3. Inverse transform method

Overview

In this lecture, we discuss how to use inverse transform method to generate samples from a discrete distribution.

Learning objective:

- Understand array index, conditional statement, function, and list comprehension
- Understand the basic algorithm of inverse transform method
- Able to generate samples from a discrete distribution given the PMF/CDF of the distribution

Motivating example

(1) A assume that we would like to generate samples that can be equal to 0 or 1 with 0.5/0.5 probability. How can we do that?

We can flip a fair 2-sided coin many times, every single time a "head" appears, we record 1, while every single time a "tail" appears, we record 0.

(2) How about we like to generate samples that can be equal to 1/2/3/4/5/6 with 1/6 probability for each outcome?

We can roll a fair dice many times. For each side, we give a corresponding number.

(3) How about 1/2/3 with 1/3 probability?
Hmmm... If we can have a "device" that helps us get 3 outcomes with the same probability.

Computer device based on random numbers

Apparently, we do not want to design a physical device to do the sampling every time. We will use random numbers as the device to help us do that.

(1) In the first example, when we draw a sample u; from a uniform distribution between 0 and 1, there is 0.5 probability that a random number is below 0.5, while 0.5 probability that a random number is above 0.5.

Thus our "device" will be

range of u	**sample value**
$0 \leq u < 0.5$	0
$0.5 \leq u < 1$	1

(2) In the second example, when we draw a sample u from a uniform distribution between 0 and 1, there is 1/6 probability that u is below 1/6, 1/6 probability that u is between 1/6 and 2/6, 1/6 probability that u is between 2/6 and 3/6, and so on...

range of u	**sample value**
$0 \leq u < 1/6$	1
$1/6 \leq u < 2/6$	2
$2/6 \leq u < 3/6$	3
$3/6 \leq u < 4/6$	4
$4/6 \leq u < 5/6$	5
$5/6 \leq u < 6/6$	6

Converting to random number to an outcome

A random number is equally like to drop on any point of the between 0 and 1. Depends on which interval it falls into, we will then convert this random number to the corresponding outcome in that interval.

More specifically, we can do the following:

- Step 1: Generate a random number RN
- Step 2: First comparison
    - If RN is higher than or equal to the CDF of the first possible outcome, continue
    - Else break out from comparison
- Step 3: Second comparison
    - If RN is higher than or equal to the CDF of the second possible outcome, continue
    - Else break out from the comparison
- ...
- Loop this compare with each CDF value until out from the comparison (RN is less than a CDF)

We call this method "inverse transform method", because we are trying to put the random number between to CDF values to back out the outcome.

Python Prep: Array indexing

To implement the algorithm, we need to keep on retrieving CDF values from the smallest to the largest. If we have an array, we then need to use index to retrieve a specific CDF value.

Let's assume that the CDF array is Array([0.1,0.4,1.0])

Every single element inside of this array has a unique index. The index is 0 for the first element, 1 for the second element, 2 for the third element, and so on. For example, 1.0 in the above array has an index equals 2.

Thus, once we declare the indices of specific elements we would like to pick, Python will be able to return the values of those elements by locating those elements based on the indices.

Assume that A is an array or a list.

- `A[index]`

  > Get an element based on the index value

- `A[-1]`

  > Get the Last element. As an extension, -2 will indicate the second to last element, and so on

We can get several elements at the same time by passing an array or a list that contains several indices.

- `A[array/list]`

Python Prep: Conditional Statement

When we are comparing RN with the CDF, we need to decide whether to stop the comparison or not based on the comparison result. We can use the conditional statements to achieve this. The syntax for conditional statements are is follows:

```
if  condition:
    operations
elif condition:
    operations
elif condition:
    operations
else：
    operations
```

The program will do the following

- Check the first condition
  - If the first condition is satisfied, it will run the operations defined under this `if` statement and **jump out** of the whole block.
  - Else Check the second condition
    - If the second condition is satisfied, it will run the operations defined under this `if` statement and **jump out** of the whole block.
    - ...
      - If none of the conditions above are satisfied, the operations under `else` will run.

In the following example, we check which segment Va is in.

▸ Run PYTHON

```
1 Va=0.1
2
3 if Va<1/3:
4     print("Va is in (-infinity, 1/3)")
5 elif Va<=2/3:
6     print("Va is in [1/3, 2/3 ]")
7 else:
8     print("Va is in (2/3,infinity) ")
```

- If no operations need to be done under a condition, we can write `pass` . Equivalently, we can simply not including that branch.

Run PYTHON

```python
Va=0.9

if Va<1/3:
    print("Va is in (-infinity, 1/3)")
elif Va<=2/3:
    print("Va is in [1/3, 2/3 ]")
else:
    pass
```

Run PYTHON

```python
Va=0.9

if Va<1/3:
    print("Va is in (-infinity, 1/3)")
elif Va<=2/3:
    print("Va is in [1/3, 2/3 ]")

```

Python Prep: Function and list comprehension

Writing a function

Defining a function would be greatly helpful when performing repetitive tasks.

```
def function_name (argument1, argument2, ....):
    operations
```

In the code above, the arguments are variables that we pass into the function and use for the operations inside the function. The operations give the tasks we want to finish with the help of the arguments we defined above.

If we want the function to return a value, we can use `return` statement

When calling the function, we simply need to define the argument value.

For example, in the above task, we can define a function that can help us compute the area of a circle.

▸ Run PYTHON

```
1 import numpy as np
2 #Define an empty array. We will later store the results from each iterat
3 def area(radius=1):
4     if radius<=0:
5         print("negative radius?")
6     else:
7         return np.pi*radius**2
8
9 #Call the function to give the results
10 #using the default value of radius
11 print(area())
12 #using a wrong radius
13 print(area(-1))
14 #print out the area when radius is equal to 2.
```

List comprehension

We can use list comprehension to run a function multiple times

`[function_name(i) for i in array1  ]`

This command will pretty much do the following:

```
for i in array1:
```

```
    return=function_name(i)`
    save the return value in a list
```

However, list comprehension is more compact and faster in execution.

For example, the following command will compute the area of circles with radius=1, 2, 3, ..., 10.

▸ Run PYTHON

```
1 import numpy as np
2 #Define an empty array. We will later store the results from each iterat
3 def area(radius=1):
4     if radius<=0:
5         print("negative radius?")
6     else:
7         return np.pi*radius**2
8
9 areas=[area(radius) for radius in range(-2,11)]
10 print(areas)
```

Python Demo

Let's try to draw samples from the following distribution

We will be doing the following:

- Implement the algorithm
  - When implementing the algorithm, we need to jump out of the loop when $u < CDF$. To do so, use `break` command
- Package this algorithm in a function. This function will give us one sample.
- Use list comprehension to run the function 1000 times to generate 1000 samples
- Visualize the theoretical distribution using a bar chart, while the sample distribution using a scatter plot.
  - To visualize the sample distribution, we need to know for each value, how many times it occurs, we can do so using `v1,v2=np.unique(a, return_counts=True)`
  - `v1` is an array that contains all the unique values in array `a`, `v2` is an array that contains how many times each unique value appeared in array `a.`

▸ Run PYTHON

```
1 import numpy as np
2 a=np.array([1,1,1,4,4,5])
3 x,counts=np.unique(a,return_counts=True)
4 print(x,counts)
```

Practice

Generate 1000 random samples from a Binomial distribution with n=6, p=0.2.

Plot the sample distribution and theoretical distribution.