow4):
            i+=1
            #end row 4 check
        elif len(col1) != len(scol1):
            i+=1
            #end col 1 check
        elif len(col2) != len(scol2):
            i+=1
            #end col 2 check
        elif len(col3) != len(scol3):
            i+=1
            #end col 3 check
        elif len(col4) != len(scol4):
            i+=1
            #end col 4 check

        else:
            count+=1
            i+=1

    pHat = count/i
    pHatList.append(pHat)
    summ+=pHat


average = summ/150
print(average) #this number should be close to the 9x10^-6 proportion
std = st.stdev(pHatList)
print(std)

graph = plt.hist(pHatList)
plt.title("Proportion of 4x4 Latin squares")
plt.xlabel("Proportion")
plt.ylabel("Frequency")
plt.axvline(average, color="red", linestyle = "solid", linewidth=1)
plt.text(average, 1.1, round(average, 5), color="red", rotation=90)
plt.show()
```