# Lab_05

October 7, 2019

In [1]:
```
from datascience import *
from prob140 import *
import numpy as np
from scipy import special
import pickle
```

In [2]:
```
# Run this code

import itertools as it
from collections import Counter

def clean_string(string):
    """
    Cleans an input string by replacing all newlines with spaces, and then
    removing all letters not in *allowable_letters*
    """
    string = string.replace("\n"," ")
    return "".join([i for i in string.lower().strip() if i in allowable_letters])

def load_bigrams(text):
    """
    Takes a string which has already been cleaned, and returns a dictionary
    of conditional bigram probabilities

    cond_bigram[(a,b)] = P(X_{n+1} = b | X_{n} = a)

    Uses Laplace smoothing with size $1$, to remove zero transition probabilities
    """
    bigram_counter = Counter(list(it.product(allowable_letters, repeat=2)))
    gram_counter = Counter(allowable_letters*len(allowable_letters))
    for l1, l2 in zip(text,text[1:]):
        bigram_counter[(l1, l2)] += 1
        gram_counter[l1] += 1
    cond_bigram = {k:v/gram_counter[k[0]] for k,v in bigram_counter.items()}
    return cond_bigram

def bigram_from_file(filename):
    """
```

```
    Given a filename, this reads it, cleans it, and returns the conditional bigram
    """
    file_text = open(filename).read()
    file_text = clean_string(file_text)
    return load_bigrams(file_text)

def reverse_cipher(cipher):
    return {v:k for k, v in cipher.items()}

def print_differences(cipher1, cipher2):
    for k in cipher1:
        if cipher1[k] != cipher2[k]:
            print("%s: %s  %s"%(k,cipher1[k], cipher2[k]))

def num_errors(cipher, encoded_text, original_text):
    decoded = np.array(list(decode_text(encoded_text, cipher)))
    original = np.array(list(original_text))
    num_errors = np.count_nonzero(decoded != original)
    return num_errors

def get_secret_text(student_id):
    with open('Lab05_data/secret_strings.p', 'rb') as f:
        texts = pickle.load(f)
    return texts[student_id % len(texts)]
```

# 1 Lab 5: Code Breaking by MCMC

Cryptography is the study of algorithms used to encode and decode messages. Markov Chain Monte Carlo (MCMC) methods have been successfully used to decode messages encrypted using substitution codes and also more complex encryption methods. In this lab you will apply MCMC and the Metropolis algorithm to decode English text that has been encrypted by a substitution code.

The lab is based on the paper The Markov Chain Monte Carlo Revolution by Persi Diaconis. It was presented at the 25th anniversary celebrations of MSRI up the hill, and appeared in the Bulletin of the American Mathematical Society in November 2008. The code is based on Simulation and Solving Substitution Codes written by Stephen Connor in 2003.

Recall that in class we worked with a tiny alphabet and a short message that had been encoded using a substitution code. We used a Markov chain model and a scoring system to choose the decoder that had the highest score. That is, we chose the decoder that had the highest likelihood given the message.

Because the alphabet was small, we were able to list all the decoders and their corresponding scores. When the alphabet is large, making an exhaustive list of all possible decoders and scores is not feasible. That's where the Monte Carlo part of the algorithm comes in. The idea is to choose decoders at random using an algorithm that favors good decoders over bad ones.

In this lab you will work with an alphabet consisting of all 26 English letters as well as a space character that will have a special status. Before you begin, please review Sections 11.3 (Code Breaking) and 11.4 (Markov Chain Monte Carlo) of the textbook. The lab follows those sections

closely.

Unlike all the other labs, this lab is not divided into parts. It's just one small project. You will learn how to: - Apply a decoder to encoded text - Use transformations for improved numerical accuracy - Implement the Metropolis algorithm - Decode a message encrypted by a substitution code

The programming needed for this lab is more complex than what can reasonably be expected based on just Data 8 background, so much of it has been done for you. Those of you who have more extensive Python knowledge might be interested in looking at the details.

Please start by running all the cells above this one.

### 1.1 Instructions

Your labs have two components: a written portion and a portion that also involves code. Written work should be completed on paper, and coding questions should be done in the notebook. You are welcome to LaTeX your answers to the written portions, but staff will not be able to assist you with LaTeX related issues. It is your responsibility to ensure that both components of the lab are submitted completely and properly to Gradescope. Refer to the bottom of the notebook for submission instructions.

### 1.2 Alphabet and Bigrams

The text that you are going to decode was written in English. For data on the frequencies of all the different bigrams in English, we will start by counting all the bigrams in *War and Peace* by Tolstoy. That is one of the longest novels in English (actually in Russian, but we are using an English translation from Project Gutenberg) and is often used as a "corpus" or body of language on which to base analyses of other text.

To keep the calculations manageable, we will restrict ourselves to a 27-character alphabet, consisting of the 26 lower case letters and a space. The cell below places all 27 in a list.

```
In [3]: allowable_letters = list("abcdefghijklmnopqrstuvwxyz ")
```

In the cell below, we find the relative frequencies of all bigrams in War and Peace and place them in the dictionary `wp_bigrams`. You don't need to know what a dictionary is. It's fine to imagine it containing the same information as a Table that has a column with the bigrams and another column with the relative frequencies.

```
In [4]: wp_bigrams = bigram_from_file("Lab05_data/warandpeace.txt")
```

#### 1.2.1 1. Transition Matrix

As in class, the model is that the sequence of characters in the text is a Markov chain. The state space of the chain is the alphabet of 27 characters. Of course the text has other punctuation, but we are stripping all of it. We are also replacing upper case letters by lower case.

Construct the transition matrix for this Markov chain based on the relative frequencies in `wp_bigrams` defined above.

We have started the code off for you by defining the transition function `wp_transition`. Use `allowable_letters` in your definition.

# sp18_midterm

February 23, 2019

Probability for Data Science
UC Berkeley, Spring 2019
Ani Adhikari and Jim Pitman


## 1 Spring 2018 Midterm

This set consists of the problems from the Spring 2018 midterm. It only involves written work; there are no coding parts.

**Recommendation:** Use this practice as an opportunity to assess your preparation for the upcoming midterm. Don't start it before you have completed the bulk of your review for the exam. Do your review, set aside all the course materials, find a place where you can work on your own, and give yourself 75 minutes to do the assignment. The problems that you are not able to do in that time will give you an indication of where you need to focus the remaining portion of your review.

#### 1.0.1 1.

In a large population of voters, 48% favor Proposition 140. A random sample of 200 voters is drawn. You can assume that the method of sampling is equivalent to drawing at random with replacement.

**a)** Find the chance that the majority (that is, more than half) of the sampled voters favor Prop 140.

**b)** Given that exactly half of the sampled voters favored Prop 140, what is the chance that the first 25 sampled voters all favored Prop 140?

#### 1.0.2 2.

A fair die that has 1 blue face, 2 red faces, and 3 green faces is rolled repeatedly. Find the expected number of rolls till two different colors have appeared.

#### 1.0.3 3.

Suppose a sample consists of random variables $X_1, X_2, \ldots, X_n$ that are independent and identically distributed. The distribution of $X_1$ is shown below. Assume $p \in (0, 1/6)$.

| $k$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $P(X_1 = x)$ | $p$ | $2p$ | $1-6p$ | $2p$ | p |

- Let $M = \max\{X_1, X_2, \ldots, X_n\}$ be the sample maximum.
- Let $N = \min\{X_1, X_2, \ldots, X_n\}$ be the sample minimum.
- Let $R = M - N$ be the range of the sample.

**a)** Find a function $f$ such that $M$ is equal in distribution to $f(N)$. Explain your answer briefly.
**b)** Find $E(R)$.

#### 1.0.4 4.

Machine A produces 1000 items, each of which is defective with chance $1/1000$ independently of all others. Independently of Machine A, Machine B produces 3000 items, each of which is defective with chance $2/1000$ independently of all others.

**a)** Pick **one** of the options (i) and (ii). Fill in the first blank of your chosen option with the name of a famous distribution and the second with its parameter or parameters. **Justify your choices.**

Let $T$ be the total number of defective items produced by Machines A and B. The distribution of $T$ is

(i) exactly _____ with parameter (or parameters) _____.

(ii) approximately _____ with parameter (or parameters) _____.

**b)** Each defective item leads to exactly one of three outcomes: no complaint, a minor complaint, or a major complaint. Suppose that for each defective item, the chance of no complaint is 0.5, the chance of a minor complaint is 0.4, and the chance of a major complaint is 0.1, independently of all other variables. Find or approximate the chance that among all the items produced there are 3 defective items with no complaint, 2 defective items with a minor complaint, and 1 defective item with a major complaint.

#### 1.0.5 5.

Fix a positive integer $k$ and two numbers $r$ and $s$ in the interval $(0, 1)$. Let $V$ have the binomial $(k, r)$ distribution. Let $T$ be independent of $V$ and have the distribution given by $P(T = j) = (1-s)^j s$ for $j \geq 0$.

**a)** Find $P(V = T)$.

**b)** For $k = 5$, $r = 2/3$, and $s = 1/2$, the answer to Part **a** is equal to $n/m$ where $n$ and $m$ are two positive integers. Fill in the blanks with their numerical values and justify your answers. No, you don't need a calculator.

$n$ = _____ $m$ = _____

#### 1.0.6 6.

For $N$ a fixed positive integer, $N$ blue particles and $N$ red particles are distributed in two containers so that there are $N$ particles in each container. A process evolves as follows.

At every step, one particle is selected uniformly at random from each container, independently of the other choice; then these two particles are switched. That is, each chosen particle is taken out of its container and placed in the other one.

For $n \geq 0$, let $X_n$ be the number of blue particles in Container 1.

**a)** What is the state space of the chain $\{X_0, X_1, X_2, \ldots\}$? Find the one-step transition probabilities.

**b)** Find the stationary distribution of the chain. Recognize this as one of the famous ones and provide its name and parameters. It might help to remember that $\binom{n}{k} = \binom{n}{n-k}$.

# Lab_04

October 1, 2019

Probability for Data Science
UC Berkeley, Spring 2019
Ani Adhikari and Jim Pitman


```
In [1]: from prob140 import *
        from datascience import *
        import numpy as np
        from scipy import stats

        import matplotlib.pyplot as plt
        %matplotlib inline
        import matplotlib
        matplotlib.style.use('fivethirtyeight')
```

```
In [2]: def split_unit_interval(cdf_table):
            plt.figure(figsize=(1,4))
            values = cdf_table.column(cdf_table.num_columns - 1)
            cum = list(np.cumsum(values))
            cur_axes = plt.gca()
            cur_axes.axes.get_xaxis().set_visible(False)
            plt.yticks([0] + cum)
            plt.ylim(-0.1, 1.1)
            plt.plot([0,0], [0,1], color="darkblue", lw=3)
            plt.xlim(-0.12, 1)
            plt.scatter([0]*(len(cum) + 1),
                        [0] + cum, s=55, color="k")
```

## 1 Lab 4: Success Runs and Other Patterns

Consider a sequence of i.i.d. Bernoulli $(p)$ trials. We will use the standard image of tossing a coin that lands heads with chance $p$. So we will denote success by $H$ and failure by $T$.

For $n > 1$, the first *run of n successes* is the earliest trial at which $n$ consecutive successes have been observed. For example, if the sequence of tosses starts out with $THHHTHHHH$, then the first run of two successes happens at Trial 3, the first run of three successes happens at Trial 4, and the first run of four successes happens at Trial 9.

Success runs carry information about $p$. For example, if you see long success runs early in the sequence of trials, you might want to infer that $p$ is large.

In this lab, you will study the random number of trials needed to get a success run of a specified length. You will then extend your calculations to the number of trials till you get patterns of successes and failures (for example, $HT$ instead of $HH$), and then to patterns of letters chosen randomly from the alphabet.

At the end of the lab you will know the expected time a monkey will take to type $ABRACADABRA$ by hitting letters uniformly at random on a keyboard, and, by extension, the expected time it will take to type out the complete works of Shakespeare.

Fortunately, monkeys know better than to engage in such things. But in the human world, success runs have uses in information theory, reliability and quality control, genetics, and other areas. Also, expected waiting times till patterns appear have beautiful patterns of their own. So they are well worth studying.

What you will learn in this lab:

- A simple formula for expected waiting time till a run of $n$ successes
- How to simulate the distribution of the waiting time till a run of $n$ successes
- Approximations to the distribution of the waiting time
- Extensions of the expectation formula to patterns of successes and failures
- How to know the expected waiting time till any pattern, just by looking at the pattern

### 1.1 Instructions

Your labs have two components: a written portion and a portion that also involves code. Written work should be completed on paper, and coding questions should be done in the notebook. You are welcome to LaTeX your answers to the written portions, but staff will not be able to assist you with LaTeX related issues. It is your responsibility to ensure that both components of the lab are submitted completely and properly to Gradescope. Refer to the bottom of the notebook for submission instructions.

### 1.2 Preliminary: Simulating Discrete Random Variables

Data scientists often simulate data under specified models of randomness. In this and future labs, you will use Python simulate random variables that have a given distribution.

The general call is `stats.distribution_name.rvs(parameters, size=n)`. This evaluates to an array of observed values of $n$ i.i.d. random variables of the specified distribution. The acronym `rvs` stands for "random variates".

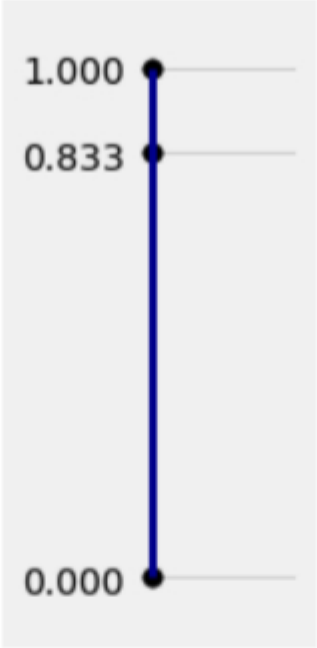
To see how this works, start by using Python to simulate a single Bernoulli (1/6) random variable. In other words, ask Python to roll a die and tell you whether or not it landed six. Run the cell a few times.

```
In [3]: stats.bernoulli.rvs(1/6, size=1)
```

```
Out[3]: array([0])
```

If you are curious about how Python generates the 0 or 1, here is a visualization of what it does. Later in the course, you will study this method in far greater generality.

```
In [4]: six_on_one_roll = Table().values([0, 1]).probabilities([5/6, 1/6])
        split_unit_interval(six_on_one_roll)
```



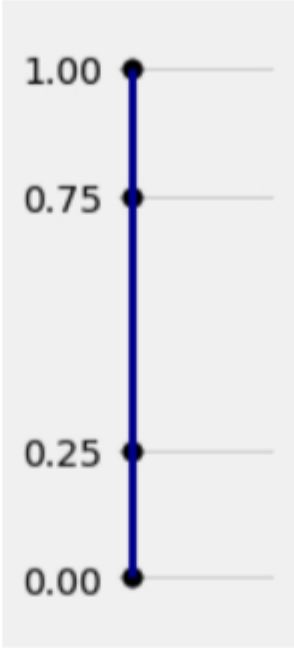

The unit interval $[0, 1]$ has been split into two parts, one of length $5/6$ (approximately 0.833) and the other of the remaining length $1/6$. This is a version of the "stacked bar chart" visualizations that you saw in Lab 3.

The Python function for generating one Bernoulli $(1/6)$ random variable essentially throws a dart uniformly on the unit interval. If the dart lands in the interval $(0, 5/6)$, it returns 0. If the dart lands in $(5/6, 1)$, it returns 1. (If you are wondering what happens if the dart lands exactly at $5/6$, or 0, or 1, for now please just accept that the chance of that is 0. We will discuss this issue later in the course.)

Since the dart is thrown uniformly, the chance that it lands in the lower interval is $5/6$ of the total length 1, which is $5/6$. So the chance that the function returns 0 is $5/6$, which is exactly what it should be. It follows that the chance that the function returns 1 is $1/6$.

There is nothing special about the Bernoulli distribution as far as this process is concerned. Every discrete distribution can be simulated in an analogous way. The graph below shows how Python simulates the number of heads in two tosses of a coin.

```
In [5]: heads_in_two_tosses = Table().values([0, 1, 2]).probabilities([1/4, 1/2, 1/4])
        split_unit_interval(heads_in_two_tosses)
```




The process:

- Throw a dart uniformly on the unit interval.
  - If the dart lands in the lowest interval, return 0.
  - If the dart lands in the middle interval, return 1.
  - If the dart lands in the highest interval, return 2.

In a future lab, you will study the process further and develop a general method of simulating random variables with specified distributions.

Now that you understand roughly what Python is doing, focus on the code. Read each cell below carefully before you run it. Also, run each cell several times to see how the output changes.

```
In [7]: # One Bernoulli (1/6) variable;
        # the output of rvs is an array

        stats.bernoulli.rvs(1/6, size = 1)
```

```
Out[7]: array([1])
```

Compare this with:

```
In [8]: # One Bernoulli (1/6) variable
        # accessed as a value that you can work with directly

        stats.bernoulli.rvs(1/6, size = 1).item(0)
```

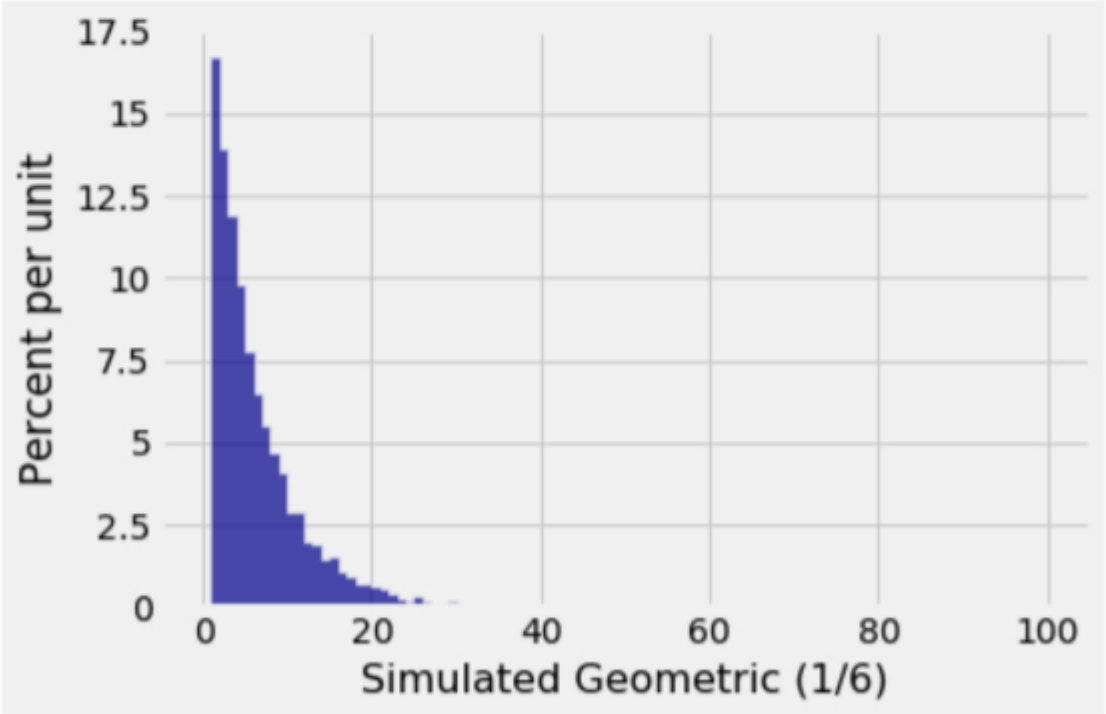
```
Out[8]: 1
```

Simulating sixes in 10 rolls of a die:

```
In [9]: # 10 i.i.d. Bernoulli (1/6) variables

        stats.bernoulli.rvs(1/6, size = 10)

Out[9]: array([0, 1, 0, 1, 0, 0, 0, 0, 0, 0])
```

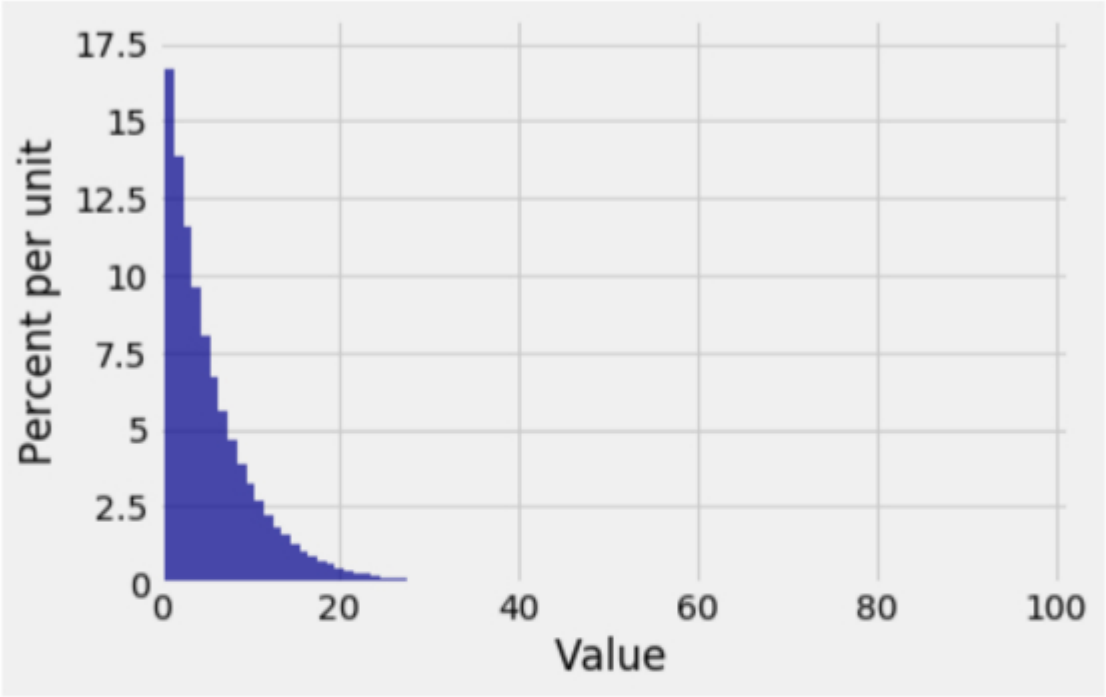
SciPy defines the geometric $(p)$ random variable in the same way as in our class: the distribution of the number of trials till the first success in i.i.d. Bernoulli $(p)$ trials. Thus the possible values of a geometric random variable are $1, 2, 3, \ldots$.

```
In [10]: # Waiting time till the first six in rolls of a die

         stats.geom.rvs(1/6, size = 1)

Out[10]: array([8])

In [11]: # Empirical histogram of 10,000 i.i.d. repetitions of
         # counting the number of rolls till the first six

         wait_till_six = stats.geom.rvs(1/6, size = 10000)
         Table().with_column('Simulated Geometric (1/6)', wait_till_six).hist(
             bins = np.arange(101))
```




This is an empirical approximation of the geometric $(1/6)$ probability histogram below.

```
In [12]: # Probability histogram of geometric (1/6) random variable

         k = np.arange(1, 101)
         probs = stats.geom.pmf(k, 1/6)
         geom_dist = Table().values(k).probabilities(probs)
         Plot(geom_dist)
```




As you know, if a random variable $X$ has the geometric $(p)$ distribution, the probability mass function of $X$ is given by

$$P(X = k) = q^{k-1}p, \ k \geq 1$$

Here $q = 1 - p$ is standard notation that we will be using throughout the lab.

For each $k \geq 2$, $P(X = k) = qP(X = k - 1)$. You can see that the size of each bar of the probability histogram is the same fraction $5/6$ of the previous bar.

### 1.3 Part 1: Expected Time Till a Success Run

Section 9.3 of the textbook is **required reading** before you go further. The probabilistic techniques you need for this lab are versions of those in the section. Skipping the reading will not save time because you will most likely just take longer to answer the questions below.

In a sequence of i.i.d. Bernoulli $(p)$ trials, let $W_{pattern}$ be the number of trials till *pattern* appears. We call this the *waiting time* till *pattern* appears.

Thus $W_H$ is the waiting time till $H$ appears, $W_{HH}$ is the waiting time till a success run of length 2, and so on. This is the same notation as in Section 9.3.

### 1.3.1 1a) [ON PAPER] Expectations of $W_H$ and $W_{HH}$

Use the results of Section 9.3 to fill in the blanks below with math expressions involving **only** positive terms of the form $1/p^k$ for $k \geq 1$. Show any algebra that you use.

(1) $E(W_H) =$ _____

(2) $E(W_{HH}) =$ _____

(1) $\frac{1}{p}$

(2) $\frac{1+p}{p^2} = \frac{1}{p^2} + \frac{1}{p}$

### 1.3.2 1b) [ON PAPER] Expectation of $W_{HHHH}$

Find $E(W_{HHHH})$ by conditioning on the trial after $W_{HHH}$, as follows.

- Notice that to get to $W_{HHHH}$ you first have to get to $W_{HHH}$, whose expectation you already know.
- Let $x = E(W_{HHH})$. This is what you are trying to find.
- Think about what happens on the trial that follows $W_{HH}$, and develop an equation for $x$. Refer to Answer 2 of the example *Waiting for HH* in Section 9.3.
- Solve the equation.

As in **1a**, your answer should involve only positive terms of the form $1/p^k$ for $k \geq 1$. Show any algebra that you use.

$x = E(W_{HH}) + 1p + (1+x)q$
So $px = E(W_{HH}) + 1 = \frac{1}{p^2} + \frac{1}{p} + 1$
So $x = \frac{1}{p^3} + \frac{1}{p^2} + \frac{1}{p}$

### 1.3.3 1c) [ON PAPER] Expected Waiting Time Till a Success Run of Length $n$

By an abuse of notation, let $W_{H,n}$ be the waiting time till $n$ consecutive successes appear. Thus for example $W_{H,7}$ is notation for $W_{HHHHHHH}$. It's more compact and it spares you having to count the number of successes in the run.

(1) Use your answers to **1a** and **1b** to guess a formula for $E(W_{H,n})$ for $n \geq 1$.

(2) Use induction to prove your guess. Adapt the method used in **1b**.

(1) Guess $E(W_{H,n}) = \sum_{k=1}^{n} \frac{1}{p^k} = \frac{1}{p^n} + \frac{1}{p^{n-1}} + \cdots + \frac{1}{p}$. By **1a** and **1b**, the formula is correct for $n = 1, 2, 3$.

(2) Assume that the formula is correct for $n$. Let $x = E(W_{H,n+1})$. By conditioning on the trial after $W_{H,n}$:

$x = E(W_{H,n}) + 1p + (1+x)q$, and so $px = E(W_{H,n}) + 1$.
So $x = \frac{1}{p}\sum_{k=1}^{n} \frac{1}{p^k} + \frac{1}{p} = \sum_{k=2}^{n+1} \frac{1}{p^k} + \frac{1}{p} = \sum_{k=1}^{n+1} \frac{1}{p^k}$ which is the guess for $n+1$.

### 1.3.4 1d) Numerical Values

Define a function `ev_W_run` that takes $p$ and $n$ as arguments and returns $E(W_{H,n})$. Just replace the ellipsis by an expression. Do not add any other lines of code.

```
In [24]: def ev_W_run(p, n):
             return sum(1 / p ** np.arange(1, n+1))
```

What should `ev_W_run(1/6, 1)` evaluate to? Run the cell below to check that your function works in this case.

```
In [25]: ev_W_run(1/6, 1)
```

```
Out[25]: 6.0
```

What should `ev_W_run(0.5, 3)` evaluate to? Run the cell below to confirm.

```
In [26]: ev_W_run(0.5, 3)
```

```
Out[26]: 14.0
```

Now use `ev_W_run` to find the expectation of each of the following random variables.

(1) number of tosses of a fair coin till 10 consecutive heads appear

(2) number of rolls of a die till six consecutive sixes appear

(3) number of times a random number generator is run till 000 appears, if the generator draws at random with replacement from the 10 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9

(4) number of days till a robot types ZZZZZ if the robot types at the rate of 10 letters per second (without breaks) and chooses letters at random with replacement from the 26 upper case letters of the English alphabet

```
In [27]: # Expected number of:

         # (1) fair coin tosses till 10 consecutive heads
         ans_1 = ev_W_run(1/2, 10)

         # (2) rolls of a die till 6 consecutive sixes
         ans_2 = ev_W_run(1/6, 6)

         # (3) runs of a random number generator till 000
         ans_3 = ev_W_run(1/10, 3)

         # (4) days till ZZZZZ by robot typist
         ans_4 = ev_W_run(1/26, 5) / (10 * 60 * 60 * 24)

         ans_1, ans_2, ans_3, ans_4
```

```
Out[27]: (2046.0, 55986.000000000015, 1109.9999999999998, 14.301655092592588)
```

#newpage

### 1.4 Part 2: Distribution of the Time Till a Success Run

For general positive integer $n$, the distribution of $W_{H,n}$ has been known for a long time. The derivation requires some math, and the formula has connections with the Fibonacci numbers, as you can see here for example.

Though the math isn't easy, you can simulate $W_{H,n}$ quite easily. That is what you will do in this part of the lab. The simulations will give you insight about the distribution using almost no math at all.

To simulate a value of $W_{H,n}$, you clearly need an iteration:

- Toss once and see if the pattern appears;
- if it doesn't, toss again and repeat ...
- until the pattern appears.

The number of iterations is therefore random. The process stops when a condition is satisfied ("the pattern appears"), not at some previously specified number of iterations as in a `for` loop.

In Python, `while` loops are used in such cases. If you have seen `while` in another class, you can go directly to **2a** below. If not, go through the preliminary example here.

As explained in Professor DeNero's Composing Programs:

"A while clause contains a header expression followed by a suite:

while < expression >:

```
< suite >
```

To execute a while clause:

- Evaluate the header's expression.
- If it is a true value, execute the suite, then return to step 1."

In what follows, `expression` will be a Boolean. If it is `True`, the "suite" of code — a sequence of lines of code — will be executed. Next, `expression` will be evaluated again, and the process will continue till at some point `expression` evaluates to `False`. Then the process will end.

It is *really bad news* if `expression` is never false, because then the process will continue forever. In our examples, `expression` will be `False` if a coin shows a specified face or if a specified pattern appears. These events are certain to happen at some point, so our loops will end.

Suppose we have a coin that lands heads with chance $p = 0.2$. In the example below, we will get Python to toss the coin until it lands heads, and display the results of all the tosses. The details:

- Starting point: an empty collection array called `failures`
- The `while` header expression: Toss the 0.2 coin and see if it is a 0
- If the coin shows 0:
  - The `suite`: append 0 to `failures`
  - Return to the `while` header expression
- If the coin shows 1, stop

When the process stops, the array `failures` will contain one 0 for every toss before the final one, but it will *not* contain 1 for the final toss that stopped the process. So we will append 1 to `failures` and display the result. In this way we will have simulated $W_H$ for $p = 0.2$.

```
In [28]: failures = make_array()

         while stats.bernoulli.rvs(0.2, size=1).item(0) == 0:  # toss; while you get 0
             failures = np.append(failures, 0)                 # append 0 to failures

         # This while loop stops when a toss lands 1.
         # Show all the failures and the success that stopped the process.

         np.append(failures, 1)

Out[28]: array([0., 0., 0., 1.])
```

### 1.4.1 2a) Simulating One Value of $W_{H,n}$

Complete the cell below to define a function `run` that takes $p$ and $n$ as arguments and returns one simulated value of $W_{H,n}$.

As you toss, you will be keeping track of how many times you have tossed and also the number of consecutive heads that include the toss just completed.

For example, suppose you are waiting for the pattern $HHH$.

- If the first three tosses are $HTH$, then you will successively note:
  - tosses = 1, consecutive heads = 1
  - tosses = 2, consecutive heads = 0
  - tosses = 3, consecutive heads = 1
  - and you will continue to toss.
- If the first six tosses are $THTHHH$, then you will successively note:
  - tosses = 1, consecutive heads = 0
  - tosses = 2, consecutive heads = 1
  - tosses = 3, consecutive heads = 0
  - tosses = 4, consecutive heads = 1
  - tosses = 5, consecutive heads = 2
  - tosses = 6, consecutive heads = 3
  - and you will stop tossing.

```
In [29]: def run(p, n):
             """Returns one simulated value of W_H,n
             in i.i.d. Bernoulli (p) trials"""

             tosses = 0                    # Number of tosses
             in_a_row = 0                  # Number of consecutive heads observed

             while in_a_row < n:           # While fewer than n consecutive heads
                 tosses = tosses + 1       # update tosses
                 if stats.bernoulli.rvs(p, size=1).item(0) == 1:
                     in_a_row = in_a_row + 1       # update in_a_row
                 else:
                     in_a_row = 0                  # reset in_a_row
```

```
        return tosses
```

For a crude check of your function, think about whether `run(0.9, 1)` should evaluate to a large number of tosses or a small number, and then see what your function returns. Run the cell a few times.

```
In [30]: run(0.9, 1)
```

```
Out[30]: 1
```

Now run the next cell to see if the function behaves differently.

```
In [31]: run(0.1, 1)
```

```
Out[31]: 8
```

### 1.4.2 2b) Repeated Simulation

For a more thorough check, it will help to use the function several times and then check the properties of the collection of simulated values.

So use the function `run` to define a new function `simulate_run` that takes as its arguments $p$, $n$, and a positive integer `repetitions`, and returns an array consisting of length equal to `repetitions`, each of whose entries is a simulated value of $W_{H,n}$. Use as many lines of code as you need.

```
In [32]: def simulate_run(p, n, repetitions):
             """Returns an array of length equal to repetitions,
             whose entries are independent simulated values of W_H,n
             in i.i.d. Bernoulli (p) trials"""
             results = make_array()
             for i in np.arange(repetitions):
                 results = np.append(results, run(p, n))
             return results
```

What should the value of `np.mean(simulate_run(0.5, 3, 10000))` be, roughly? Look at **1d** for a reminder. Then run the cell below to confirm. The simulation might take a while, and the approximation might be a bit off. That's OK.

```
In [33]: np.mean(simulate_run(0.5, 3, 10000))
```

```
Out[33]: 13.9744
```

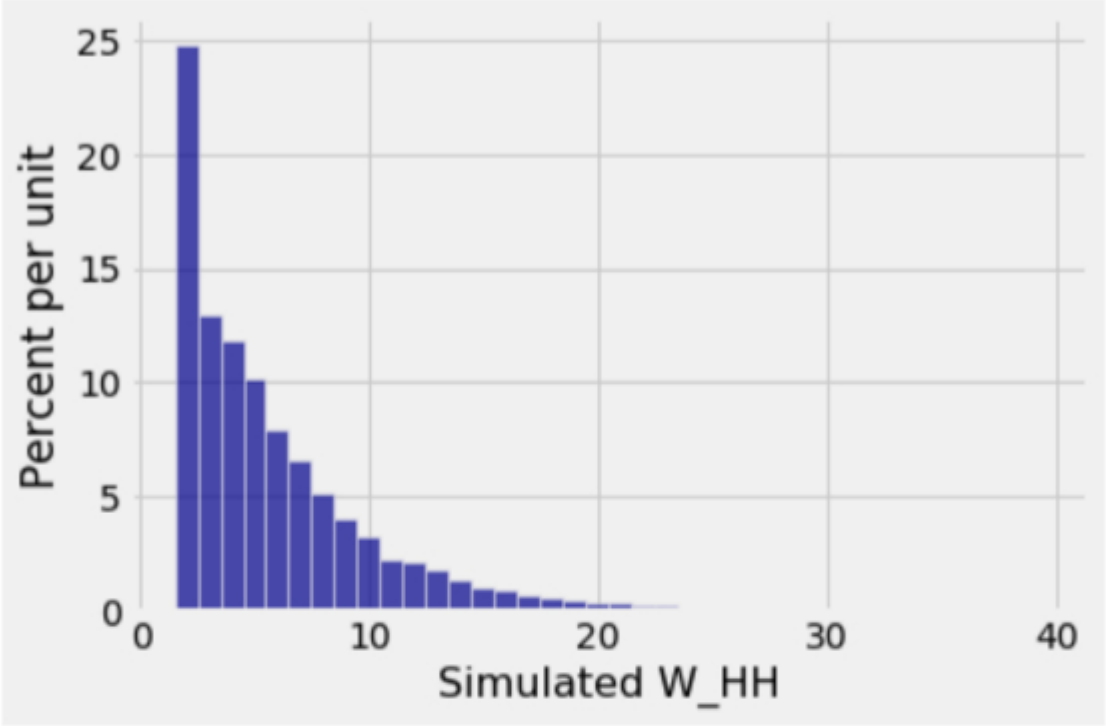
### 1.4.3 2c) Empirical Distribution of $W_{HH}$ When $p = 0.5$

Create an array `sim_W_HH` consisting of 10,000 independent simulated values of $W_{HH}$ for a fair coin.

```
In [34]: sim_W_HH = simulate_run(0.5, 2, 10000)
```

Run the cell below to display an empirical histogram of $W_{HH}$ for a fair coin.

```
In [35]: Table().with_column('Simulated W_HH', sim_W_HH).hist(bins = np.arange(1.5, 40.5))
```




That's an unusual shape! As a check, run the cell below and compare the results:

```
In [36]: ev_W_run(0.5, 2), np.mean(sim_W_HH)

Out[36]: (6.0, 5.967)
```

Calculate the probabilities corresponding to the first three bars, as listed below. Remember that the histogram is an empirical approximation to the exact probability histogram, so what you calculate might not be exactly what you see. But is should be close. Also remember that the coin is fair.

Find the exact values of the following and see if they are consistent with the approximations in the histogram.

(1) $P(W_{HH} = 2)$

(2) $P(W_{HH} = 3)$

(3) $P(W_{HH} = 4)$

(1) Event = HH on first two tosses. So chance = $1/4$; consistent.

(2) Event = THH on first three tosses. So chance = $1/8$; consistent.

(3) Event = TTHH or HTHH on the first four tosses. So chance = $2(1/16) = 1/8$, consistent.

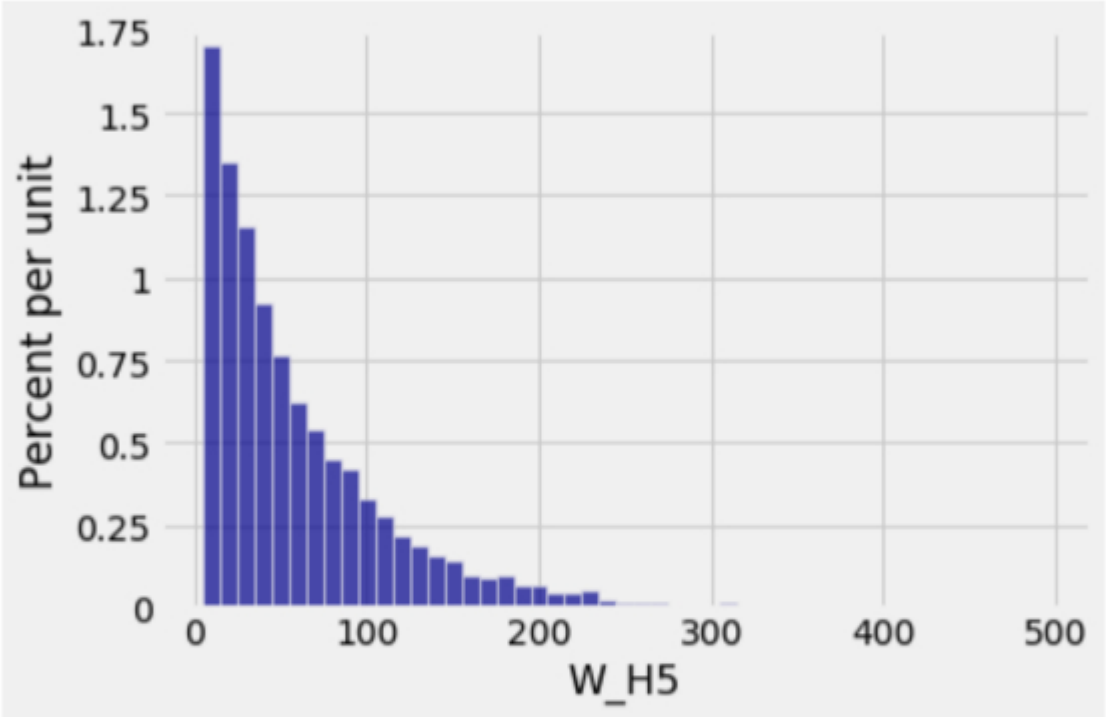
### 1.4.4 2d) Empirical Distribution of $W_{H,5}$ When $p = 0.5$

Create an array `sim_W_H5` consisting of 10,000 independent simulated values of $W_{H,5}$. The simulation takes a while.

```
In [37]: sim_W_H5 = simulate_run(0.5, 5, 10000)
```

Draw the empirical histogram of the values in `sim_W_H5`. Use `hist` with the option `bins = np.arange(5, 501, 10)`.

```
In [38]: Table().with_column('W_H5', sim_W_H5).hist(bins = np.arange(5, 501, 10))
         plt.ylim(0, 0.0175); # ignore; forces the vertical scale to go up to 1.75 %/unit
```




To check that your simulated values make sense, run the cell below and compare the two resulting values.

```
In [39]: ev_W_run(0.5, 5), np.mean(sim_W_H5)
```

```
Out[39]: (62.0, 61.3676)
```

### 1.4.5 2e) Approximate Distribution of $W_{H,5}$

The histogram in **2d** has a recognizable shape. Fill in the blanks below to figure out a straightforward approximation to the probability distribution of $W_{H,5}$.

(1) Fill in the blank with a number: $E(W_{H,5}) =$ _____.

(2) Fill in the blank with a range of values. If the range is infinite, it's fine to use "..." or any reasonable description.

The possible values of $W_{H,5}$ are _____.

(3) Fill in the blank with one of the names binomial, Poisson, and geometric:

The shape of the empirical distribution of $W_{H,5}$ resembles the shape of a _____ distribution, but the possible values are different.

(4) Fill in the first blank with a number, the second with the name you chose in (3), and the third with the appropriate numerical parameter or parameters. Carefully go through your answers to (1)-(3) before you fill in these blanks.

The distribution of $W_{H,5}$ is pretty close to the distribution of $X +$ _____ where $X$ has the _____ distribution with parameter (or parameters) _____.

(1) 62

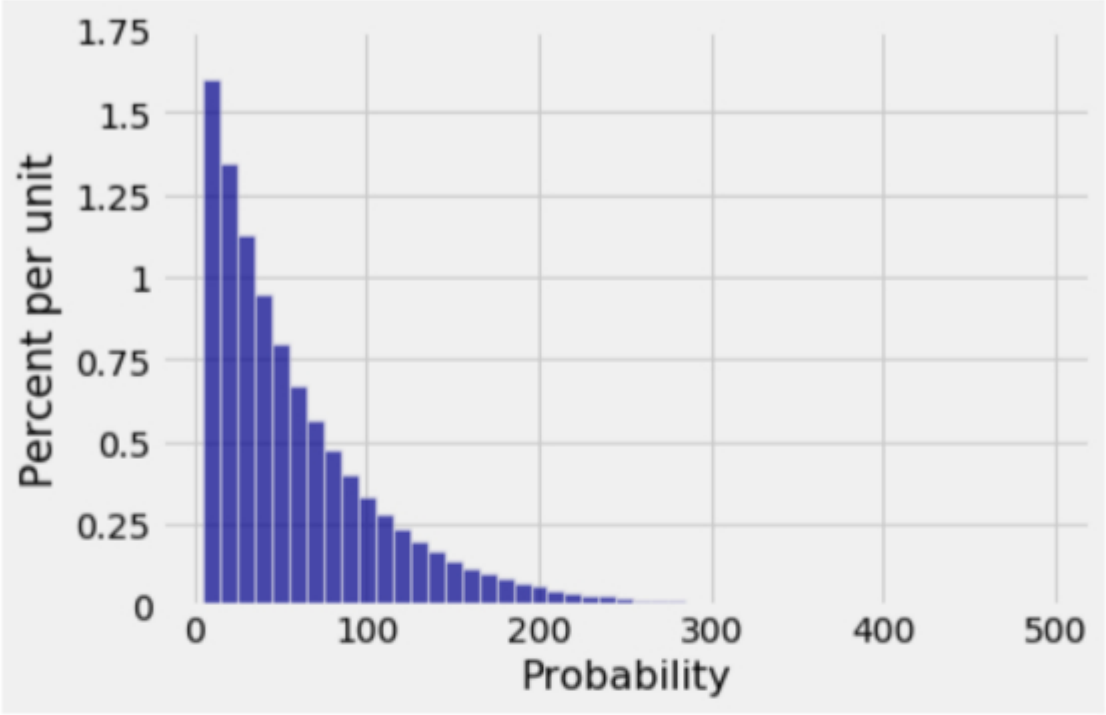
(2) 5, 6, 7, ...

(3) geometric

(4) 4, geometric, 1/58

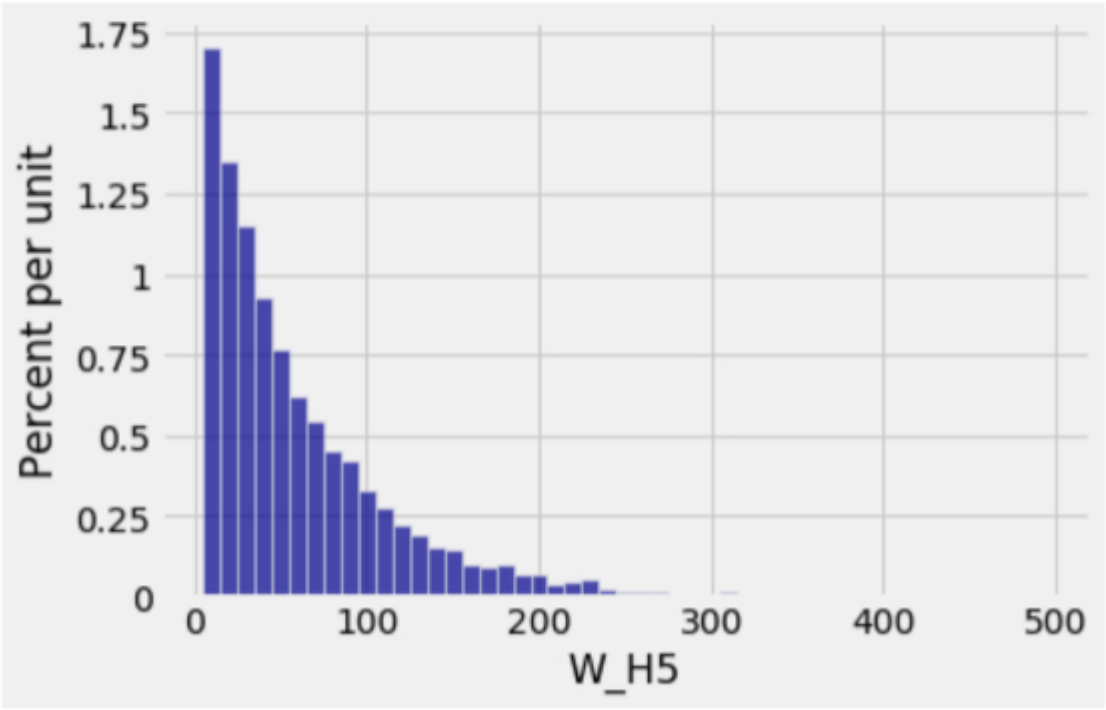
Now complete the cell to plot the probability histogram of the approximating distribution in (4). We have started you off with an array of possible values but you might have to adjust the possible values later in the cell.

```
In [40]: k = np.arange(1, 1001)
         approx_probs = stats.geom.pmf(k, 1/58)
         approx_dist = Table().values(k+4).probabilities(approx_probs)

         # Ignore; Forces hist to use the same scale as the empirical histogram
         approx_dist.hist(bin_column='Value', bins=np.arange(5, 501, 10))
         plt.ylim(0, 0.0175);
```




Here is your empirical distribution of $W_{H,5}$ again, for ease of reference.

```
In [41]: Table().with_column('W_H5', sim_W_H5).hist(bins = np.arange(5, 501, 10))
```



Your approximation should be pretty good though of course it will be a bit off. What you have discovered is also true for larger values of $n$, that is, for waiting times till longer success runs.

As above, let $p = 0.5$. In the cell below, find an approximate value of $P(W_{H,5} > 100)$ as an arithmetic expression using just elementary operations. You should be able to calculate it on a hand calculator if you didn't have this notebook. **Do not** use the distributions calculated above or `stats`, `special` or other modules.

```
In [13]: (57/58) ** 96
```

```
Out[13]: 0.18832100471449834
```

Now check if your answer is consistent with your simulation, by writing an expression that evaluates to the proportion of entries in `sim_W_H5` that are greater than 100.

```
In [42]: np.count_nonzero(sim_W_H5 > 100) / 10000
```

```
Out[42]: 0.1856
```

#newpage

## 1.5 Part 3: Other Patterns

In this part of the lab, the patterns aren't just runs of successes. Instead, they include failures.

**Note:** Markov Chains are often used to find these expectations. But that method ends up with pretty much the same set of equations that you will derive here, and the equations are cumbersome to solve when the patterns are long. Part 4 of the lab takes care of that problem.

### 1.5.1 3a) Decomposing $W_{HT}$

Suppose the first few trials are $TTTTHHHTHHT$. Identify the value of $W_{HT}$ in this case, and then fill in the blank below:

To get to $W_{HT}$, I waited till the first $H$ and then I waited till the first _____.

T

The cell below generates 50 i.i.d. Bernoulli $(0.4)$ variables, so $H$ is 1 and $T$ is 0 as usual. Run the cell and look for the first occurrence of the pattern 10. Is your observation above still true?

Run the cell several times, and check every time whether your observation holds. It is possible, though not likely, that you don't see 10 in the 50 trials. That's fine. Just run the cell again.

```
In [25]: stats.bernoulli.rvs(0.4, size=50)
```

```
Out[25]: array([0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 1, 0, 0, 1])
```

Fill in the blanks:

$W_{HT} = W_H + V$ where $V$ is independent of $W_H$ and has the _____ distribution.

geometric (q)

Find $E(W_{HT})$ in terms of $p$ and $q$.

1/p + 1/q

### 1.5.2 3b) Comparing $W_{HH}$ and $W_{HT}$ When $p = 0.5$

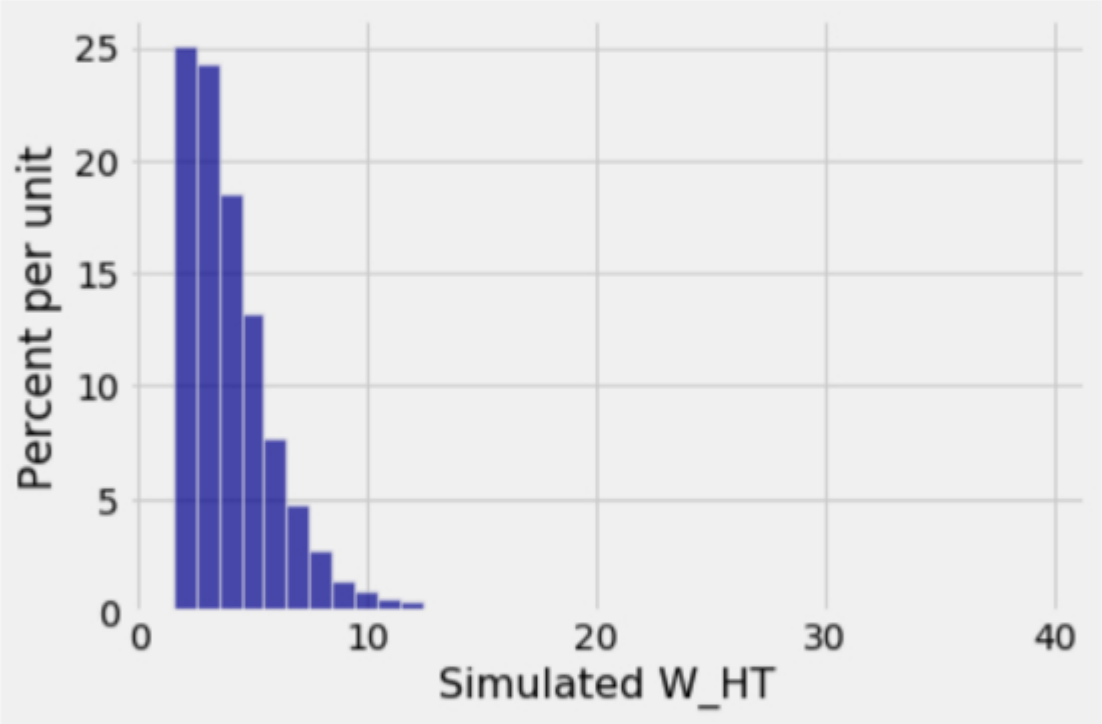
Use your answers in **2d** and **3a** to find $E(W_{HH})$ and $E(W_{HT})$ in the case $p = 0.5$.

From 2d, E(W_HH) = 6. By 3a, E(W_HT) = 2 + 2 = 4.

The two answers aren't equal, even though the coin is fair and the patterns have the same length.

To see why, start by completing the cell below. Use 10,000 simulated values and remember that $p = 0.5$.

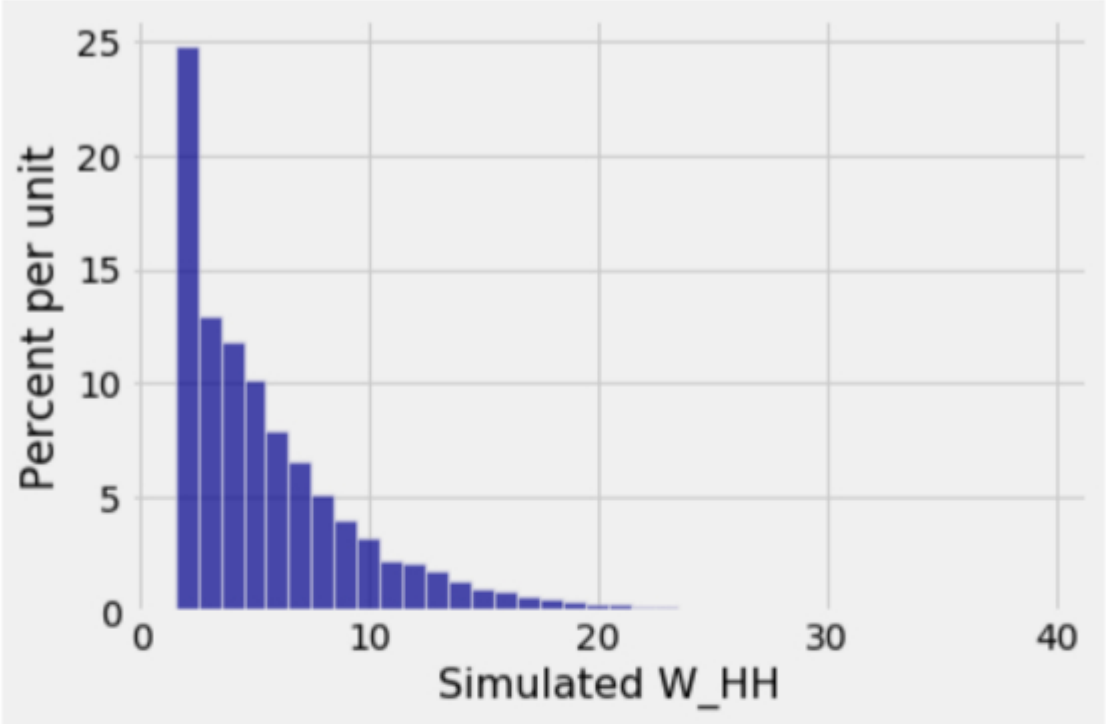
In [43]:
```
# Fair coin: Empirical histogram of W_HT based on 10,000 simulated values

sim_W_HT = stats.geom.rvs(0.5, size = 10000) + stats.geom.rvs(0.5, size = 10000)

Table().with_column('Simulated W_HT', sim_W_HT).hist(bins = np.arange(1.5, 40.5))
```




Now compare with the empirical distribution of $W_{HH}$ below. You drew a histogram of that distribution in **2c**; the code is repeated below for ease of reference.

Also use the subsequent code cell to find the means of the two sets of simualted values and check that they are consistent with your answers at the start of this exercise.

In [44]:
```
# Fair coin: Empirical histogram of W_HH based on 10,000 simulated values

Table().with_column('Simulated W_HH', sim_W_HH).hist(bins = np.arange(1.5, 40.5))
```




```
In [45]: np.mean(sim_W_HH), np.mean(sim_W_HT)
```

```
Out[45]: (5.967, 4.0275)
```

The histograms clearly show why the two expectations are not equal. But you can also see the reason with minimal calculation and no graphs, as follows.

Fill in the blanks with patterns:

To compare $W_{HH}$ and $W_{HT}$ in tosses of a fair coin, - in both cases it takes the same expected time to first get to $W_H$; - in both cases, the pattern appears on the next toss with the same chance 0.5; - in both cases, the pattern fails to appear on the next toss with the same chance 0.5, and then: - to get to $W_{HH}$ you have to wait till _____ appears in the subsequent tosses, whereas - to get to $W_{HT}$ you have to wait till _____ appears in the subsequent tosses.

HH, T

### 1.5.3 3c) [ON PAPER] Expectation of $W_{HTH}$

Find $E(W_{HTH})$ by conditioning on $W_{HT}$. By now you're an expert on how to do this. Follow the process in **1b** and **1c**.

Let $x = E(W_{HTH})$. Then $x = E(W_{HT}) + 1p + (1 + x)q$, and so $px = \frac{1}{p} + \frac{1}{q} + 1$.

So $x = \frac{1}{p^2} + \frac{1}{pq} + \frac{1}{p}$.

#newpage

## 1.6 Part 4: A Pattern in the Answers

For several different patterns of successes and failures, you have calculated the expected waiting time till the pattern appears. Each calculation depended on the particular pattern. It's not unreasonable to think that the longer or more complicated the pattern, the more complicated the calculation will be.

**But part of the job of a data scientist, or indeed any scientist or mathematician, is to notice patterns in what they have calculated. These insights help develop further results.**

In this part of the lab you will take a more careful look at the expected waiting times that you have calculated thus far, and use them to make a conjecture about a straightforward formula for the expected waiting time till any pattern.

You will check your conjecture by calculation in a couple of cases. For a proof that works in general, take Stat 150 to learn about martingale methods.

### 1.6.1 4a) [ON PAPER] Beginning and End

(1) Take a look at the formula that you derived in **3a** for $E(W_{HT})$.

If you run two trials, what is $P(HT)$ in terms of $p$ and $q$? Write $E(W_{HT})$ in terms of $P(HT)$.

(2) Now look at the formula for $E(W_{HTH})$ in **3c**. Simplify it if needed till you see a term involving $P(HTH)$ analogous to what you found in (1) in the case of $HT$. But there's more in the formula for $E(W_{HTH})$. So now compare the end of the pattern to the beginning of the pattern $HTH$. Specifically, compare:

- the first and last elements of the pattern
- the first two and the last two elements of the pattern

Notice how when you are at the end of the pattern you are also partway into another occurrence of the pattern.

Look again at the formula for $E(W_{HTH})$. How can you connect it with the comparisons that you have made?

(3) Repeat (2) for $E(W_{HHH})$ instead of $E(W_{HTH})$. You found $E(W_{HHH})$ in **1b**.

(1) $P(HT) = pq$, $E(W_{HT}) = \frac{1}{p} + \frac{1}{q} = \frac{1}{pq} = \frac{1}{P(HT)}$

(2) The first and last elements are both H.

$P(HTH) = pqp = p^2q$.
Now $E(W_{HTH}) = \frac{q+p+pq}{p^2q} = \frac{1}{pqp} + \frac{1}{p} = \frac{1}{P(HTH)} + \frac{1}{P(H)}$

That's the sum of the inverses of two chances: the chance of the pattern and the chance of the overlap in the beginning and the end.

(3) In $HHH$ the first and last elements are equal, and also the first two and last two. As above, the expected waiting time $\frac{1}{p^3} + \frac{1}{p^2} + \frac{1}{p}$ is the sum of inverses of the chance of the pattern and the chances of the two different beginning-end overlaps.

### 1.6.2 4b) [ON PAPER] Your Conjecture

By now you should have a pretty good guess as to how these expected waiting times work out in general. It might be a bit of a pain to write out the guess in general, so just provide your guess for the expected waiting time till each of the following patterns. No explanations needed.

(1) $HTT$

(2) $HTTH$

(3) $HTHTHT$

(4) $HHHTHHH$

(5) $HHTTHHT$

(1) $\frac{1}{pq^2}$ (2) $\frac{1}{pq^2p} + \frac{1}{p}$
(2) $\frac{1}{pqpqpq} + \frac{1}{pqpq} + \frac{1}{pq}$
(3) $\frac{1}{p^3qp^3} + \frac{1}{p^3} + \frac{1}{p^2} + \frac{1}{p}$ (5) $\frac{1}{p^2q^2p^2q} + \frac{1}{p^2q}$

### 1.6.3 4c) [ON PAPER] Check Your Conjecture

Checking these guesses by direct calculation can be laborious, but for the short patterns it is well worth knowing how to find the expected waiting time by conditioning.

Check your guess for $E(W_{HTT})$ as follows.

- Notice that you have to first wait for $HT$. In **3a** you have already calculated the expected time needed for that.
- Let $x = E(W_{HTT})$. This is what you are trying to find.
- Suppose the first trial has been conducted. Let $Y$ be the *additional* number of trials till the pattern $HTT$ appears. For example, if the first four trials are $THTT$ then $Y = 3$; if the first three trials are $HTT$ then $Y = 2$; and so on. Let $y = E(Y \mid \text{the first trial is } H)$.

Fill in the blanks with math expressions involving $x$ and $y$:

$$x \;=\; E(W_{HT}) \;+\; (1 + \_\_\_\_\_)p \;+\; (\_\_\_\_\_)q$$

$$y \;=\; (1 + \_\_\_\_\_)p \;+\; (2 + \_\_\_\_\_)qp \;+\; (\_\_\_\_\_)q^2$$

Hence find $x = E(W_{HTT})$ and simplify it if necessary to show that it is the same as your conjecture in **4b**.

$$x \;=\; E(W_{HT}) \;+\; (1+y)p \;+\; 1q$$

$$y \;=\; (1+y)p \;+\; (2+y)qp \;+\; 2q^2$$

So $x = E(W_{HT}) + 1 + py$ and $(1 - p - qp)y = p + 2qp + 2q^2$.
From **4a** we know $E(W_{HT}) = \frac{1}{pq}$. Also $p + q = 1$ so $p + 2qp + 2q^2 = p + 2q = 1 + q$.

$x = \frac{1}{pq} + 1 + py$ and $q^2 y = 1 + q$. So

$$x = \frac{1}{pq} + 1 + p\frac{1+q}{q^2} = \frac{q + pq^2 + p^2 + p^2 q}{pq^2} = \frac{q + pq + p^2}{pq^2} = \frac{1}{pq^2}$$

as conjectured. For the algebra in the final step, look for factors of $p + q$ in the numerator.

### 1.6.4 4d) [ON PAPER] Robot at a Typewriter

By now you have a pretty good sense of how these expected waiting times work when the patterns involve just two symbols. So you're ready to branch out beyond just success/failure trials.

Suppose letters are drawn uniformly at random with replacement from the 26 upper case letters of the English alphabet.

(1) Find $E(W_A)$, the expected number of draws till the letter $A$ appears. No conjectures here — use what you have already shown.

(2) Find $E(W_{AZ})$ by developing two simultaneous equations in two unknowns, as follows. The process is analogous to that in **4c**.

- Let $x = E(W_{AZ})$. This is what you are trying to find.
- Suppose the first draw has been made. Let $Y$ be the *additional* number of draws till the pattern $AZ$ appears. So $Y \geq 1$. Let $y = E(Y \mid \text{first draw is } A)$.

Fill in the blanks with math expressions involving $x$ and $y$.

$$x = \underline{\qquad\qquad} \cdot \frac{25}{26} + \underline{\qquad\qquad} \cdot \frac{1}{26}$$

$$y = \underline{\qquad\qquad} \cdot \frac{24}{26} + \underline{\qquad\qquad} \cdot \frac{1}{26} + \underline{\qquad\qquad} \cdot \frac{1}{26}$$

Now find $x = E(W_{AZ})$. It's a good idea to start by multiplying both sides of the two equations by 26.

(3) Adapt the method in (2) above to find $E(W_{AA})$. Your answer should be in terms of powers of 26.

(1) 26. $W_A$ is geometric $(1/26)$.

(2)

$$x = (1+x) \cdot \frac{25}{26} + (1+y) \cdot \frac{1}{26}$$

$$y = (1+x) \cdot \frac{24}{26} + (1+y) \cdot \frac{1}{26} + 1 \cdot \frac{1}{26}$$

That is, $x = 26 + y$ and $25y = 26 + 24x$. So $y = 25 \cdot 26$ and $x = 26^2$.

(3) Let $x = E(W_{AA})$ and let $Y$ be the additional number of draws required after the first draw till $AA$ appears. Let $y = E(Y \mid \text{first draw is } A)$.

Then $x = (1+x) \cdot \frac{25}{26} + (1+y) \cdot \frac{1}{26}$ and $y = (1+x) \cdot \frac{25}{26} + 1 \cdot \frac{1}{26}$.
So $x = 26 + y$ and $26y = 26 + 25x$. So $y = 26^2$ and $x = 26 + 26^2$.

### 1.6.5 4e) Extending Your Conjecture

Compare your answers to **4d** with your conjecture in **4b**, and remember that the chance of each letter is 1/26 on a single draw.

Now extend your conjecture in **4b** to guess the expected waiting time till each of the following patterns. The expressions should all be in terms of powers of 26.

*BOO, BOB, GAGA, RADIOGAGA, ABRACADABRA*

In [61]:
```
# A
ev_W_A = 26

# AZ
ev_W_AZ = 26**2

# AA
ev_W_AA = 26**2 + 26

# BOO
ev_W_BOO = 26**3

# BOB
ev_W_BOB = 26**3 + 26

# GAGA
ev_W_GAGA = 26**4 + 26**2

# RADIOGAGA
ev_W_RADIOGAGA = 26**9

# ABRACADABRA
ev_W_ABRACADABRA = 26**11 + 26**4 + 26

ev_W_A, ev_W_AZ, ev_W_AA, ev_W_BOO, ev_W_BOB, ev_W_GAGA, ev_W_RADIOGAGA, ev_W_ABRACAD
```

Out[61]:
```
(26, 676, 702, 17576, 17602, 457652, 5429503678976, 3670344487444778)
```

Curious about the proof? After the lab deadline, we will provide a sketch of a "fair gambling game" argument for this. If you want all the details, take Stat 150.

## 1.7 Conclusion

What you have learned:

- How to simulate discrete random variables in Python
- A formula for the expected waiting time till a success run of a specified length
- Approximate distributions of waiting times till success runs
- Expected waiting times till patterns involving failures
- A pattern in these expected waiting times
- How to write down the expected waiting time till any pattern, just by looking at the pattern

## 1.8 Submission Instructions

Many assignments throughout the course will have a written portion and a code portion. Please follow the directions below to properly submit both portions.

### 1.8.1 Written Portion

- Scan all the pages into a PDF. There are many free apps available that allow you to convert your work into PDFs from your phone, or you can use a scanner. Please **DO NOT** simply take pictures using your phone.
- Please start a new page for each question. If you have already written multiple questions on the same page, you can crop the image or fold your page over (the old-fashioned way). This helps expedite grading.
- It is your responsibility to check that all the work on all the scanned pages is legible.

### 1.8.2 Code Portion

- Save your notebook using File > Save and Checkpoint.
- Download the PDF file and confirm that none of your work is missing or cut off.

### 1.8.3 Submitting

- Combine the PDFs from the written and code portions into one PDF. Here is a useful tool for doing so.

- Submit the assignment to Lab4 on Gradescope.
- **Make sure to assign each page of your pdf to the correct question.**

### 1.8.4 We will not grade assignments which do not have pages selected for each question or were submitted after the deadline.

# fa18_midterm

October 1, 2019

Probability for Data Science
UC Berkeley, Spring 2019
Ani Adhikari and Jim Pitman


## 1 Fall 2018 Midterm

This set consists of the problems from the Fall 2018 midterm. It only involves written work; there is no code on the midterm.

### 1.0.1 Recommendation

Use this practice as an opportunity to assess your preparation for the upcoming midterm. First complete the bulk of your review for the exam. **After your review, set aside all course materials, find a place where you can work on your own, and give yourself 75 minutes to do the assignment.** The problems that you are not able to do in that time will give you an indication of where you need to focus the remaining portion of your review.

### 1.0.2 1.

A box contains 1 red ticket and 1 blue ticket.

- A ticket is drawn at random, and then returned to the box along with 2 more tickets of its color and 1 more ticket of the other color.
- Then another ticket is drawn at random from the box.

Let $R_1$ be the event that the first ticket drawn is red, and let $R_2$ be the event that the second ticket drawn is red.

**a)** Find $P(R_2)$ and compare it to $P(R_1)$. Which is bigger? Or are they equal?

**b)** Find $P(R_1 \mid R_2)$ and compare it to $P(R_2 \mid R_1)$. Which is bigger? Or are they equal?

### 1.0.3 2.

On one day, an advisor offers students 12 consecutive 15-minute slots for appointments: 9:00-9:15, 9:15-9:30, and so on, up to 11:45-12 noon. That's four slots in each of the three hours 9-10, 10-11, and 11-12.

Five students sign up for one slot each. You can assume that the chosen slots are five draws made at random without replacement from the 12 available slots.

**a)** Find the chance that exactly two of the chosen slots are in the 9-10 hour.

**b)** Find the chance that the chosen slots consist of two in the 9-10 hour, two in the 10-11 hour, and one in the 11-12 hour.

**c)** Find the chance that there is one chosen slot in one of the three hours and two chosen slots in each of the other two hours.

**d)** Find the **decimal value** of the expected number of slots after the last (that is, the one with the latest start time) of the chosen slots.

#### 1.0.4 3.

A population consists of individuals in three categories, in the following proportions.

| Category | A | B | C |
|---|---|---|---|
| **Proportion** | 0.1 | 0.6 | 0.3 |

Let the random variable $N$ have the Poisson $(30)$ distribution, and consider a sample of size $N$ drawn at random with replacement from the population.

Please remember the exam instruction: **Don't leave infinite sums in any answer.**

**a)** Find the chance that exactly one of the categories does not appear in the sample.

**b)** Find the expected number of categories that appear fewer than 10 times in the sample.

#### 1.0.5 4.

Consider the Metropolis algorithm for Markov Chain Monte Carlo as described in class, with the same notation as in class:

- strictly positive probability distribution $\pi$ on the finite state space
- proposal transition matrix $\mathbb{Q}$, with $Q(i,j)$ denoting its $(i,j)$ element
- Markov chain \$X_0, X_1, X_2, ...\$ with transition matrix $\mathbb{P}$ that has $(i,j)$ element $P(i,j)$ constructed using the proposal matrix $\mathbb{Q}$ and additional randomization based on the ratio $\pi(j)/\pi(i)$

As we showed in class, the algorithm ensures that the constructed chain $X_0, X_1, X_2, \ldots$ has $\pi$ as its steady state distribution.

**a)** Is the constructed chain $X_0, X_1, X_2, \ldots$ reversible? Why or why not?

**b)** The proposal matrix $\mathbb{Q}$ is always symmetric because that is a requirement of the Metropolis algorithm. Is the transition matrix $\mathbb{P}$ of the constructed chain \$X_0, X_1, X_2, ...\$ also always symmetric? Why or why not?

**c)** For states $i$ and $j$, define the function $s$ by

$$s(i,j) = \min\left(1, \frac{\pi(j)}{\pi(i)}\right)$$

Write a formula for $P(X_{n+1} \neq i \mid X_n = i)$ in terms of elements of $\mathbb{Q}$ and values of the function $s$. Justify your formula.

### 1.0.6 5.

A random number generator draws repeatedly at random with replacement from the 10 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Let $W$ be the number of draws till the digits 4, 5, and 6 have appeared. For example, if the sequence of draws starts out with 92261344685 then $W = 11$.

**a)** Find $E(W)$.

**b)** For $k \geq 3$, find $P(W > k)$.

**c)** Use the answer to Part (b) to find the distribution of $W$.

### 1.0.7 6.

I have $d$ dice. I roll them in stages as follows.

**Stage 1:** All $d$ dice are rolled. Those that show six spots are removed.

**Stage 2:** If dice remain after Stage 1, they are rolled. Those that show six spots are removed.

**Stage 3:** If dice remain after Stage 2, they are rolled. Those that show six spots are removed.

$\$\ldots\$$ And so on, till no dice are left.

**a)** What is the distribution of the number of dice rolled in Stage 3?

[No long calculations are needed here. Just think about an individual die.]

**b)** Let $N$ be the number of stages till no dice are left. Find the cumulative distribution function (cdf) of $N$.

[Keep thinking about an individual die. When is it removed?]