Lab_03

February 18, 2025

Probability for Data Science

UC Berkeley, Spring 2025

Michael Xiao and Ani Adhikari

[91]:
```
import warnings
warnings.filterwarnings('ignore')

from datascience import *
from prob140 import *
import numpy as np
from scipy import stats

import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
%matplotlib widget

import ipywidgets as widgets
from ipywidgets import interact
```

0.0.1 Instructions

Similar to your homeworks, your labs will generally 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.
- Start the work for the written portions of each section on a new page. - You are welcome to LaTeX your answers to the written portions, but staff will not be able to assist you with LaTeX related issues. - Show your work. Give reasoning. The question isn't always going to ask for it, because we assume that you will provide justification for your answers. Every answer should contain a calculation, reasoning, or diagrams that are clearly labeled to show what's going on. - It is your responsibility to ensure that both components of the lab are submitted completely and properly to Gradescope. **Make sure to assign each page of your pdf to the correct question. - Refer to the bottom of the notebook for submission instructions.**

1 Lab 3: Chinese Restaurant Process (Part A due 5pm Feb 10, Part B due 12pm noon Feb 18)

The word *stochastic* means random, or determined by a probability distribution. In this lab you will analyze a stochastic model for *clustering*, which is a process of visualizing and organizing data based on similarities between sampled individuals. Methods of identifying clusters have applications in a wide variety of areas. Here are just a few examples.

- Biology: populations segmented by patterns in genes
- Criminology: locations where a type of crime occurs
- Marketing: customers who share preferences
- Natural Language Processing: words grouped by similarity of meaning or grammatical properties

One commonly used model is *k-means clustering*, which assumes that the data are continuous numerical variables and that the number of clusters in the population is known to be k. By contrast, the *Chinese Restaurant Process* is a discrete time stochastic process (that is, a random sequence observed at times 0, 1, 2, etc) that can be used as a model for clustering and does not assume a fixed number of clusters. Instead, the clusters evolve randomly as individuals enter the system, according to a specified probabilistic structure. The data could be categorical or numerical.

This process is an example of a *generative Bayesian model* which specifies the probabilstic rules by which the process evolves and in which the representation itself evolves as more data come in.

History: The process has its origins in the work of Warren Ewens in the early 1970's, in particular the Ewens Sampling Formula of population genetics. Since then, the development of the theory of the stochastic process and its use in machine learning has been very much a Berkeley enterprise: Jim Pitman, David Aldous, and Mike Jordan are among the people involved. The restaurant analogy is due to Jim Pitman and our late colleague Lester Dubins, during one of their regular cafe meetings decades ago. Though there is a popular story that they came up with it while eating at a Chinese restaurant, in fact they were at the Strada on Bancroft and College, known at the time as the Roma.

What you will learn: In Data 140 we will focus on understanding the probability model. You can then go on to other classes, for example in natural language processing, to see how to fit the model to data.

This lab is about the distribution of the total number of clusters observed in a fixed amount of time. This theory helps answer questions like these: - How many different animal species do you expect to see? - What is the distribution of the number of different types of documents that you will have? - What is the chance that everybody is retweeting the same tweet?

The "restaurant" image is that each person entering the restaurant chooses a table to join. Thus each table becomes a cluster. The model allows for an infinite number of tables, each of infinite size, though of course if you make n observations then the observed number of occupied tables will be finite. That's the number of different clusters observed.

In this lab you will study the random number of observed clusters. In Part A you will simulate the Chinese Restaurant process. In Part B you will study the distribution and expectation of the number of clusters, both empirically and analytically.

1.1 Preliminary

To understand what follows, it will help to work out the two probabilities below. Suppose there are two positive numerical parameters α_1 and α_2. They are just positive numbers; they don't have to be probabilities or integers. Now suppose an event A happens with probability proportional to α_1 while A^c happens with probability proportional to α_2. Fill in the blanks with expressions in terms of α_1 and α_2.

$P(A) =$ _____

$P(A^c) =$ _____

$P(A) = \alpha_1 \ / \ (\alpha_1 + \alpha_2)$

$P(A^c) = 1$ - $P(A)$

1.2 The Process

In keeping with the Chinese Restaurant (CR) analogy, think of clusters as groups of people sitting at the same table. We will only consider occupied tables, so the number of tables is equal to the number of clusters.

The CR process evolves according to the following rules. - There is a positive parameter θ. - People enter the restaurant one at a time. - Person 1 enters and sits at a table which we will call Table 1. - Each subsequent person - either joins an existing table with probability proportional to the number of people already at that table, or - starts a new table with probability proportional to θ. - People choose tables independently of each other.

Don't worry about running out of room. The restaurant has infinite capacity and each table is infinitely large. You can imagine infinitely many such tables at the start, or imagine new tables appearing magically each time a person's random choice is to start a new table. We prefer the second image because it consists only of the occupied tables.

Note that the tables are not numbered at the start. We number them according to their order of formation. Thus Table 1 is the table at which Person 1 sits. Table 2 is the next new table to be formed. We can't say exactly who will start it, because that's random. Table 3 is the third new table to be formed. And so on.

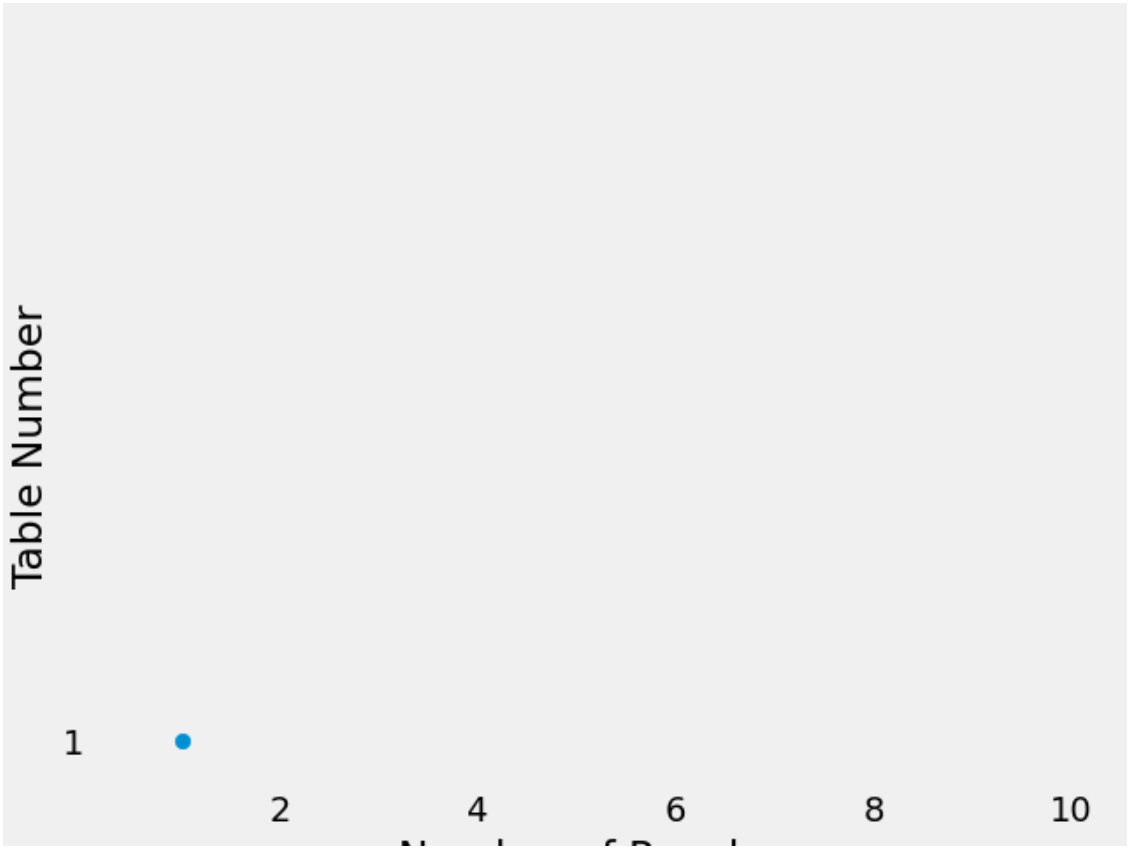
Note on the name: In Prof. A's experience, nobody chooses tables at a restaurant, Chinese or otherwise, in the way described above. A former 140 student once proposed a better analogy of new freshmen choosing tables in a dining hall. But a large restaurant with large round tables was an image that caught on quickly. The CR process now has variations such as the Indian Buffet Process.

To visualize the CR process, run the cell below. It simulates 100 people arriving according to a CR process with some value of θ. Each table (that is, cluster) is represented by a color. Move the slider slowly at first, so that you can see the people coming in one at a time.

Run the cell several times. The value of θ changes each time you run the cell, and you will see quite a bit of variation in the results. This indicates that you might be able to use this model in varied applications, by choosing θ appropriately.

[92]:
```
plt.close()
visualize_cr()
```

```
interactive(children=(IntSlider(value=1, continuous_update=False,␣
↪description='n', min=1), Output()), _dom_cla…
```



2 Part A of the lab starts here. It is due by 5 PM on Monday, February 10th.

2.1 Identify Your Lab Partner

This is a multiple choice question. Please select **ONE** of following options that best describes how you complete Lab 3.

- I am doing Part A of this lab by myself and I don't have a partner.
- My partner for Part A of this lab is [PARTNER'S NAME] with email [berkeley.edu email address]. [SUBMITTER'S NAME] will submit to Gradescope and add the other partner to the group on Gradescope after submission.

Please copy and paste **ONE** of above statements and fill in blanks if needed. If you work with a partner, make sure only one of you submit on Gradescope and that the other member of the group

is added to the submission on Gradescope. Refer to the bottom of the notebook for submission instructions.

Type your answer in this cell.

2.2 We will not grade assignments which do not have pages selected for each question.

2.3 Section 1: Steps towards Simulation

Start by running the cell below to set the parameter θ. We are starting with the value 1. Later in the lab you will vary θ.

[93]:
```
theta = 1
```

The goal is to create an array `people` that shows the number of people at each table. For example, if there are ten people of whom six are at Table 1, three at Table 2, and one at Table 3, then the array created should be `[6, 3, 1]`.

All entries in the array must be positive. The length of the array will be the total number of tables formed. This will be a random number because the choices of tables will be random.

The process starts with the first person entering and sitting at a table which we will call Table 1.

The cell below starts off the array `people` accordingly. Note that `np.array([1])` is the same as `make_array(1)` from the `datascience` library. You are welcome to use `make_array` in later cells if you prefer.

Run the cell.

[94]:
```
people = np.array([1])
people
```

[94]:
```
array([1])
```

To write the code below it might help you to keep the array `[6, 3, 1]` in mind as an example of what `people` might look like after 10 people are seated.

2.3.1 1a) Number of people

How can you use the array `people` to determine how many people are already seated when a new person comes in? Assign this expression to `n` in the cell below. Your code should work not just for the "starter" array `people` created above, but for any `people` array that shows the number of people at each table.

[95]:
```
# The expression should involve the array named people


n = sum(people)
```

Your process hasn't started evolving yet, so at this stage what should the value of `n` be? Run the cell below to check that it gives the right answer. If it doesn't, reset `people` above and then fix your calculation of `n`.

[96]:
```
n
```

[96]:
```
1
```

2.3.2 1b) Table choices for a new person

Use the array `people` to create an array `tbl_choices` that contains the choices of tables that the next newly entering person (Person $n + 1$) will have. There are no probabilities involved at this stage. Just create an array of choices, that is, the labels of all the possible tables at which the new person could sit.

[97]:
```
# Array of all possible table choices that the new person has:
# Use the people array!
# add 2 to take into account a NEW table and existing table

tbl_choices = np.arange(1, len(people) + 2)
```

Thus far, only 1 person has entered. So at this stage `tbl_choices` should reflect the choices that Person 2 has. Run the cell below to make sure this is the case. If it isn't, go back and fix your construction of `tbl_choices` and run the cell again.

[98]:
```
tbl_choices
```

[98]:
```
array([1, 2])
```

2.3.3 1c) [On Paper] Distribution of Table Choices

This is the main probability calculation for constructing the process, so take your time and get it right. Everything else in the lab will be based on this calculation.

Suppose that there are t occupied tables. For $1 \leq i \leq t$, let n_i be the number of people at Table i, and let $\sum_{i=1}^{t} n_i = n$.

For each i in the range $1, 2, \ldots, t, t+1$, write an expression for the chance that the newly entering Person $n+1$ chooses Table i. Your answer should be in terms of θ, n, and $n_1, n_2, \ldots, n_t$.

2.3.4 1d) Coding the distribution

Use your answer to **1c** to reate an array `tbl_probs` containing the corresponding probabilities of selection. That is, the ith element of `tbl_probs` should be the probability that the new person selects the ith element of `tbl_choices`.

[99]:
```
# Array of probabilities of selecting the different tables
# Use the people array!

tbl_probs = people / (n + theta)
tbl_probs = np.append(tbl_probs, theta / (n + theta))
tbl_probs
```

[99]:
```
array([ 0.5,  0.5])
```

Run the cell below to check that `tbl_probs` is a probability distribution.

[100]:
```
sum(tbl_probs)
```

[100]: 1.0

2.3.5 1e) Simulating the new person's choice

Now simulate the new person's choice of table, and assign it to the name `choice`. Use **1b**, **1c**, and `np.random.choice` to do so.

The call `np.random.choice(values, p = probabilities)` makes one random draw from the distribution that has possible values in the array `values` and the corresponding probabilities in the array `probabilities`.

[101]:
```
#generate when second person comes

#OR they create a new table
#existing and new

#tbl_choices should also provide new table

# The random choice made by the new person:
choice = np.random.choice(tbl_choices, p = tbl_probs)
choice
```

[101]: 2

At this stage, your process is just beginning to evolve: you started with one person, and now a new person has made their choice. What could the possible values of `choice` be at this stage?

Run the cell below to confirm that `choice` is one of these possible values. In the comment, provide the other possibility or possibilities.

[102]:
```
choice # Could also have been [fill in this blank]
```

[102]: 2

2.3.6 1f) Updating

Write code that updates the array `people` appropriately based on the random choice made by the new person in **1e**. Your code should work for a new person entering at any stage.

Your code can use any of the quantities calculated in earlier cells: `theta`, `n`, `people`, `tbl_choices`, `tbl_probs`, `choice`. Use as many lines as you need.

If you need to change the element at index `k` of an array `my_array`, for example by multiplying the original element by 2, you can use `my_array[k] = my_array[k] * 2`.

The code isn't complicated but it depends on getting array indices right. Remember that array indices start at 0, not 1.

After you run the cell, `people` should be consistent with `choice` from **1e**. If you need to troubleshoot, remember to Run All Above first, so that all the variables get reset.

[103]:
```
# Update the array people based on the random choice of the new person.

if choice < len(people):
    people[choice - 1] += 1

else:
    people = np.append(people, 1)

choice, people
```

[103]:
```
(2, array([1, 1]))
```

2.4 Section 2: Running the Process

2.4.1 2a) The simulation

Collect the code in Section 1 and use it to define a function `cr` that takes `N` and `theta` as its arguments, runs the CR process till `N` people have been seated, and returns an array of the number of people at each table in the order of table formation.

- If you call your function with the arguments 1 and any positive θ, it should return the array [1]. That represents the one person seated at Table 1.
- If you call your function with the arguments 2 and any positive θ, it should return either the array [2] if both people sit at Table 1, or [1, 1] if Person 2 starts a new table.
- If you call your function with the arguments 3 and any positive θ, then it should return one of the following arrays: [3], [2, 1], [1, 2], [1, 1, 1].
- And so on.

Use as many lines as you need.

[104]:
```
# Create the cr function.

def cr(N, theta):

    people = np.array([])

    for i in np.arange(N):
        n = sum(people)
        tbl_choices = np.arange(1, len(people) + 2)
        tbl_probs = people / (n + theta)
        tbl_probs = np.append(tbl_probs, theta / (n + theta))
        choice = np.random.choice(tbl_choices, p = tbl_probs)

        if choice <= len(people):
            people[choice - 1] += 1

        else:
            people = np.append(people, 1)


    return people
```

2.4.2 2b) Quick check

Perform a basic ballpark check that your function is working right: run the cell below and confirm that the expression is evaluating to the right answer.

[105]:
```
sum(cr(1000, 1))
```

[105]:
```
1000.0
```

2.4.3 2c) Running the simulation

Run the process with $\theta = 1$ and 100 people. Display the simulated results a table that has two columns: - `Table`: The labels of the occupied tables, in order of formation, so that the first entry is 1 and the last entry is the number of tables at the end of the run. - `People at Table`: The number of people seated at the corresponding table.

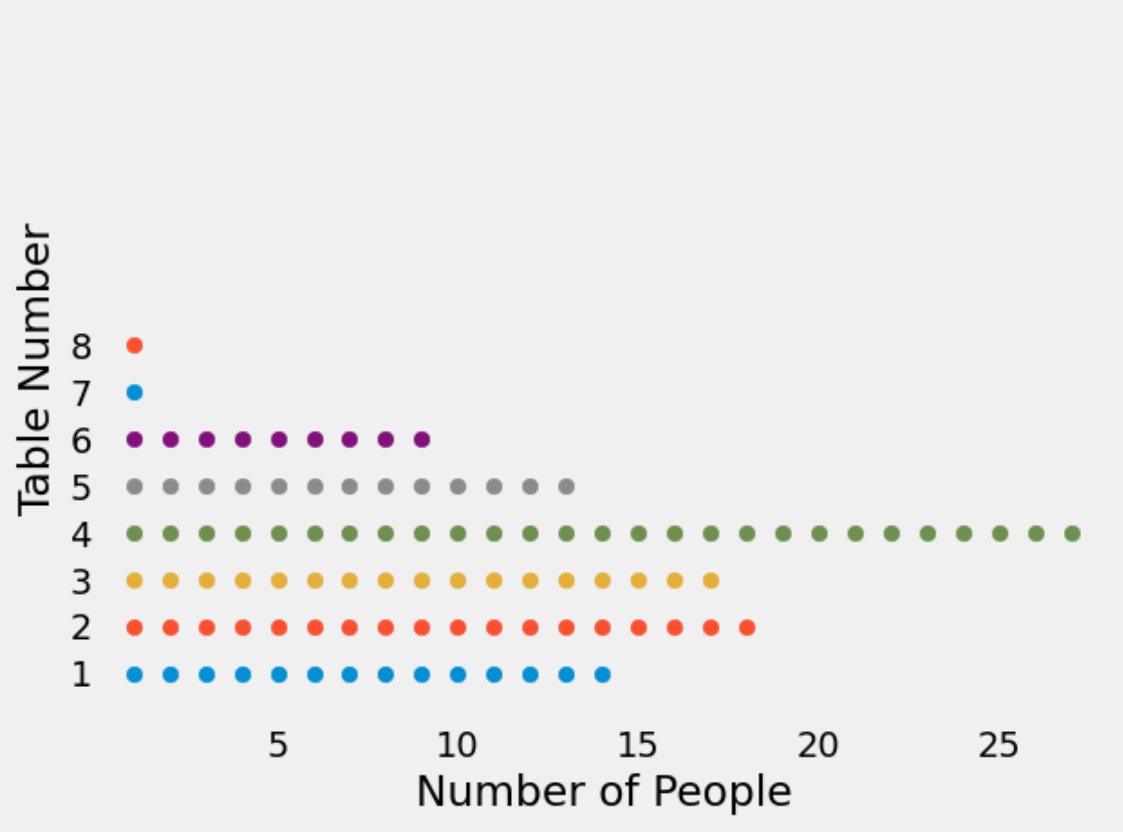
Use as many lines as you need.

Run the cell several times to get a sense of the variability of the results. The last line produces a visualization of the clusters.

[106]:
```
%matplotlib inline
N = 100
theta = 1

people = cr(N, theta)

simulated_process = Table().with_columns(
    'Table', np.arange(1, len(people) + 1),
    'People at Table', people
)

plt.close()
print('Check: Column 1 sum =', sum(simulated_process.column(1)))
simulated_process.show()
visualize_cr(people)
```

```
Check: Column 1 sum = 100.0
```

```
<IPython.core.display.HTML object>
```

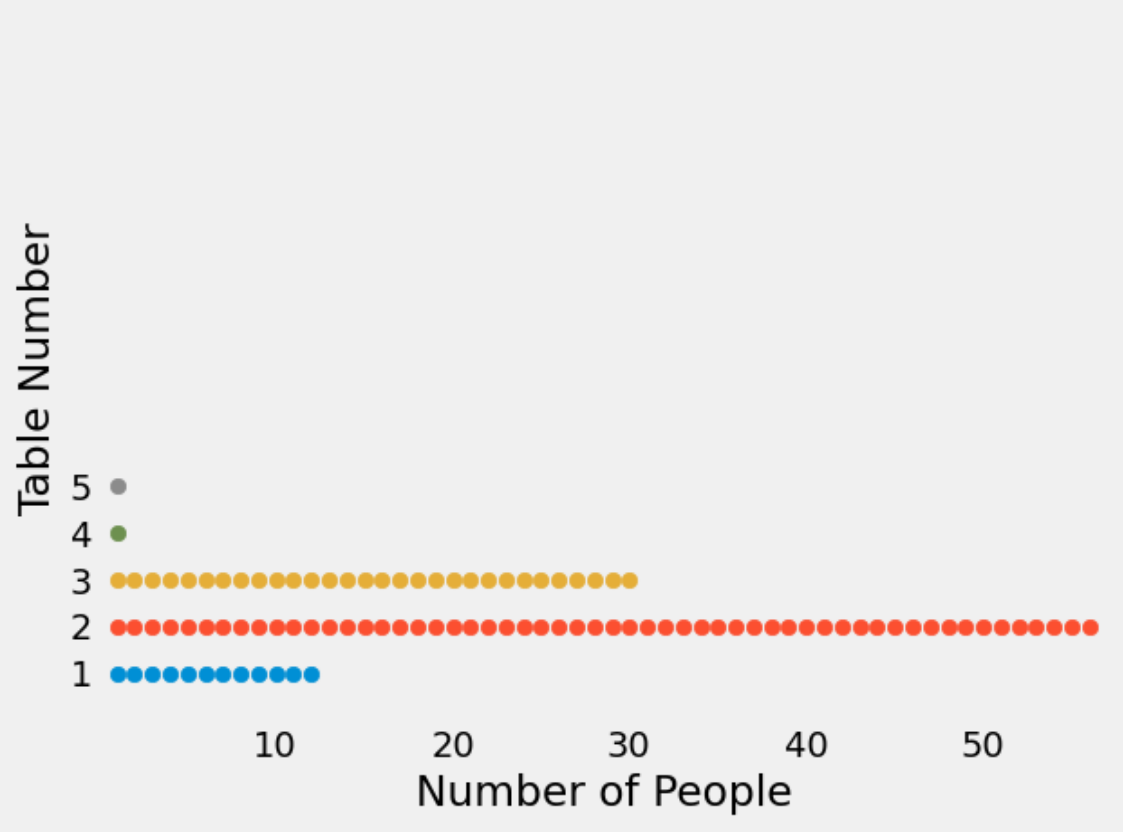


2.4.4 2d) Varying the parameters

Repeat the simulation for varying values of N and θ. Be sure to try $N = 100, 1000$, and 10000 with $\theta = 0.5, 1$, and 2.

For each N, θ pair, run the simulation multiple times. Keep your eye on the number of tables as well as the distribution of people across tables.

[107]:
```
N = 100
theta = 0.5
visualize_cr(cr(N, theta))
```



2.4.5 2e) The nature of the clusters

Give a brief qualitative description of what you have seen in the simulations for $N = 100$ and $\theta = 0.5, 1$, and 2. Address questions such as: - Are there lots of tables, not many, or is it not possible to tell? - Is the distribution of the number of people roughly uniform across the tables? If not, describe what you see about the number of big and small clusters. - In what way does θ make a difference? Or is it not possible to tell?

For $\theta = 0.5$, the distribution exhibits the least variability. In some cases, all individuals are seated at a single table, while in others, multiple tables are used. However, most distributions tend to have a smaller number of tables. When $\theta = 1$, the variability increases, leading to a wider range of table counts. The highest variability is observed at $\theta = 2$, where the number of occupied tables fluctuates the most. Across all simulations, the maximum number of tables observed was 15.

3 Part A of the lab ends here. It is due by 5 PM on Monday, February 10th.

3.1 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.

3.1.1 Written Portion

- Scan all the pages into a PDF. You can use any scanner or a phone using applications such as CamScanner. 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 in CamScanner 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.
- If you used LaTeX to do the written portions, you do not need to do any scanning; you can just download the whole notebook as a PDF via LaTeX.

3.1.2 Code Portion

- Save your notebook using `File > Save Notebook`.
- Generate a PDF file using `File > Save and Export Notebook As > PDF`. This might take a few seconds and will automatically download a PDF version of this notebook.
  - If you have issues, please post a follow-up on the general Lab 3A Ed thread.

3.1.3 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 Lab 3A on Gradescope.
- **Make sure to assign each page of your pdf to the correct question.**
- **It is your responsibility to verify that all of your work shows up in your final PDF submission.**

If you are having difficulties scanning, uploading, or submitting your work, please read the Ed Thread on this topic and post a follow-up on the general Lab 3A Ed thread.

3.2 We will not grade assignments that do not have pages selected for each question.

4 Part B of the lab starts here. It is due by 12 PM noon on Tuesday, February 18th.

4.1 Identify your Lab Partner

This is a multiple choice question. Please select **ONE** of following options that best describes how you complete Lab 3.

- I am doing Part B of this lab by myself and I don't have a partner.
- My partner for Part B of this lab is [PARTNER'S NAME] with email [berkeley.edu email address]. [SUBMITTER'S NAME] will submit to Gradescope and add the other partner to the group on Gradescope after submission.

Please copy and paste **ONE** of above statements and fill in blanks if needed. If you work with a partner, make sure only one of you submit on Gradescope and that the other member of the group is added to the submission on Gradescope. Refer to the bottom of the notebook for submission instructions.

I am doing part b of this lab by myself and I don't have a partner.

4.2 Section 3: Empirical Distribution of the Number of Clusters

4.2.1 3a) Simulating the number of tables

Define a function `num_tables` that takes `N`, `theta`, and `repetitions` (the number of times to simulate the process) as its arguments. In each repetition, the function runs the CR process with `N` people and counts the number of tables occupied by those people once they are all seated. The function `num_tables` should return an array of length `repetitions` consisting of the simulated table counts.

Use as many lines as you need.

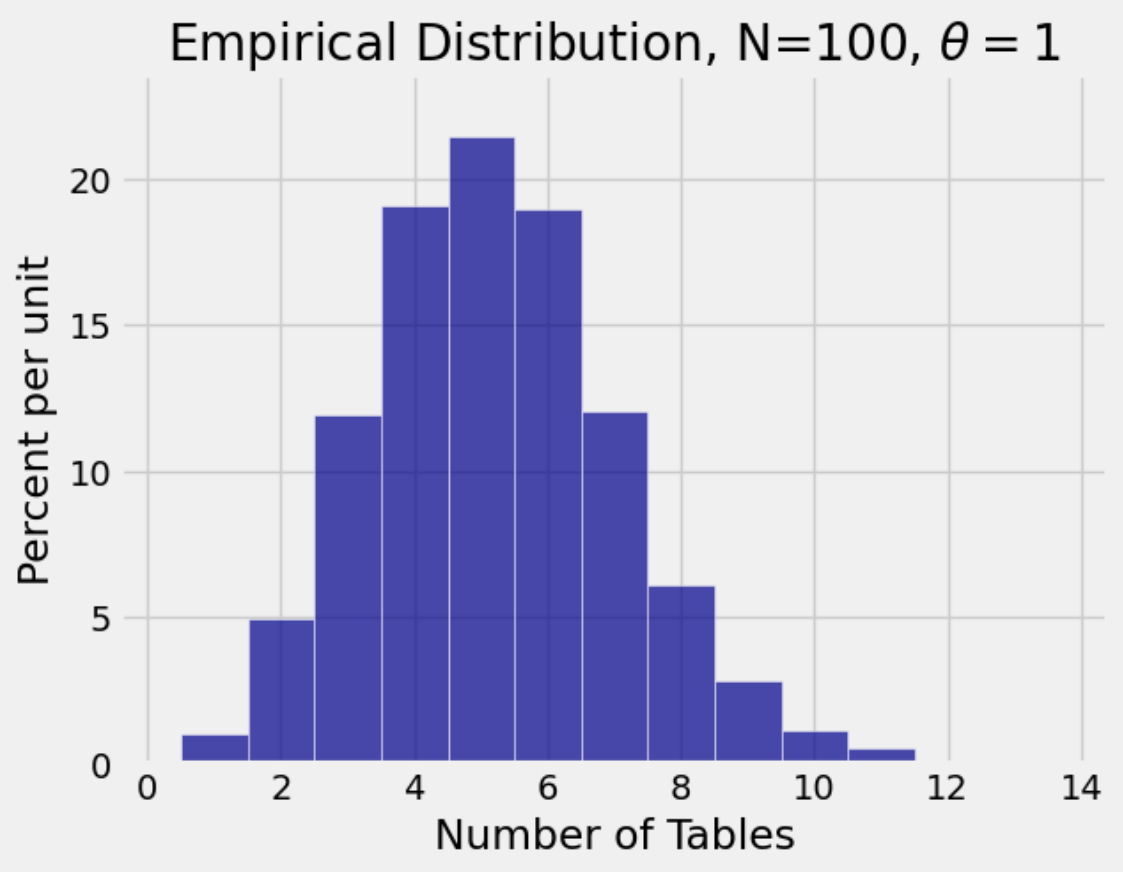
[108]:
```
def num_tables(N, theta, repetitions):
    array_length = []

    for i in np.arange(repetitions):
        table_length = len(cr(N, theta))
        array_length = np.append(array_length, table_length)

    return array_length
```

4.2.2 3b) Empirical distribution of the number of tables

Draw the empirical histogram of the number of tables in the case $N = 100$ and $\theta = 1$, based on 5000 repetitions. Be prepared to wait as the simulation chugs.

The code below uses the `prob140` library method `emp_dist`, which takes an array of integer data and plots its histogram. It saves you the trouble of turning the array into a table and calling `hist` with the appropriate bins.

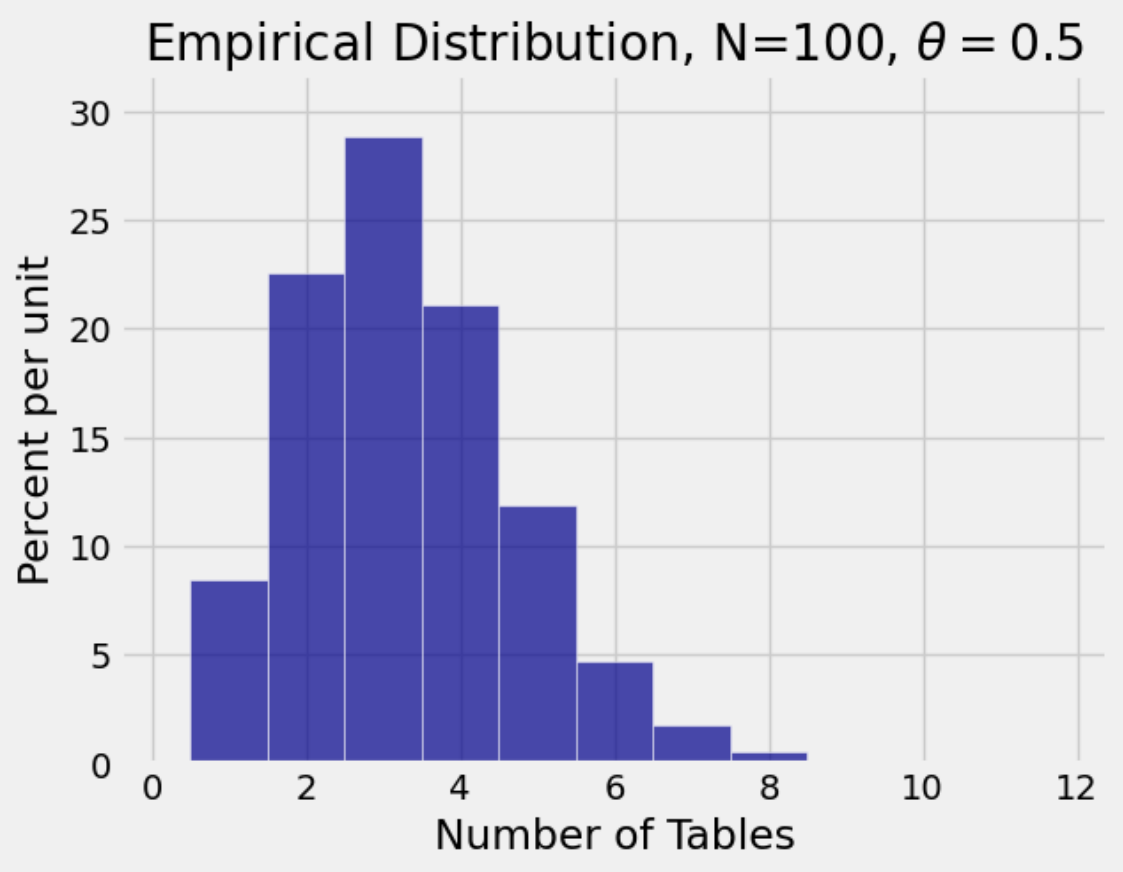
[109]:
```
N = 100
theta = 1
repetitions = 5000
sim_100_1 = num_tables(N, theta, repetitions)


Plot(emp_dist(sim_100_1))
plt.xlabel('Number of Tables')
plt.title('Empirical Distribution, '+'N='+str(N)+', $\\theta=$'+str(theta));
```



4.2.3 3c) Varying the parameters

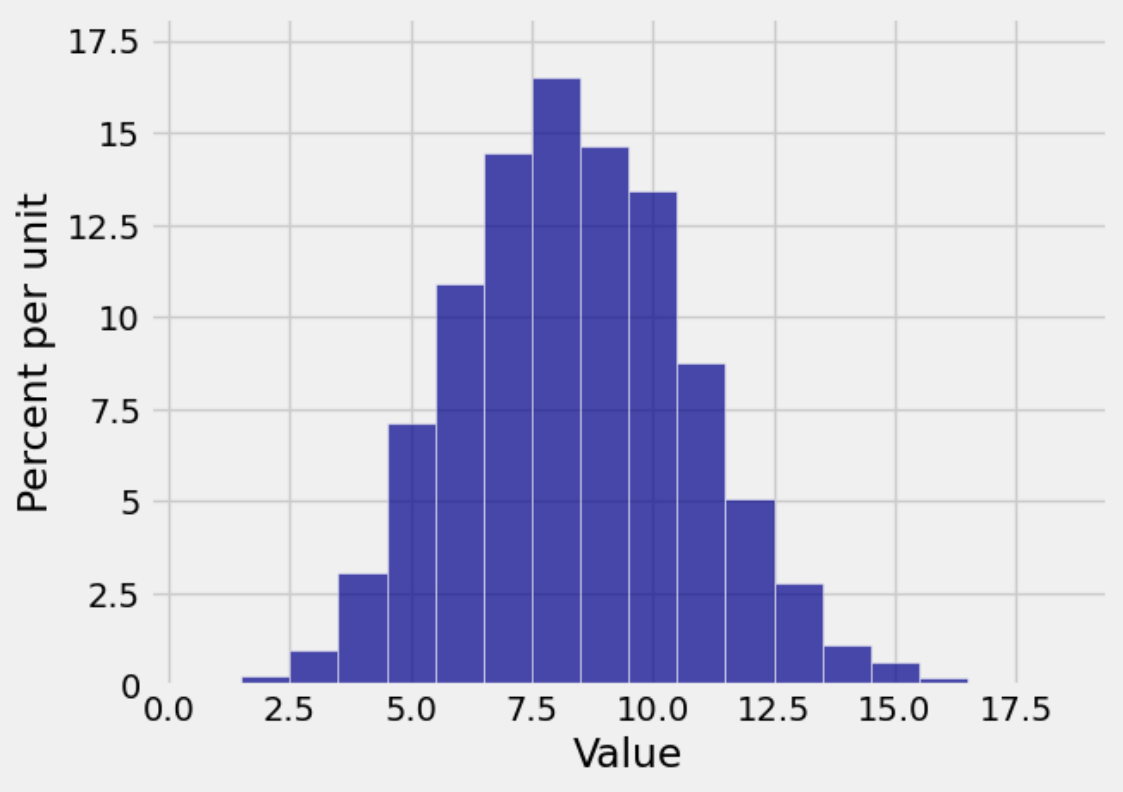
Draw the empirical histogram of the number of tables in the case $N = 100$ and $\theta = 0.5$, based on 5000 repetitions.

[110]:
```
N = 100
theta = 0.5
repetitions = 5000
sim_100_pt5 = num_tables(N, theta, repetitions)

Plot(emp_dist(sim_100_pt5))
plt.xlabel('Number of Tables')
plt.title('Empirical Distribution, '+'N='+str(N)+', $\\theta=$'+str(theta));
```



Now draw the empirical histogram of the number of tables in the case $N = 100$ and $\theta = 2$, based on 5000 repetitions.

[111]:
```
N = 100
theta = 2
repetitions = 5000
sim_100_2 = num_tables(N, theta, repetitions)

Plot(emp_dist(sim_100_2))
```



4.2.4 3d) Consistency check

Are the empirical histograms above consistent with the your answers to **2e**?

Yes, the empirical histograms above are consistent with my answers to 2e. A smaller theta results in fewer tables being created, with the distribution concentrated around lower values. Likewise, a bigger theta leads to more tables being created, with more spread in the histogram. This matches the expectation that as theta increases, the probability of customers sitting at new tables also increases.

4.3 Section 4: Expected Number of Clusters

4.3.1 4a) [ON PAPER] The expectation, analytically

Fix a positive integer N and suppose the process is run till N people have been seated. Let T_N be the number of tables, which is the same as the number of clusters. Find $E(T_N)$ in terms of N and θ.

It is helpful to note that the number of tables is equal to the number of people who started new tables.

4.3.2 4b) The expectation, numerically

Define a function `ev_num_clusters` that takes N and θ as arguments and returns $E(T_N)$.

[112]:
```
def ev_num_clusters(N, theta):
    arr_expectations = []
    for n in np.arange(1, N + 1):
        expectation_Tn = (theta/(theta + (n - 1)))
        arr_expectations = np.append(arr_expectations, expectation_Tn)
    return sum(arr_expectations)
```

To check that your function is working, run it below with the arguments $N = 100$ and $\theta = 0.5$. Compare the output with the empirical mean of your simulated counts `sim_100_pt5` in **3c**. The two numbers should be close.

[113]:
```
emp_mean_100_pt5 = np.mean(sim_100_pt5)
ev_num_clusters(100, 0.5), emp_mean_100_pt5
```

[113]:
```
(3.2843421893016322, 3.3012000000000001)
```

Repeat this test for the pairs $N = 100$, $\theta = 1$ and $N = 100$, $\theta = 2$, using the appropriate simulated arrays from **3b** and **3c**

[114]:
```
emp_mean_100_1 = np.mean(sim_100_1)
ev_num_clusters(100, 1), emp_mean_100_1
```

[114]:
```
(5.1873775176396206, 5.202)
```

[115]:
```
emp_mean_100_2 = np.mean(sim_100_2)
ev_num_clusters(100, 2), emp_mean_100_2
```

[115]:
```
(8.3945570154772593, 8.3872)
```

4.3.3 4c) Rate of growth

The sum $\sum_{i=1}^{k} \frac{1}{i}$ grows slowly with k. Its value is roughly $\log(k)$ for large k.

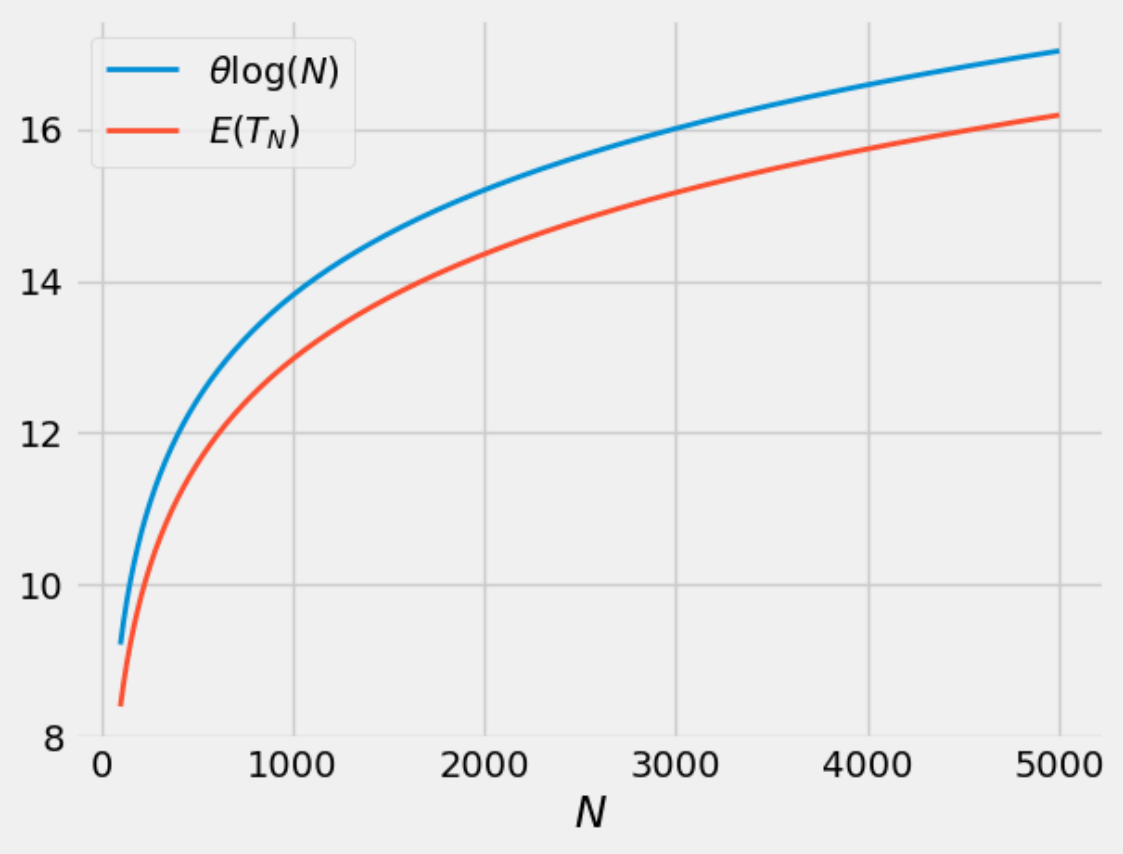
Set $\theta = 2$. On the same axes, plot as a function of N in the range 100 to 5000:

- Your answer to **4a**

- $\theta \log(N)$

The code below uses `matplotlib`, specifically `plt.plot`, to draw the plots. The required first two arguments are an array of the values on the horizontal axis and an array of the values on the vertical axis. Your job is to create `array_N`, `theta_log_N`, and `expected_values` appropriately.

[116]:
```
theta = 2

array_N = np.arange(100,5001)

theta_log_N = theta * np.log(array_N)

expected_values = []

for i in array_N:
    expected_values = np.append(expected_values, ev_num_clusters(i, theta))


plt.plot(array_N, theta_log_N, lw=2, label='$\\theta\log(N)$')
plt.plot(array_N, expected_values, lw=2, label='$E(T_N)$')
plt.xlabel('$N$')
plt.legend();
```



The graphs should justify the statement, "The expected number of clusters grows like $\theta \log(N)$."

4.4 Section 5: Distribution of the Number of Clusters

In Section 3 you simulated this distribution. It is now time to describe the distribution analytically. You already have its expectation in Section 4.

Since the first person always starts a new table, the randomness in the number of tables T_N is based on the number of new tables started by persons 2 through N. So let's write T_N as $T_N = 1 + R_N$ where R_N is the number of new tables started by Persons 2 through N.

4.4.1 5a) [On Paper] Exact distribution and parameters

The *Poisson-binomial* distribution is the distribution of the number of successes in a fixed number of independent Bernoulli trials that need not be identically distributed. The parameter is a list whose i-th entry is the probability of success for the i-th trial; the number of entries in this list of probabilities is the same as the number of trials.

Review the method you used in **4a** to find $E(T_N)$, the expected number of tables formed by N people. Then fill in the blank below with the parameters of the distribution and **explain your answer**.

R_N has the Poisson-binomial distribution with parameters _____.

4.4.2 5b) [On Paper] Poisson approximation

Though in principle we know the exact distribution of R_N, the probabilities its distribution are complicated to calculate. Since each of the $N - 1$ Bernoulli random variables has a different parameter, the chance of each sequence of N zeros and ones doesn't just depend on the number of ones but also on where the ones are.

A good, simple approxmation would be useful. The empirical histograms in the earlier parts of this lab motivate us to try a Poisson approximation. There is a formal justification for this, but we won't go into it.

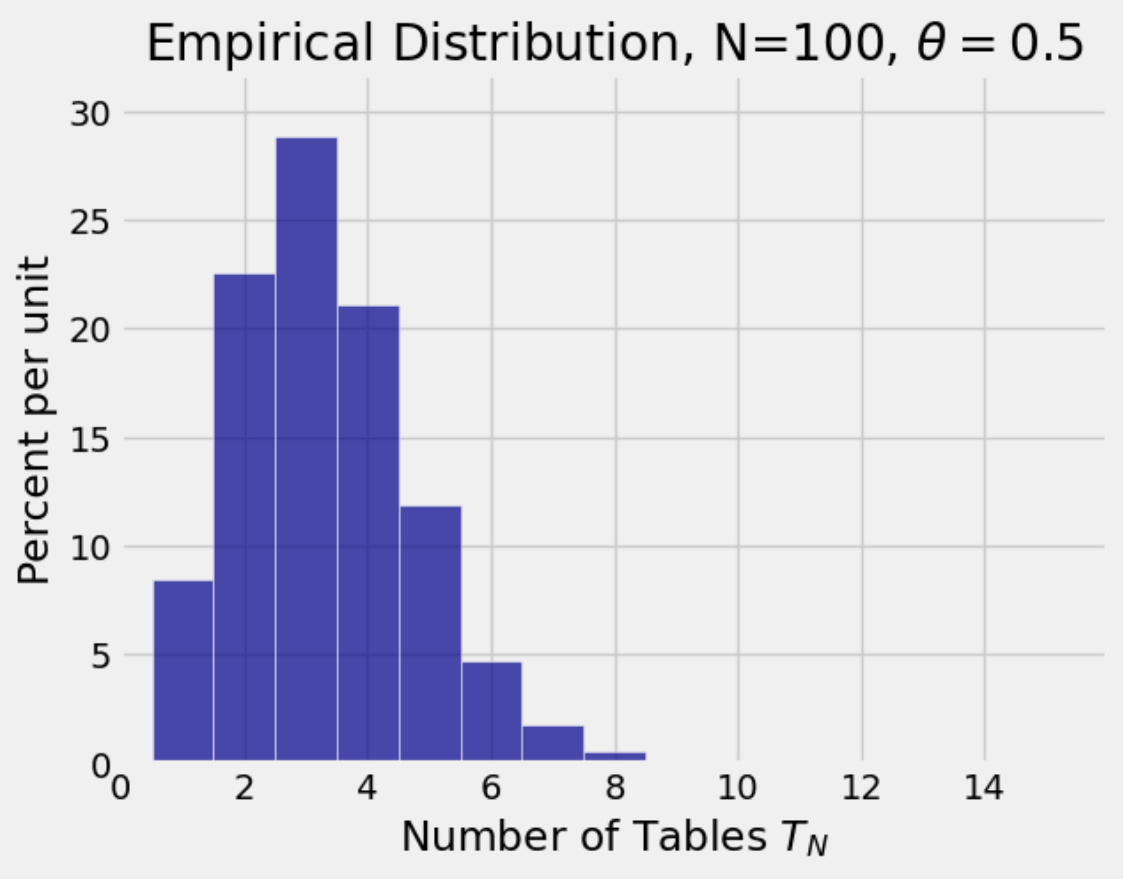
The key is to identify the parameter of the approximating Poisson distribution. Recall what the parameter of a Poisson random variable represents, and use your work earlier in this lab to fill in the blank:

The distribution of R_N is approximately Poisson (_____) .

4.4.3 5c) Visualizing the approximation

In **3c** you drew an empirical histogram of the distribution of T_N in the case $N = 100$ and $\theta = 0.5$. Here it is again.

[117]:
```
N = 100
theta = 0.5

Plot(emp_dist(sim_100_pt5))
plt.xlim(0, 16)
plt.xticks(np.arange(0, 16, 2))
plt.xlabel('Number of Tables $T_N$')
plt.title('Empirical Distribution, '+'N='+str(N)+', $\\theta=$'+str(theta));
```

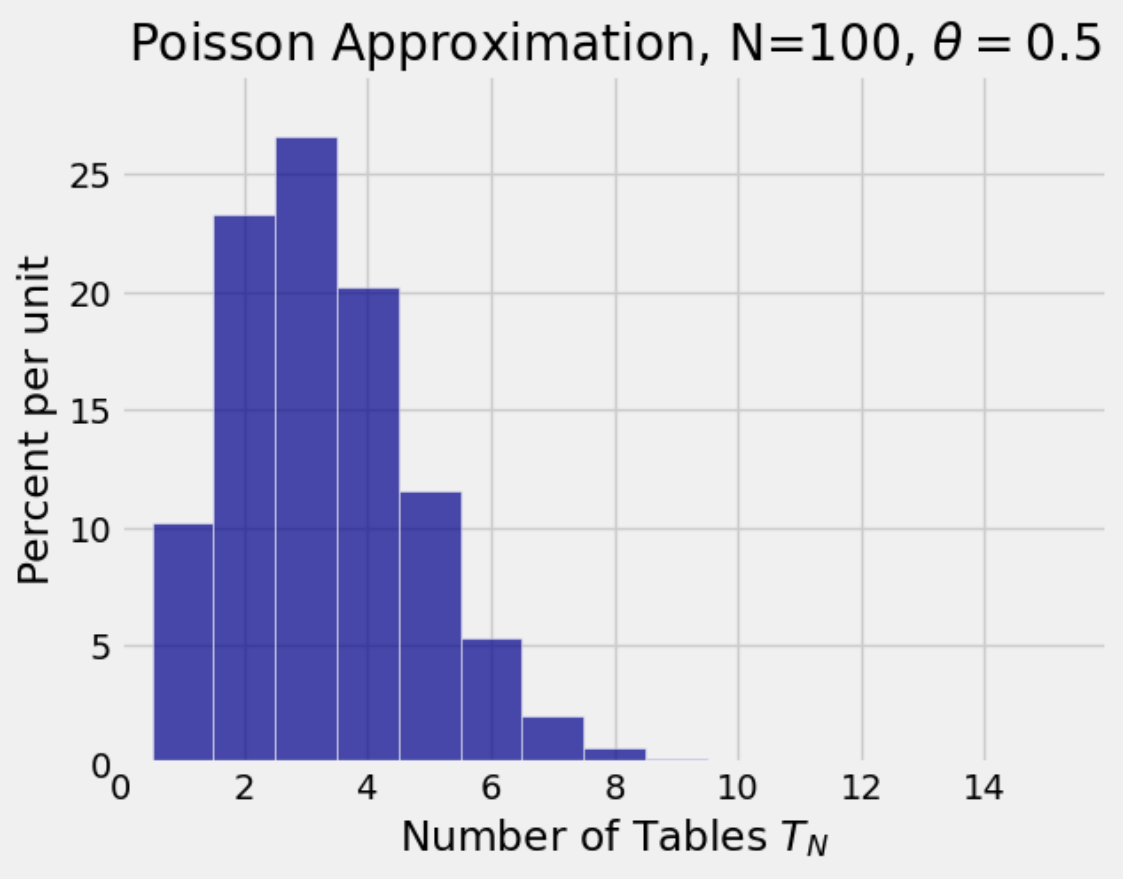


The empirical histogram above was based on 5000 repetitions. So it should be pretty similar to the exact Poisson-binomial histogram.

Now draw the histogram of the approximate distribution of T_N based on the approximation in **5b**. Does it look similar to the one above?

When you write your code, it's important to keep in mind that the approximation in **5b** is for R_N, not T_N. Also note that you can save some calculation by an appropriate use of your function `ev_num_clusters` from **4b**.

[118]:
```
N = 100
theta = 0.5
k = np.arange(1, 16)
probs = stats.poisson.pmf(k - 1, ev_num_clusters(N, theta) - 1)
approx_dist = Table().values(k).probabilities(probs)

Plot(approx_dist)
plt.xlim(0, 16)
plt.xticks(np.arange(0, 16, 2))
plt.xlabel('Number of Tables $T_N$')
plt.title('Poisson Approximation, '+'N='+str(N)+', $\\theta=$'+str(theta));
```



4.4.4 5d) Applying the approximation: Example 1

In the case $N = 100$ and $\theta = 0.5$, write an expression that evaluates to the approximate value of $P(T_N \geq 4)$ based on **5b**.

Complete the cell with just one expression, not multiple lines.

[123]:
```
N = 100
theta = 0.5

# Approximate value of P(T_N >= 4)
# Look at P(R_N >= 3)
1 - stats.poisson.cdf(2, ev_num_clusters(N, theta) - 1)
```

[123]:
```
0.39980478464497948
```

Now compare this approximation with the corresponding empirical approximation based on the 5000 simulated values `sim_100_pt5` in **3c**.

Complete the cell with just one expression, not multiple lines.

[124]:
```
# Empirical approximation to P(T_N >= 4) in the case N = 100, theta = 0.5
np.mean(sim_100_pt5 >= 4)
```

[124]:
```
0.4012
```

4.4.5 5e) Applying the approximation: Example 2

The calculations above should indicate that the Poisson approximation to the distribution of T_N is pretty good. Now try it in a case where we don't have an empirical approximation for comparison.

In the cell below, find $E(T_N)$ in the case $N = 1000$ and $\theta = 0.7$.

[125]:
```
N = 1000
theta = 0.7

ev_num_clusters(N, theta)
```

[125]:
```
5.6895851980329075
```

With the same parameters $N = 1000$ and $\theta = 0.7$, approximate $P(T_N < 10)$.

Complete the cell with one expression, not multiple lines.

[126]:
```
# Approximate value of P(T_N < 10) in the case N = 1000, theta = 0.7
# Look at P(R_N < 9)
stats.poisson.cdf(8, ev_num_clusters(N, theta) - 1)
```

[126]:
```
0.95030145168508862
```

5 Part B of the lab ends here. It is due by 12 PM noon on Tuesday, February 18th.

5.1 Conclusion

Congratulations! You have completed Lab 3. What you have learned: - The assumptions of a clustering model in which individuals aren't labeled by type ahead of time and the number of clusters is unknown - How to simulate data under this model - The distribution of the total number of clusters, with empirical as well as analytical approximations - The exact value of the mean of the number of clusters, and the rate at which the mean grows

5.2 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.

5.2.1 Written Portion

- Scan all the pages into a PDF. You can use any scanner or a phone using applications such as CamScanner. 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 in CamScanner 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.
- If you used LaTeX to do the written portions, you do not need to do any scanning; you can just download the whole notebook as a PDF via LaTeX.

5.2.2 Code Portion

- Save your notebook using `File > Save Notebook`.
- Generate a PDF file using `File > Save and Export Notebook As > PDF`. This might take a few seconds and will automatically download a PDF version of this notebook.
  - If you have issues, please post a follow-up on the general Lab 3B Ed thread.

5.2.3 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 Lab 3B on Gradescope.
- **Make sure to assign each page of your pdf to the correct question.**
- **It is your responsibility to verify that all of your work shows up in your final PDF submission.**

If you are having difficulties scanning, uploading, or submitting your work, please read the Ed Thread on this topic and post a follow-up on the general Lab 3B Ed thread.

5.3 We will not grade assignments that do not have pages selected for each question.

[ ]:

4a. sum of successes → indicator

$T_N \sim$ # of tables

$$I = \begin{cases} 1, & \text{person starts new table (success)} \\ 0, & \text{existing table (fail)} \end{cases}$$

$$E(T_N) = E(I_1) + E(I_2) + \dots + E(I_N)$$

$$= \sum_{n=1}^{N} \frac{\Theta}{\Theta + (n-1)}$$

summation of successes ONLY

$n = 1$, ALWAYS has to start at table 1

$\therefore p = 1$

5a. $T_N = 1 + R_N$

$\therefore R_N = T_N - 1$

$$E[T_N] = \sum_{n=1}^{N} \frac{\theta}{\theta + n - 1}$$

- returns list of probabilities of success of starting new table

$$E[R_N] = E[T_N] - 1 = \left[\sum_{n=0}^{N} \frac{\theta}{\theta + (n-1)} \right] - 1$$

n-0, cause 2nd person doesn't HAVE to start new table

$$E[R_N] = \frac{\theta}{\theta + (1-1)} + \frac{\theta}{\theta + (2-1)} + \frac{\theta}{\theta + (3-1)} + \cdots + \frac{\theta}{\theta + (N-1)} - 1$$

$$= \frac{\theta}{\theta + 1} + \frac{\theta}{\theta + 2} + \cdots + \frac{\theta}{\theta + (N-1)}$$

$$R_N \sim \text{Poisson-binomial}\left(\frac{\theta}{\theta+1}, \frac{\theta}{\theta+2}, \cdots, \frac{\theta}{\theta+(N-1)} \right)$$

$\therefore$ parameters for R_N comes from the expected values of R_N, which is the sum of the individual probabilities of each person starting from person 2 to N will create a new table. Since each person is a Bernoulli trial, the probability depends on n, hence why the parameters are a list of different probabilities.

5b. - parameter keeps changing

$$T_N = 1 + K_N$$

$$\therefore K_N = T_N - 1$$

$$E[K_N] = E[T_N] - 1 = \left[\sum_{n=0}^{N} \frac{\theta}{\theta + (n-1)} \right] - 1$$

$$E[K_N] = \frac{\theta}{\theta + (1-1)} + \frac{\theta}{\theta + (2-1)} + \frac{\theta}{\theta + (3-1)} + \cdots + \frac{\theta}{\theta + (N-1)} - 1$$

$$= \frac{\theta}{\theta + 1} + \frac{\theta}{\theta + 2} + \cdots + \frac{\theta}{\theta + (N-1)}$$

$$\therefore K_N \sim \text{Poisson}\left(\frac{\theta}{\theta + 1} + \frac{\theta}{\theta + 2} + \cdots + \frac{\theta}{\theta + (N-1)} \right)$$

$$\sim \text{Poisson}\left(\sum_{i=2}^{N} \frac{\theta}{\theta + (n-1)} \right)$$