A Study in Derangements

Bharath Variar
2019B5A70930H

CS F424: Applied Stochastic Processes
Department of Mathematics
BITS Pilani, Hyderabad Campus
September - October 2022

Contents

1 Expected value of consecutive heads for a fair coin tossed 100 times

```python
import numpy as np
from matplotlib import pyplot as plt
from collections import Counter


def coin_flip(num_flips=100, probability=0.5):
    '''
    Flip coin num_flips times with probability for each side = 'probability'

    returns: array of size 1 x num_flips with output of each flip
    '''
    result = np.random.binomial(n=1, p=probability, size=(1, num_flips))
    return np.where(result == 1, 'H', 'T')[0]


def longest_heads(results):
    '''
    Input: 1-D array of results from coin toss

    Returns: Maximum consecutive occurrances of heads
    '''
    max_heads = 0
    counter = 0
    for i in range(len(results)):
        if (results[i] == 'H'):
            counter += 1
        else:
            if (counter > max_heads):
                max_heads = counter
            counter = 0
    return max_heads


def plot_results(experiment_results):
    '''
    Input: Takes in array of lengths of longest substring of heads in an experiment of 100 coin flips

    THe function plots a bar graph of frequency of maximum length of consecutive heads, and also shoes the minimum, maximum and the mean values of the length of these substrings.

```

```python
    Returns: None
    '''
    average = np.mean(experiment_results)
    max_head_chain = np.max(experiment_results)
    min_head_chain = np.min(experiment_results)
    hist_dict = Counter(experiment_results)
    indices = list(hist_dict.keys())
    values = list(hist_dict.values())
    fig, ax = plt.subplots()
    bars = ax.bar(indices, values, width=0.5, color='purple')
    ax.bar_label(bars)
    plt.xlabel("Longest Head Chain")
    plt.text(
        9, 2300, f'Average Longest Head Chain: {average}\nLongest Head Chain: {max_head_chain}\nShortest Head Chain: {min_head_chain}')
    plt.xticks(np.arange(0, max_head_chain + 2, step=1))
    plt.ylabel("Frequency")
    plt.title("Longest chain of heads in 10,000 experiments of 100 flips")
    plt.show()
    return


experiment_results = []
for i in range(10000):
    experiment_results.append(longest_heads(coin_flip()))

plot_results(experiment_results=experiment_results)
```

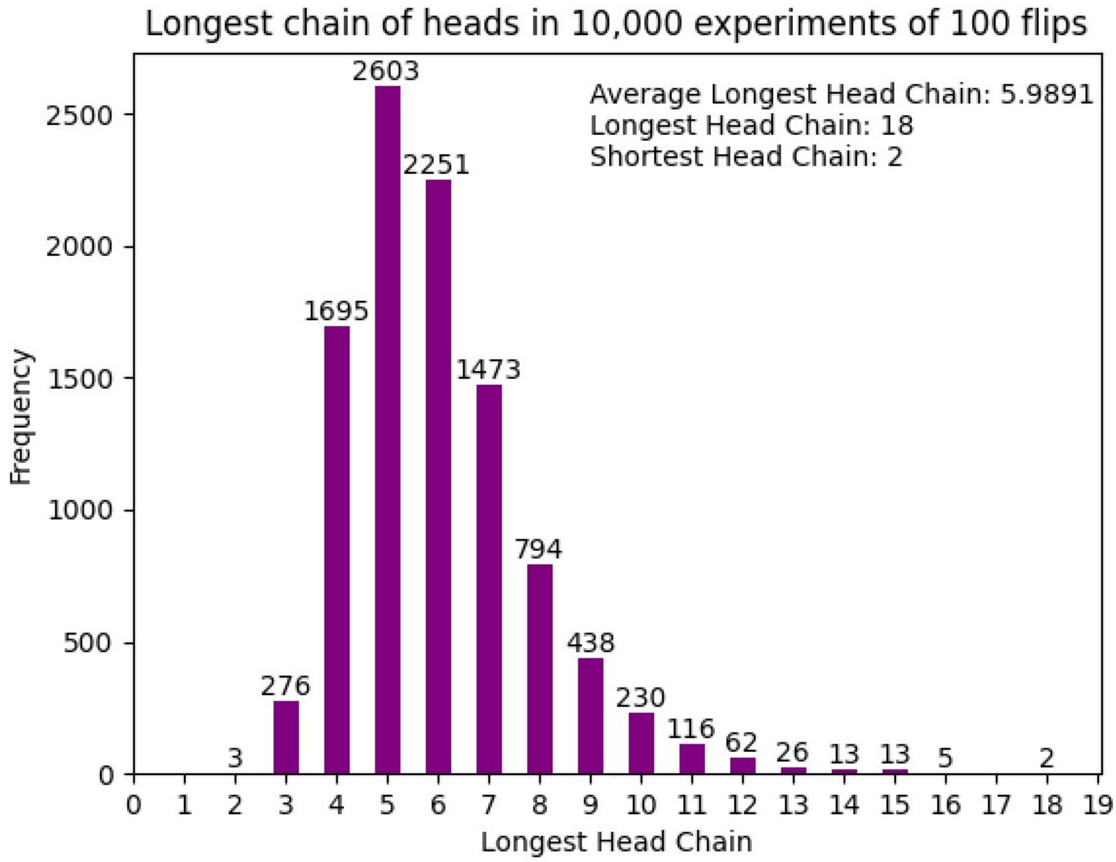

Figure 1: Longest chain of heads over 10,000 runs of 100 tosses

2 Probability of derangements in N items

2.1 What is a derangement

A derangement of a collection is defined as a permutation in which none of the elements in the collection appear at their designated location. For example the derangements of a set $\{1, 2, 3\}$ would be the sets $\{2, 3, 1\}$ and $\{3, 1, 2\}$. We can see that in both these sets neither 1, 2 or 3 appear as the first, second or third element in the sequence.
The number of derangements of a set of n items is denoted by $!n$ (or as $D(n)$) and is also called the sub-factorial of n

2.2 Closed form equation for derangements

To obtain the closed form equation for number of derangements for n-objects ($n \geq 0$), we use the inclusion-exclusion principle.

2.2.1 Derivation

Let Ω be the set of all possible arrangements for n objects.
Without loss in generality, we can assume that the correct arrangement for this set of objects is such that object$_i$ is position i, that is, object 1 is in the first position, object 2 in the second position and so on.
We aim to find the cardinality of set D, where D is the set of derangements.

$$\therefore D = \{arrangement \mid position\ of\ object_i \neq i, \forall i \leq n\}$$

Define Q_i as the set of all combinations such that object i goes in position i.

$$D = \Omega - \bigcup_{i=1}^{n} Q_i \tag{1}$$

Equation (1) implies that the set of derangements is equal to the set difference of the sample set (Ω) and the union of all sets where at least one object is in place.

$$\therefore |D| = |\Omega| - |\bigcup_{i=1}^{n} Q_i| \tag{2}$$

$$|\bigcup_{i=1}^{n} Q_i| = |Q_1 \cup Q_2 \cup \cdots \cup Q_n|$$

$$\begin{aligned} |Q_1 \cup Q_2 \cup \cdots \cup Q_n| = &\sum_{i} |Q_i| - \sum_{i<j} |Q_i \cap Q_j| \\ &+ \sum_{i<j<k} |Q_i \cap Q_j \cap Q_k| - \dots \\ &+ (-1)^n |Q_1 \cap Q_2 \cap \cdots \cap Q_n| \end{aligned} \tag{3}$$

Now, we know that,

$$
\begin{aligned}
\sum_{i} |Q_i| &= {}^nC_1(n-1)! \\
\sum_{i<j} |Q_i \cap Q_j| &= {}^nC_2(n-2)! \\
\sum_{i<j<k} |Q_i \cap Q_j \cap Q_k| &= {}^nC_3(n-3)! \\
&\vdots \\
|Q_1 \cap Q_2 \cap \cdots \cap Q_n| &= {}^nC_n(n-n)! = 1
\end{aligned}
$$

$$
\begin{aligned}
\therefore |D| &= |\Omega| - \sum_{i=1}^{n} (-1)^{i-1} \cdot {}^nC_i(n-i)! \\
&= n! - \sum_{i=1}^{n} (-1)^{i-1} \cdot {}^nC_i(n-i)! \qquad (4)
\end{aligned}
$$

Now,

$$
{}^nC_r = \frac{n!}{(n-r)! \cdot r!}
$$

$$
\begin{aligned}
\therefore |D| &= n! - \sum_{i=1}^{n} (-1)^{i-1} \cdot \frac{n!}{i!(n-i)!}(n-i)! \qquad (5) \\
&= n! - \sum_{i=1}^{n} (-1)^{i-1} \cdot \frac{n!}{i!} \\
&= n! + n! \left(-\frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!} \right)
\end{aligned}
$$

$$
\therefore |D| = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!} \right) \qquad (6)
$$

2.2.2 Code

```
def derangements(n):
    '''
    input: non-negative integer (n)
    output: number of derangements of n-objects (D(n))
    '''
```

```
    factorial_arr = []
    for i in range(n+1):
        factorial_arr.append(factorial(i))
    num_derangements = 0
    sign = 1
    for i in range(n+1):
        num = sign / factorial_arr[i]
        num_derangements += num
        sign *= -1
    num_derangements *= factorial_arr[-1]
    return int(num_derangements)
```

```
for i in range(21):
    print(f"D({i}) = {derangements(i)}")
```

```
D(0) = 1
D(1) = 0
D(2) = 1
D(3) = 2
D(4) = 9
D(5) = 44
D(6) = 265
D(7) = 1854
D(8) = 14833
D(9) = 133496
D(10) = 1334961
D(11) = 14684570
D(12) = 176214841
D(13) = 2290792932
D(14) = 32071101049
D(15) = 481066515734
D(16) = 7697064251745
D(17) = 130850092279664
D(18) = 2355301661033953
D(19) = 44750731559645120
D(20) = 895014631192902400
```

Figure 2: Derangement numbers using closed-form equation

2.3 Recurrence Relation

2.3.1 Derivatinon

Let $A = \{A_i \mid 1 \leq i \leq n\}$ be a set of n objects, and their indices represent their actual (correct) position in the set.

In order to create a derangement in set A, we can take a random object, A_k ($k \neq 1, 1 < k \leq n$) and place it in the first place. From this position, two possible cases arise:

Case 1: A_1 goes in place k

Case 2: A_1 goes in place l where $l \neq k, 1 < l \leq n$

Total number of ways for total derangement ($D(n)$) can be given by:
D(n) = (number of ways to chose A_k) AND (number of ways Case 1 can occur OR number of ways Case 2 can occur)

$$\begin{aligned} \text{Number of ways to choose } A_k &: {}^{n-1}C_1 = n-1 \\ \text{Number of ways Case 1 can happen } &: D(n-2) \\ \text{Number of ways Case 2 can happen } &: D(n-1) \end{aligned}$$

$$D(n) = (n-1)\left\{D(n-1) + D(n-2)\right\} \tag{7}$$

2.3.2 Code

```
def recursive_derangements(n):
    '''
    input: non-negative integer (n)
    output: number of derangements of n-objects (D(n))
    '''
    if (n == 0): return 1
    if (n == 1): return 0
    num_derangements = ((n-1) * (recursive_derangements(n-1) +
   recursive_derangements(n-2)))
    return num_derangements
```

```
for i in range(21):
    print(f"D({i}) = {recursive_derangements(i)}")
```

```
D(0) = 1
D(1) = 0
D(2) = 1
D(3) = 2
D(4) = 9
D(5) = 44
D(6) = 265
D(7) = 1854
D(8) = 14833
D(9) = 133496
D(10) = 1334961
D(11) = 14684570
D(12) = 176214841
D(13) = 2290792932
D(14) = 32071101049
D(15) = 481066515734
D(16) = 7697064251745
D(17) = 130850092279664
D(18) = 2355301661033953
D(19) = 44750731559645106
D(20) = 895014631192902121
```

Figure 3: Derangement numbers using the Recursion Relation

2.4 Probability of a derangement

2.4.1 Derivation

For a set of n-objects, the probability of it being in a state of derangement ($P\{D(n)\}$) is given by:

$$P\{D(n)\} = \frac{|D|}{n!} \tag{8}$$

However, substitution equation (6) in the above equation we obtain:

$$P\{D(n)\} = \frac{n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!}\right)}{n!}$$
$$\therefore P\{D(n)\} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!} \tag{9}$$

It can observed that equation (9) is the Taylor series expansion of e^{-1} and it converges towards this value for large n. Therefore, we get:

$$\lim_{n\to\infty} P\{D(n)\} = \frac{1}{e} \tag{10}$$

2.4.2 Code

```
def derangement_probability(n):
    '''
    input: non-negative integer (n)
    output: probability of derangement of n-objects (P(D(n)))
    '''
    return recursive_derangements(n)/factorial(n)
```

```
for i in range(21):
    print(f"Probability of derangements in {i} objects = {derangement_probability(i)}")
print("1/e = 0.36787944117144232 (upto 17 decimal places)")
```

```
Probability of derangements in 0 objects = 1.0
Probability of derangements in 1 objects = 0.0
Probability of derangements in 2 objects = 0.5
Probability of derangements in 3 objects = 0.3333333333333333
Probability of derangements in 4 objects = 0.375
Probability of derangements in 5 objects = 0.36666666666666664
Probability of derangements in 6 objects = 0.3680555555555556
Probability of derangements in 7 objects = 0.3678571428571429
Probability of derangements in 8 objects = 0.36788194444444444
Probability of derangements in 9 objects = 0.36787918871252206
Probability of derangements in 10 objects = 0.3678794642857143
Probability of derangements in 11 objects = 0.3678794392336059
Probability of derangements in 12 objects = 0.3678794413212816
Probability of derangements in 13 objects = 0.36787944116069116
Probability of derangements in 14 objects = 0.3678794411721619
Probability of derangements in 15 objects = 0.3678794411713972
Probability of derangements in 16 objects = 0.367879441171445
Probability of derangements in 17 objects = 0.36787944117144217
Probability of derangements in 18 objects = 0.36787944117144233
Probability of derangements in 19 objects = 0.36787944117144233
Probability of derangements in 20 objects = 0.36787944117144233
1/e = 0.36787944117144232 (upto 17 decimal places)
```

Figure 4: Convergence of $P\{D(n)\}$ for large n

3 Practical Applications

While derangements can be of great interest to mathematicians and computer scientists due to its fascinating properties, it also has a few applications in real life.

- Derangement sequences can be used to assign different jobs to a set of soldiers in a military corp so that they can be trained equally in all jobs.
- It can also be used by scientists to conduct experiments such that no two experiments have the same initial conditions

References

[1] All the code in this report have been programmed by me, and can be found in this GitHub Repository.

[2] https://local.disia.unifi.it/merlini/papers/Derangements.pdf