Simulation by Random Choice
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/simulation/simulation.pdf

---

---

1 A Pack of Wolves

The wildlife manager at a park is responsible for the health of 100 wolves, which run wild in the park. A deadly virus is known, that can infect wolves, and later spread to other animals. Since the wolves are wild, the only way the manager can examine them is to use a trap, which, overnight, will catch exactly one wolf at random.

Suppose that each wolf has a unique tattoo, from 0 to 99, and that wolf 50 is the only wolf that has been infected. The manager uses the trap every night, until wolf 50 is caught and can be treated. On average, how many days will this take?

Suppose, instead, that no wolf has been infected, but every wolf must be vaccinated. The manager traps a random wolf every night, and inoculates the unprotected ones. On average, how many days will this process take?

2 Collect A Special Card

We can transfer this problem to the more familiar case of a pack of playing cards. For the first problem, let's suppose we repeatedly pick one card from the deck and then replace it. We keep doing this until we draw the ace of spades. Can we describe this process statistically? The minimum number of draws is obviously 1, and the maximum is ..., well, infinity. So more useful numbers of the mean or average, and the standard deviation, which is roughly a measure of how close to the mean most cases will be.

We can simulate a single case of this process using `np.random.randint()`, but we will prefer to use `numpy.random.choice()`:

```
card = np.random.choice ( 52, 1, replace = True )
```

We assume that the Ace of Spades is card 0. We have to repeat this process indefinitely, until we see the desired card.

```
draws = 0
card = -1
while ( card != 0 ):
  card = np.random.choice ( 52, 1, replace = True )
  draws = draws + 1
```

This only gives us one estimate. To get statistics, we have to repeat this test many times:

```
n = 1000
draws = np.zeros ( n, dtype = int )

for i in range ( 0, n ):
  draws[i] = 0
  card = -1
  while ( card != 0 ):
    card = np.random.choice ( 52, 1, replace = True )
    draws[i] = draws[i] + 1

print ( '  Mean number of draws was', np.mean ( draws ) )
print ( '  Standard deviation was ', np.std ( draws ) )

plt.hist ( draws, bins = 20 )
plt.show ( )
```

3 Collect A Special Card, and No Replacement

Suppose that, while looking for the Ace of Spades, we do not replace the cards we draw. Then you should probably guess that, as the deck gets smaller and smaller, we have at most 52 draws, and probably about half that many on average.

To verify this, we can simply override the default `replace = True` option in the `np.random.choice()` function:

```
    card = np.random.choice ( 52, 1, replace = False )
```

and indeed, the mean waiting time drops dramatically.

In the wolf pack example, this would be like sending each trapped wolf to another park, so that the wild population drops by 1 every day, making the capture of the special wolf inevitable.

4 Collect Every Card

Now suppose instead of just wanting to draw the Ace of Spades, we want to draw every card at least once. In this situation, it will help to keep track of how many times we have drawn each card. So we create a `count()` array to do this. As long as any count is zero, we have to keep drawing another card.

A single search might go like this:

```
count = np.zeros ( 52, dtype = int )
draws = 0
```

```
  while ( np.any ( count == 0 ) ):
    card = np.random.choice ( 52, 1, replace = True )
    count[card] = count[card] + 1
    draws = draws + 1
```

and again, in order to get a mean and variance to describe this process, we have to try a large number of searches.

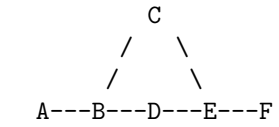
```
n = 1000
draws = np.zeros ( n, dtype = int )

for i in range ( 0, n ):
  count = np.zeros ( 52, dtype = int )
  draws[i] = 0
  while ( np.any ( count == 0 ) ):
    card = np.random.choice ( 52, 1, replace = True )
    count[card] = count[card] + 1
    draws[i] = draws[i] + 1
```

From these experiements with cards, you should be able to go back to the wild wolf problem and work out the answers!

5 A Fly in a Maze

Consider the following very simple maze, which is represented by a graph whose links go both ways (that is, this is not like the directed graphs we saw in the PageRank discussion):



A fly starts in room A, and then, every minute, choose a neighboring room at random. Although the succcessive positions of the fly are random, is there an overall pattern to the number of times each room is visited? From our discussion of the PageRank algorithm, you might suspect that there is something going on.

To model this problem, we might write:

```
n = 21
room = np.zeros ( n, dtype = str )

room[0] = 'A'

for i in range ( 1, n ):
  if ( room[i-1] == 'A' ):
    room[i] = np.random.choice ( [ 'B' ] )
  elif ( room[i-1] == 'B' ):
    room[i] = np.random.choice ( [ 'A', 'C', 'D' ] )
  elif ( room[i-1] == 'C' ):
    room[i] = np.random.choice ( [ 'B', 'E' ] )
  elif ( room[i-1] == 'D' ):
    room[i] = np.random.choice ( [ 'B', 'E' ] )
  elif ( room[i-1] == 'E' ):
    room[i] = np.random.choice ( [ 'C', 'D', 'F' ] )
  elif ( room[i-1] == 'F' ):
    room[i] = np.random.choice ( [ 'E' ] )

plt.hist ( room, bins = 11 )
plt.grid ( True )
plt.show ( )
```

The pattern is not obvious for a low value of n. Get a large enough value that you can look at the histogram and make a guess that explains the results.

6 A Sunny Week in Pittsburgh?

Gloomy people will tell you that the sun never shines in Pittsburgh. Let's assume that the weather data shows that there are three types of daily weather, with the following probabilities:

S:	sunny, 20% chance
C:	cloudy, 50% chance
R:	rainy, 30% chance

What are the changes of a relatively sunny work week in Pittsburgh, that is, a set of 5 days in a row with at least 4 sunny days? If we use `np.random.choice()` to select 5 times (with replacement) from the list of weather choices, each weather type has an equally likely chance of showing up on each day. However, we are allowed an optional additional argument, a vector containing the probability of each choice. So consider a call like this:

```
week = np.random.choice ( [ 'S', 'C', 'R' ], 5, replace = True, p = [ 0.2, 0.5, 0.3 ] )
```

We can call this work week weather wizard repeatedly, until we get a work week with at least 4 occurrences of 'S' and can declare victory.

```
tries = 0
while ( True ):
  tries = tries + 1
  week = np.random.choice ( [ 'S', 'C', 'R' ], 5, replace = True, p = [ 0.2, 0.5, 0.3 ] )
  if ( 4 <= np.sum ( week == 'S' ) ):
    print ( 'After ', tries, ' tries, we got a mostly sunny work week!' )
    print ( week )
    break
```

And if we really wanted to depress ourselves, we could do this experiment 100 times, and estimate how likely on average a sunny work week would be!