Lab_07

October 26, 2019

Probability for Data Science

UC Berkeley, Fall 2019

Ani Adhikari and Jim Pitman

[32]:
```
# SETUP
from datascience import *
from prob140 import *
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import pylab
from scipy import stats
import ipywidgets as widgets
from ipywidgets import interact
from IPython.display import display
from matplotlib.ticker import FormatStrFormatter
```

[33]:
```
def search(x_limits, cdf, u):
    """
    Runs a binary search to find the inverse cdf.
    """
    # Handle possible asymptotes.
    if cdf(x_limits[0]) > u:
        return x_limits[0]
    if cdf(x_limits[1]) < u:
        return x_limits[1]

    mid = (x_limits[0] + x_limits[1])/2
    diff = u - cdf(mid)
    if np.abs(diff) < 0.01:
        return mid
    if diff < 0:
        return search((x_limits[0], mid), cdf, u)
    return search((mid, x_limits[1]), cdf, u)
```

```
def plot_axes(cdf_table):
    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="k", lw=3)
    plt.xlim(-0.02, 1)
    plt.scatter([0]*(len(cum) + 1),
                [0] + cum, s=55, color="k")

def plot_discrete_cdf(cdf_table, u=None):
    """
    Plots the cdf of a discrete distribution.

    Parameters
    ----------
    cdf_table : Table
        Table of cdf values.
    u : float
        Value from (0, 1) to plot inverse cdf of.
    """
    values = cdf_table.column(0)
    values = np.append(values[0] - 2, values)

    cum = cdf_table.column(cdf_table.num_columns - 1)
    cum = np.append(0, np.cumsum(cum))

    for i in range(len(values) - 1):
        plt.plot([values[i], values[i+1]], [cum[i], cum[i]],
                 color="darkblue")
        plt.plot([values[i+1], values[i+1]], [cum[i], cum[i+1]],
                 ls="--", color="darkblue" )
    plt.scatter(values, cum, s=50, color="darkblue")

    plt.plot([values[-1], values[-1] + 2], [1,1],
             color="darkblue")

    plt.xlim(values[0], values[-1] + 2)
    plt.ylim(-0.1, 1.1)
    plt.xlabel('$x$')
    plt.ylabel('CDF at $x$')
    plt.title('Graph of CDF');

    if u != None:
        for i in range(len(values)):
```

```
            if u <= cum[i]:
                index = values[i]
                break
        height = u

        plt.plot([values[0], (index+values[0])/2], [height, height],
                 marker='>', color='red', lw=1)
        plt.plot([(index+values[0])/2, index], [height, height],
                 color='red', lw=1)
        plt.plot([index, index], [height, height/2], marker="v",
                 color="red", lw=1)
        plt.plot([index, index], [0, height/2], color="red", lw=1)

def plot_continuous_cdf(x_limits, cdf, u=None):
    """
    Plots the cdf of a continuous distribution.
    """
    x = np.linspace(*x_limits, 100)
    cdf_values = list(map(cdf, x))
    plt.plot(x, cdf_values, color="darkblue")
    plt.xlabel('$x$')
    plt.ylabel('CDF at $x$')
    plt.title('Graph of CDF');

    if not u is None:
        index = search(x_limits, cdf, u)
        height = u

        plt.plot([x_limits[0], (index+x_limits[0])/2],
                 [height, height], marker='>', color='red', lw=1)
        plt.plot([(index+x_limits[0])/2, index],
                 [height, height], color='red', lw=1)
        plt.plot([index, index], [height, height/2],
                 marker="v", color="red", lw=1)
        plt.plot([index, index], [0, height/2], color="red", lw=1)


    plt.xlim(*x_limits)

def unit_interval_to_discrete(cdf_table):
    uniform_slider = widgets.FloatSlider(
        value=0.5,min=0,max=1,step=0.02, description='u')
    @interact(u = uniform_slider)
    def plot(u):
        plot_discrete_cdf(cdf_table, u)
```

```
def unit_interval_to_continuous(x_limits, cdf):
    uniform_slider2 = widgets.FloatSlider(
        value=0.5, min=0,max=1,step=0.02, description='u')

    @interact(u = uniform_slider2)
    def plot(u):
        if (cdf(u) > x_limits[1] or cdf(u) < x_limits[0]):
            plot_continuous_cdf(x_limits, cdf)
        else:
            plot_continuous_cdf(x_limits, cdf, u)

def override_hist(*args, **kwargs):
    """
    This cleans up some unfortunate floating point precision
    bugs in the datascience library
    """
    #kwargs['edgecolor'] = 'w'
    Table.hist2(*args, **kwargs)
    ax = plt.gca()
    ticks = ax.get_xticks()
    if np.any(np.array(ticks) != np.rint(ticks)):
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.2f'))

if not hasattr(Table, 'hist2'):
    Table.hist2 = Table.hist

Table.hist = override_hist
```

[34]:
```
def plot_radial_distances():
    n = 500
    sampled_thetas = np.random.uniform(0, 2 * np.pi, n)
    sampled_radii = np.sqrt(np.random.uniform(0, 1, n))
    x = sampled_radii * np.cos(sampled_thetas)
    y = sampled_radii * np.sin(sampled_thetas)
    theta = np.linspace(0, 2 * np.pi, 100)
    uniform_slider = widgets.IntSlider(
        value=10,
        min=1,
        max=n,
        step=1,
        description='n'
    )
    @interact(i=uniform_slider)
    def plot(i):
        fig = plt.figure(figsize=(10, 5))
        ax1 = fig.add_subplot(1, 2, 1)
        ax1.plot(np.cos(theta), np.sin(theta), color='gold')
```

```
        ax1.scatter(x[:i], y[:i], color='darkblue', s=10)
        ax1.set_aspect('equal')
        ax1.set_title('Simulated Points')
        ax1.set_xticks(np.arange(-1, 1.5, 0.5))
        ax1.set_yticks(np.arange(-1, 1.5, 0.5))
        ax1.set_xlabel('x')
        ax1.set_ylabel('y')
        ax2 = fig.add_subplot(1, 2, 2)
        ax2.set_title('Empirical Histogram of Radius')
        ax2.hist(sampled_radii[:i], bins=np.linspace(0, 1, 25), density=True,␣
↪color='darkblue')
        ax2.set_ylabel('Percent per Unit')
        plt.yticks(ax2.get_yticks(), ax2.get_yticks() * 100);
        ax2.set_xlabel('r')
        plt.subplots_adjust(wspace=0.5)
```

0.1 Lab Resources

- `prob 140` Library Documentation
- Data 8 Python Reference
- Prob 140 Code Reference Sheet
- `scipy.stats` Documentation

1 Lab 7: Simulation and the CDF

Simulation helps us understand properties of random variables. For example, earlier in the term you saw `simulate_path` for simulating Markov Chains; this was helpful for understanding transition behavior and reversibility. The `Table` method `sample` simulates drawing uniformly at random from the rows of a table; in Data 8, you used it understand the bootstrap. Simulation is also important because properties observed in simulations can lead to the development of new results.

In this lab you will simulate random variables with specified distributions.

What you will learn: - How to use `SciPy` for simulation - How to construct and read graphs of cumulative distribution functions (cdfs) - How being able to simulate just one distribution allows us to simulate all others

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.

2 newpage

2.1 Part 1: Simulation in SciPy

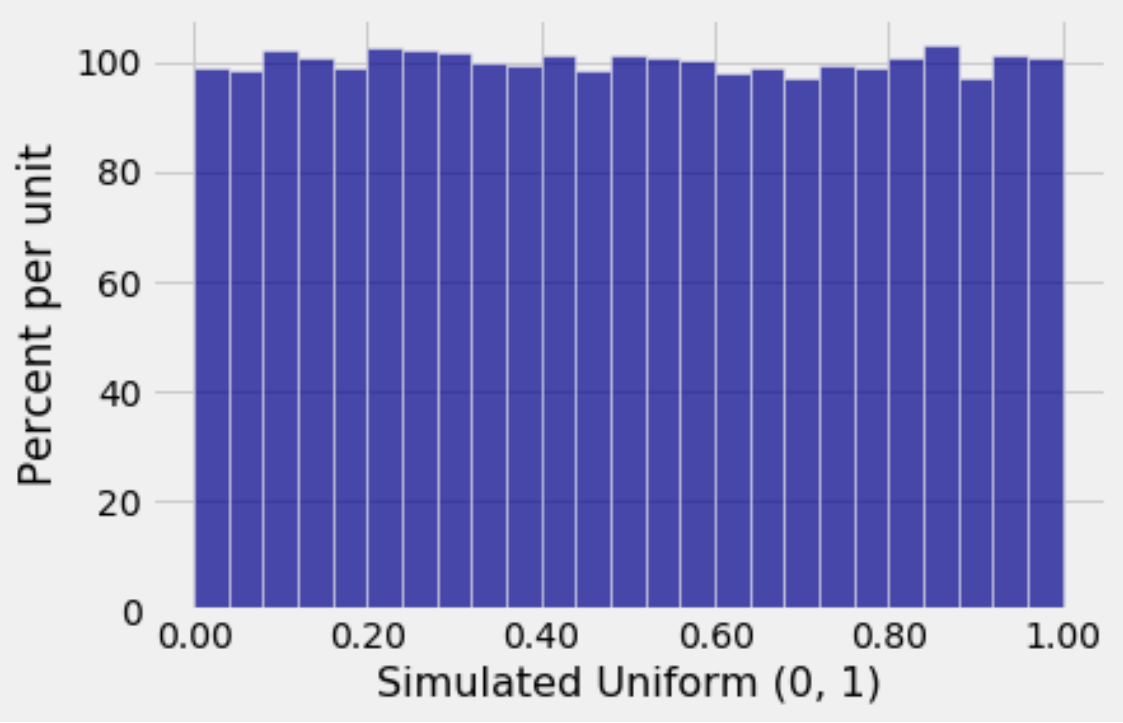
The `stats` module of `SciPy` is familiar to you by now. For any of the well known distributions, you can use `stats` to simulate values of a random variable with that distribution. The general call is `stats.distribution_name.rvs(size = n)` where `rvs` stands for "random variates" and `n` is the number of independent replications you want.

Every statistical system has conventions for how to specify the parameters of a distribution. In this lab we will tell you the specifications for a few distributions. Later you will be able to see a general pattern in the specifications.

2.1.1 1a) Uniform $(0, 1)$

The call is straightforward: `stats.uniform.rvs(0, 1, size=n)`. Complete the cell below to draw the histogram of 100,000 simulated values of a random variable that has the uniform $(0, 1)$ distribution. The `hist` option `bins=25` results in 25 equal bins.

[35]:
```
sim_uniform = stats.uniform.rvs(0, 1, size=100000)
sim_uniform_tbl = Table().with_column(
    'Simulated Uniform (0, 1)', sim_uniform
)
sim_uniform_tbl.hist(bins=25)
```



2.1.2 1b) Reading the Scales of the Histogram

The unit on the horizontal axis is any unit of length; you can think of it as centimeters if you want, but we will just refer to it as the "unit". Fill in the blanks below and provide units where appropriate. Some units have been provided for you.

(i) The width of each bin is _____ units.

(ii) The height of each bar is approximately _____ per _____.

(iii) Histograms represent percents by _____, so the answers to (i) and (ii) imply that the percent of simulated values in each bin is approximately _____%.

(iv) Let the random variable U have the uniform $(0, 1)$ distribution, and let B be any bin of the histogram. The answer to (i) implies that $P(U \in B) =$ _____%.

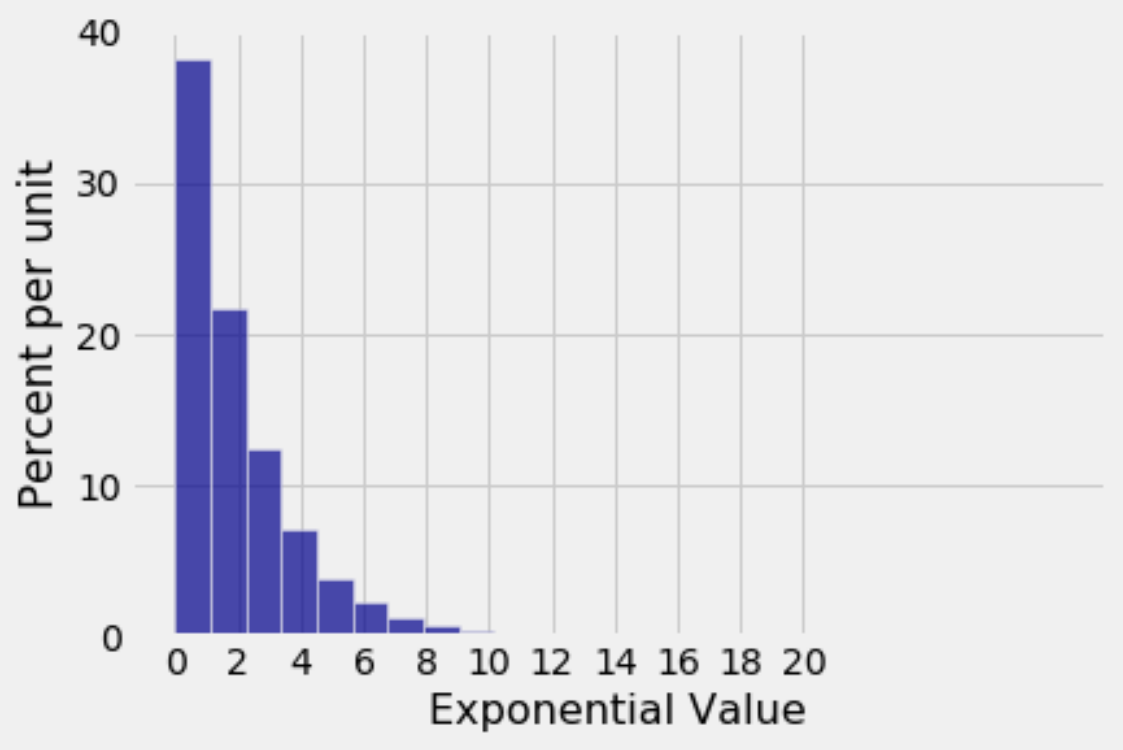
(v) If instead of `bins=25` we had used `bins=20` as the option to `hist`, then the answer to (iv) would have been _____%

(i) 0.04 (ii)100% per unit (iii)width of the bin*heigh of each bar && 0.04*100% = 0.04 = 4% (iv)(1/25)*100% = 0.04 = 4% (v)(1/20)*100% = 0.05 = 5%

2.1.3 1c) Exponential

The exponential distribution has two common parametrizations. One is the rate λ, which is what we use in Prob 140. The other is the mean $\frac{1}{\lambda}$. The mean is called the `scale` parameter in `stats`. Complete the cell below to simulate 100,000 values of a random variable that has the exponential distribution with rate 0.5.

[36]:
```
sim_expon = stats.expon.rvs(scale=1/0.5, size=100000)                    # array␣
 ↪of simulated values
sim_expon_tbl = Table().with_column(
    'Exponential Value',sim_expon
).hist(bins=25)
#sim_uniform_tbl = Table().with_column(
#    'Simulated Uniform (0, 1)', sim_uniform
#)
#sim_uniform_tbl.hist(bins=25)
plt.xticks(np.arange(0, 21, 2));
```



Find the average of the simulated values and check that it is consistent with the rate.

[37]:
```
sample_mean = np.mean(sim_expon)
print(sample_mean)
sample_scale = 1/sample_mean
print(sample_scale)
```

```
1.998562835490264
0.5003595494933196
```

3 newpage

3.1 Part 2. The Idea

How are all these random numbers generated? In the rest of the lab we will develop the method that underlies all the simulations above, by considering examples of increasing complexity.

Our starting point is a distribution on just four values.

Suppose X has the distribution `dist_X` below.

[38]:
```
vals_X = make_array(-2, 1, 4, 7)
probs_X = make_array(0.3, 0.1, 0.2,0.4)
```

```
dist_X = Table().values(vals_X).probabilities(probs_X)
dist_X
```

[38]:
```
Value | Probability
-2    | 0.3
1     | 0.1
4     | 0.2
7     | 0.4
```

Our goal is to simulate one value of X. That is, we want to come up with a process that returns one of the four possible values of X with the right probabilities.

3.1.1 2a) A Vertical Unit Interval

The graphic below shows the probabilities in `dist_X` stacked vertically as in Lab 4. From the bottom to the top, therefore, you have the unit interval.

Now imagine throwing a dart at the unit interval. That is, let U be a random variable that has the uniform distribution on $(0, 1)$, and suppose you mark the value of U on the unit interval shown in the graph.

[39]:
```
plot_axes(dist_X)
```

Find the following probabilities and see how they are related to the distribution of X.

(i) $P(U \leq 0.3)$

(ii) $P(0.3 < U \leq 0.4)$

(iii) $P(0.4 < U \leq 0.6)$

(iv) $P(0.6 < U \leq 1)$

i. 0.3

ii. 0.1

iii. 0.2

iv. 0.4

3.1.2 2b) Idea for Simulating X

Starting with a uniform $(0, 1)$ random variable U, propose a method of generating a value of X.

Your method should take U as its input and return one of the four possible values as output, in such a way that for each $i = -2, 1, 4, 7$, the chance of returning the value i is $P(X = i)$.

Just describe your method in words. No formula or code is needed.

There is two way that I can generate a value of X. First method is simply to use the probability function call from data8, where my input array value is [-2,1,4,7] and my probability array is [0.3,0.1,0.2,0.4] with size n. The call is going to reaturn a value X according to the given input array and probability array. Second method is that I can use a random number generator that generate random value between 0 to 1. Since the build in random number genertor generate is uniformaly distribute in the given interval. If the generator return (0,0.3], value X is returned -2; if the generator return(0.3,0.4], X returned 1;if the generator return(0.4,0.6], X returned 4 and if the generator return(0.6,1], X returned T.

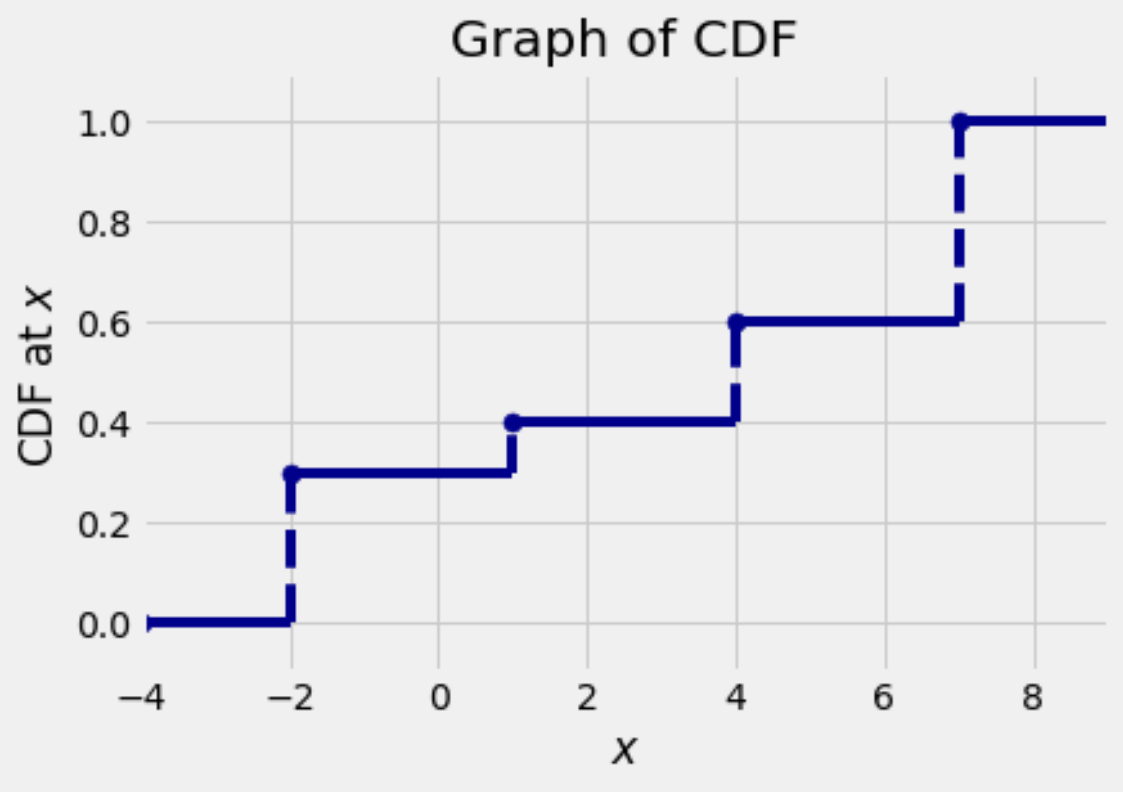
4 newpage

4.1 Part 3. Visualizing the Idea

The method `plot_discrete_cdf` takes a distribution as its argument and plots a graph of the cdf.

Run the cell below to get a graph of the cdf of the random variable X in Part 1.

[40]:
```
plot_discrete_cdf(dist_X)
```



4.1.1 3a) Reading the Graph

Let F_X be the cdf of X. What is the definition of $F_X(2.57)$? Does the graph show the correct value for $F_X(2.57)$?

$F_x(2.57)$ represent the area under the density function of X from negative infinite to 2.57. Since F_x is the cdf of X, $F_x(2.57)$ represent the cdf of X=2.57, which is P(X<=2.57). And $P(X <= 2.57) = P(X = -2) + P(X = 1) = 0.3 + 0.1 = 0.4$ Thus the graph does show the correct value for $F_x(2.57)$

4.1.2 3b) Jumps

At what points x does the graph have a jump? For each point x at which there is a jump, find the size of the jump in terms of the distribution of X.

The graph have a jump when x=-2,1,4,7 which jumps at the possible value of X. And the size of the jump at x is the same as the probaility of X at that value. So the size of the jumps is the same as the distribution of X.

4.1.3 3c) From the Unit Interval to Values of X

The function `unit_interval_to_discrete` takes a distribution as its argument and displays an animation of a method that generates one value of a random variable that has the given distribution. The method starts with a number on the unit interval.

Run the cell below. Move the slider around and see how the returned value changes **depending on the starting value** in the unit interval. How is the method that is being displayed related to the one you proposed in Part 1?

[41]:
```
unit_interval_to_discrete(dist_X)
```

```
interactive(children=(FloatSlider(value=0.5, description='u', max=1.0, step=0.02), Output()),
```

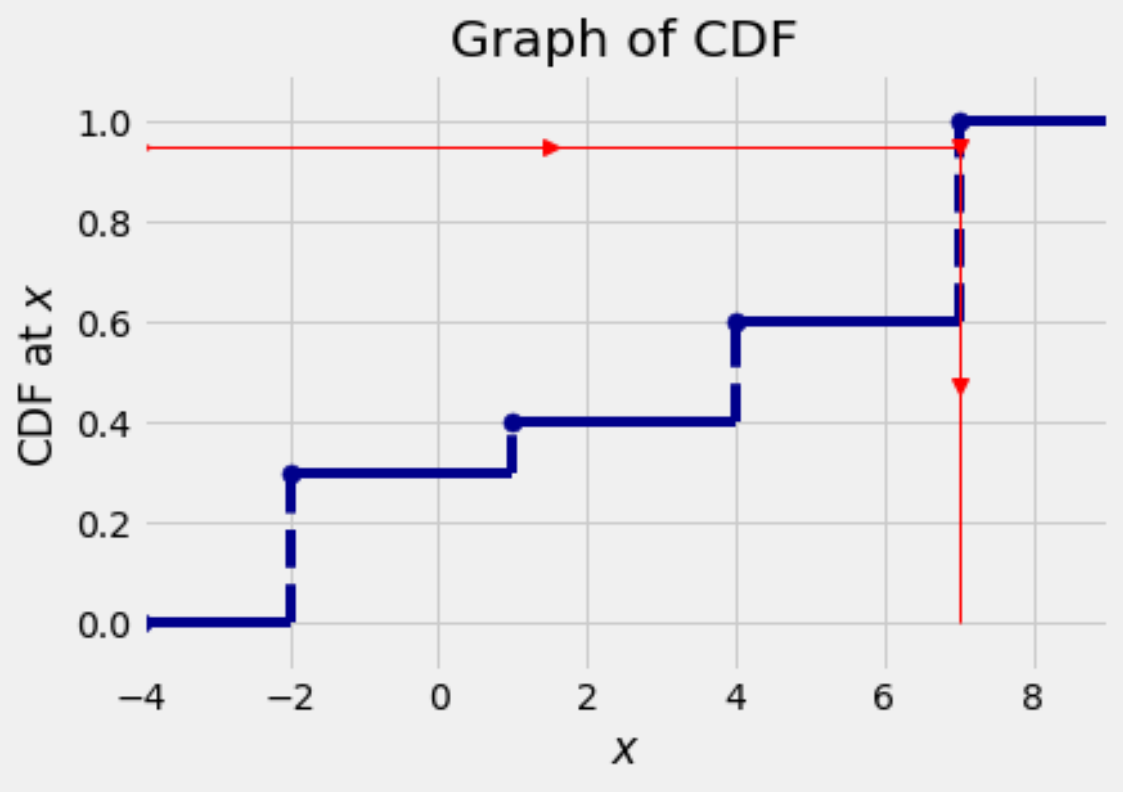
It matches the method in part one as the value return is the same X value according to the cdf of X.

4.1.4 3d) A Random Starting Point

The method `plot_discrete_cdf` that you used earlier also takes a second argument which is a number between 0 and 1.

Complete the cell below so that the second argument is picked uniformly at random from (0, 1). Run the cell a few times. How is it related to the method you proposed in Part 1 for generating a value of X?

[42]:
```
plot_discrete_cdf(dist_X, np.random.uniform(0,1))
```



This is realted to the method I proposed in part 1 for generating a value of X because the probaility of P(X<=x) is the same as the proability I use to generate the value of X. And the graph of cdf looks like it as well. Also the number that return is the cdf=0.5, which is the expected value of X. So this graphs is related to the method I proposed in part 1 for generating a value of X.

5 newpage

5.1 Part 4. Extension to Continuous Distributions

Now suppose you want to generate a random variable that has a specified continuous distribution. Let's start with the exponential (λ) distribution.

5.1.1 4a) [ON PAPER] Exponential CDF

Let T have the exponential distribution with rate λ. Let F_T be the cdf of T. Write the formula for F_T. Remember that the cdf is a function on the entire number line $(-\infty, \infty)$; make sure you specify the function on the whole line.

5.1.2 4b) Plotting the Exponential CDF

As a numerical example, let T be a random variable that has the exponential distribution with rate $\lambda = 0.5$, or equivalently, expectation 2. Define a function `expon_mean2_cdf` that takes a numerical argument x and returns $F_T(x)$. Use `np.exp(y)` for e^y.

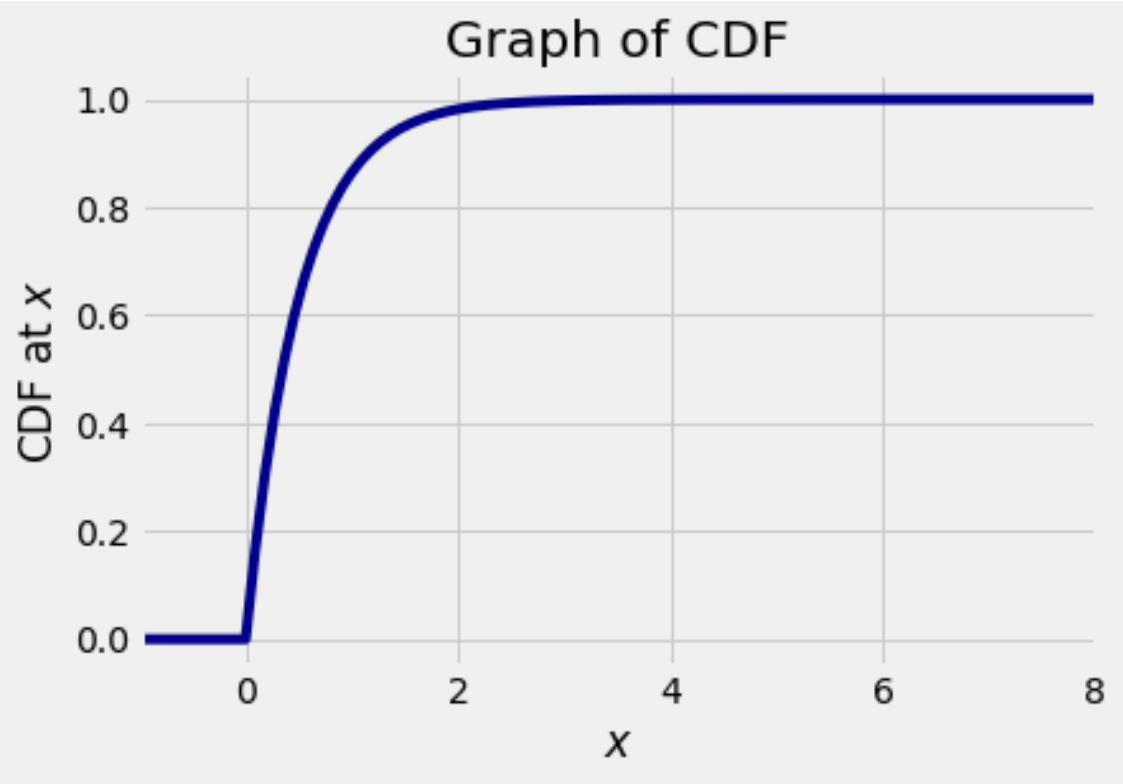
Make sure that for **all** numerical values of x your function returns the value you specified in **4a**.

[43]:
```
# don't use "lambda" as that means something else in Python
lamb = 0.5

def expon_mean2_cdf(x):
    if x<=0:
        return 0
    po=1/lamb
    area = 1-np.exp(-1*po*x)
    return area
```

The function `plot_continuous_cdf` plots the cdf of a continuous variable. The first two arguments: - an interval (a, b) over which to draw the cdf - the name of a cdf function that takes a numerical input and returns the value of the cdf at that input

Run the cell below to check that your function `expon_mean2_cdf` looks good.

[44]:
```
plot_continuous_cdf((-1, 8), expon_mean2_cdf)
```



5.1.3 4c) Idea for Simulating an Exponential Random Variable

Suppose you are given one uniform $(0, 1)$ random number and are asked to simulate T. Based on Part 3 of the lab, propose a method for doing this by using the graph above.

You don't have to prove that the method works. We'll do a formal proof in lecture. Just propose the method.

To generate T using the random number, I will first find the accroding F_T value from the expon func on every possible value of x. Then using all the T value to seperate the interval in between 0,1. And then I will matching the random number to the actual T base on those divided interval.

5.1.4 4d) Visualizing the Idea

The animation in the cell below is analogous to the one in Part 3. Its arguments are: - a plotting interval - the name of a continuous cdf function

The output demonstrates a method for picking a number on the positive real line starting with a value on the unit interval that forms the vertical axis.

Run the cell and move the slider around to see how the returned value changes depending on the starting value on the vertical axis.

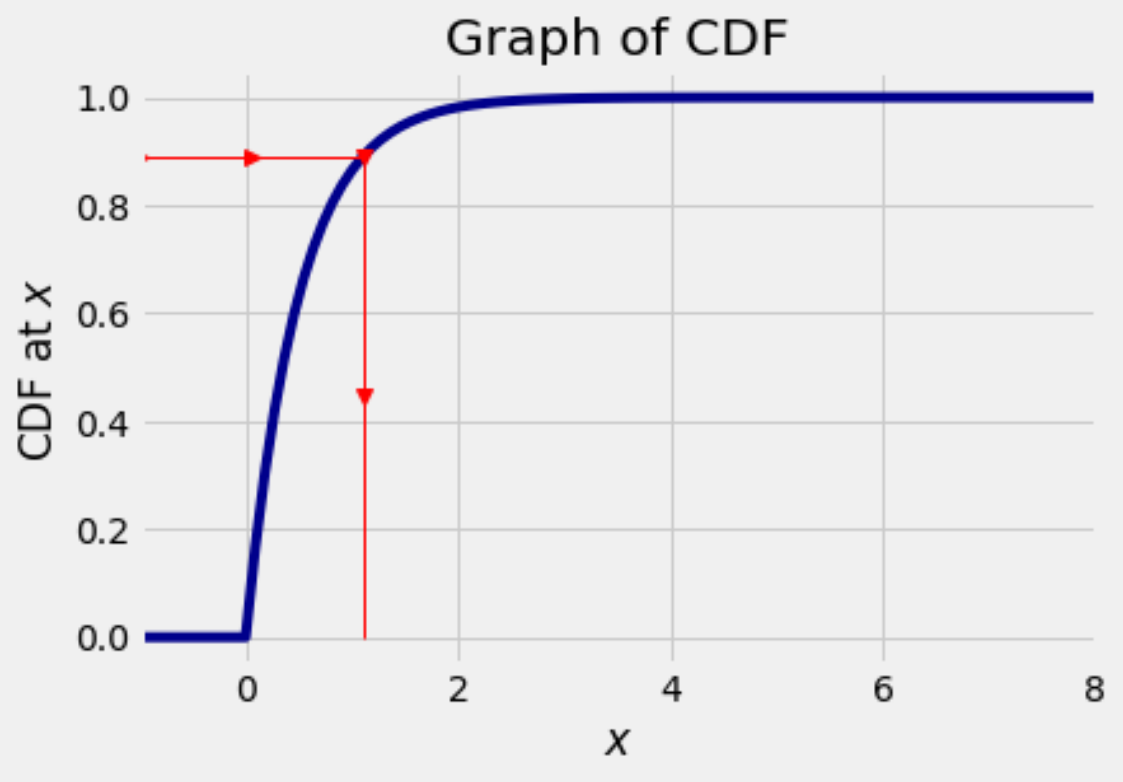
[45]:
```
unit_interval_to_continuous((-1, 8), expon_mean2_cdf)
```

```
interactive(children=(FloatSlider(value=0.5, description='u', max=1.0, step=0.02), Output()),
```

The method `plot_continuous_cdf` takes an optional third argument that is a number between 0 and 1.

Complete the cell below so that the third argument is a random number picked uniformly from (0, 1). Run the cell a few times. How is the output related to the method you proposed in **4c** for generating a value of T?

[46]:
```
plot_continuous_cdf((-1, 8), expon_mean2_cdf,np.random.uniform(0,1))
```



The output is related to the method I proposed in 4c for generating a value of T. The output T is according to the x value that is generate from the expo func.

5.1.5 4e) [ON PAPER] The General Method

Let F be any continuous increasing cdf. That is, suppose F has no jumps and no flat bits.

Suppose you are trying to create a random variable X that has cdf F, and suppose that all you have is F and a number picked uniformly on $(0, 1)$.

(i) **Fill in the blank:** Let U be a uniform $(0, 1)$ random variable. To construct a random variable $X = g(U)$ so that X has the cdf F, take $g =$ _____.

(ii) **Fill in the blank:** Let U be a uniform $(0, 1)$ random variable. For the function g defined by

$$g(u) \;=\; \_\_\_\_\_, \quad 0 < u < 1$$

the random variable $X = g(U)$ has the exponential (λ) distribution.

[Note: If F is a discrete cdf then the function g is complicated to write out formally, so we're not asking you to do that. The practical description of the method of simulation is in Parts 1 and 2.]

6 newpage

6.1 Part 5. Empirical Verification that the Method Works

6.1.1 a) The Initial Values

Create a table that is called `sim` for simulation and consists of one column called `Uniform` that contains the values of 100,000 i.i.d. uniform $(0, 1)$ random variables.

[47]:
```
N = 100000
u = stats.uniform.rvs(0, 1, size=100000)
sim = Table().with_column("Uniform", u)
sim
```

[47]:
```
Uniform
0.282784
0.985698
0.428643
0.853638
0.494319
0.0788824
0.228085
0.552506
0.350265
0.708749
… (99990 rows omitted)
```

6.1.2 b) Tranformation to Exponential

Use **4e** and the values in the column `Uniform` to create an array of values that have the exponential distribution with rate 0.5. This is what is going on "under the hood" in `stats.expon.rvs`.

Do not simulate new random numbers, as you will lose the connection with the values in `Uniform`. Use `np.log(y)` for $\log(y)$.

Augment `sim` with a column containing the new array.

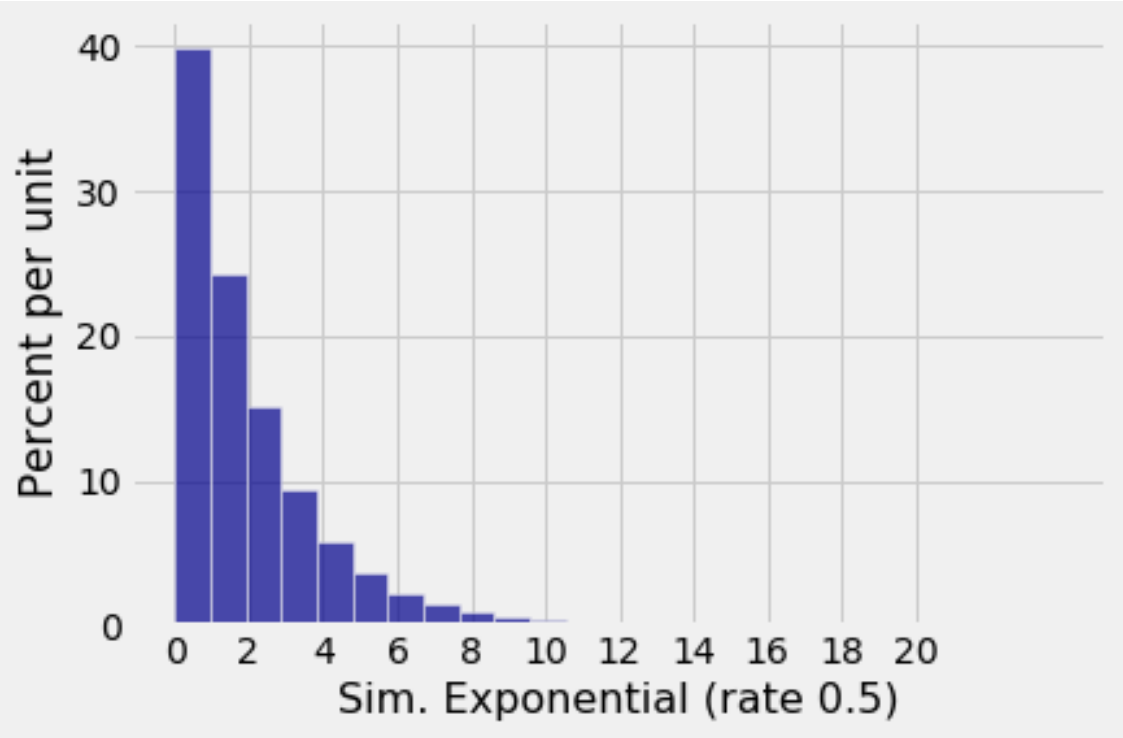
[48]:
```
def uniform_to_exponential_mean2(u):
    return np.log(1-u)/(-0.5)

exponential_mean2 = sim.apply(uniform_to_exponential_mean2,'Uniform')
sim = sim.with_column('Sim. Exponential (rate 0.5)', exponential_mean2)
sim
```

[48]:
```
Uniform   | Sim. Exponential (rate 0.5)
0.282784  | 0.664756
0.985698  | 8.49476
```

```
0.428643   | 1.11948
0.853638   | 3.84334
0.494319   | 1.3637
0.0788824  | 0.164335
0.228085   | 0.517762
0.552506   | 1.60819
0.350265   | 0.862382
0.708749   | 2.46714
… (99990 rows omitted)
```

Run the cell below and compare with the histogram in 1(d) to confirm that your calculation is correct.

[49]:
```
sim.hist('Sim. Exponential (rate 0.5)', bins=25)
plt.xticks(np.arange(0, 21, 2));
```



At this point, go back and look through Part 5. Notice that the only time you generated random numbers was when you simulated 100,000 uniform (0, 1) values. All the other variables were deterministic transformations of the uniform variable.

6.2 Conclusion

You have learned that: - To simulate a random variable with a desired distribution, what you need is a uniform random number generator and the cdf of the desired distribution. You can then use

the method of this lab to simulate the value. - The only random numbers a statistical system needs are uniform on (0, 1). Random numbers from all other distributions follow by the method you have developed in this lab. - Discrete cdfs consist of jumps and flat parts, at places that you have identified.

Since uniform $(0, 1)$ random numbers are central to all simulations, their quality is very important for the accuracy and reliability of simulations. Testing and assessing uniform random number generators is serious business, because random number generators don't really produce random numbers. They follow deterministic processes that produce results that have properties that resemble those of random numbers. That is why they are called Pseudo Random Number Generators or PRNGs. Python uses the Mersenne Twister, one of the most tested and reliable PRNGs. SciPy uses the Mersenne Twister for RandomState and draws from a large number of discrete and continuous distributions. Take a look at the list on the RandomState page and see how many you can recognize.

6.3 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.

6.3.1 Written Portion

- Scan all the pages into a PDF. You can use any scanner or a phone using an application. 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.

6.3.2 Code Portion

- Save your notebook using File > Save and Checkpoint.
- Generate a PDF file using File > Download as > PDF via LaTeX. This might take a few seconds and will automatically download a PDF version of this notebook.
  - If you have issues, please make a follow-up post on the general Lab 7 Piazza thread.

6.3.3 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 7 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.**

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

[ ]:

lab 07

4.

a.

$\Rightarrow$ exponential dist = $f(t) = \lambda * e^{-\lambda t}$

$\Rightarrow$ cdf of + = $\int_{-\infty}^{\infty} \lambda * e^{-\lambda t}$

$= 1 - e^{-\lambda t}$

$\Rightarrow$ thus, $F_T(t) = 1 - e^{-\lambda t}$ & $t \geq 0$.

e. $\rightarrow T \rightarrow$ continuous increasing cdf (no jump / flat).

$\rightarrow$ random variable X with cdf T

$\rightarrow$ random k from uniform dist (0, 1).

(i). $\rightarrow$ U be uniform (0,1) random variable.

$\rightarrow$ construct $X = g(u)$ & X has cdf T.

$\Rightarrow g = T^{-1}$

(ii)

$\Rightarrow$ cdf for exponential func

$F_T = 1 - e^{-\lambda t}$

$y = 1 - e^{-\lambda t}$

$$\Rightarrow \quad t = 1 - e^{-\lambda y}$$

$$e^{-\lambda y} = 1 - t$$

$$-\lambda y = \ln(1-t)$$

$$y = \frac{\ln(1-t)}{-\lambda}$$

$$\Rightarrow \text{thus, } g(u) = \frac{\ln(1-u)}{-\lambda}, \quad \text{for} \quad 0 < u < 1$$