Simulation
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

A three way duel occurs in the film "The Good, the Bad, and the Ugly".

Simulation

- *A stochastic process is described by a state that changes over time;*
- *The process includes a random element, which means the evolution from one state to the next is not predetermined;*
- *Simulating the process many times yields statistics such as average and variance;*
- *Histograms can be used to estimate an underlying probability density function;*
- *Simulations can suggest properties of a system that might be provable mathematically.*

1 Simulation

In a previous discussion, we consider a problem in which we imagined that a fly landed at a randomly chosen position on a circular plate, for which the distance d from the center was a quantity of interest. Although mathematically we can't predict the actual position, we can imagine that if this event was an experiment, carried out many times, we could say something about the average distance $\bar{d}$, sometimes written as $\mu(d)$. It's also important to know how far from the mean a typical result will lie, which is reported in the variance $\sigma^2(d)$.

For this simple problem, the mean and variance can be determined directly using integration. However, we were also able to make reasonable estimates using simulation. That is, we made a probabilistic model of the situation, executed it many times, and analyzed the results. Simulation is the creation and analysis of models of physical processes that involve random elements. It allows us to compare our results with mathematical approaches, or even to discover an unknown mathematical pattern, or especially to come up with a statistical description of a random process for which no mathematical approach can be carried out in a reasonable amount of time and effort.

We will look at a variety of relatively simple problems that illustrate the techniques of simulation.

2 Two flies land on a plate

The problem of one fly landing on a square or circular plate was not too difficult to solve exactly, and to approximate computationally. But what happens if we assume that *two* flies land on a square plate, the $[0,1] \times [0,1]$ unit square. Suppose we want to know the average distance d between the pair of them?

Mathematically, it's not so clear how to formulate an integral representation of what is happening. Presumably, we would need some sort of quadruple integral of the pairwise distance over all values of x_1, y_1, x_2, y_2. And while it's easier to sample the unit square than the unit disk, it turns out to be much harder to mathematically determine the mean and variance of the correspodning distance function.

On the other hand, computationally, it's quite easy to estimate these quantities!

```
Purpose:
  Simulate two flies randomly landing in a square of "radius" 1
Input:
  n: the number of simulations
Output:
  d(n): the pairwise distance for each simulation
Compute
  d = array of size n
  Iterate n times, on i
    Generate two random values (x1,y1) in [-1,+1]
    Generate two random values (x2,y2) in [-1,+1]
    Compute d[i] = ||(x1,y1)-(x2,y2)||
  Return d
```

Once we have the array `d` we can easily compute the statistics we are interested in.

For the two flies on a square plate problem, the exact formulas for mean and variance are known, but they were hard to work out, and even a little scary to look at:

$$\begin{aligned}
\mu &= (\sqrt{2} + 2 + 5\log(1.0 + \sqrt{2}))/15.0 && \approx 0.5214 \\
\sigma^2 &= (69 - 4\sqrt{2} - 10(2\sqrt{2}\sinh^{-1}(1) - 25(\sinh^{-1}(1))^2)/225 && \approx 0.0615 \\
\sigma &= \sqrt{\sigma^2} && \approx 0.2479
\end{aligned}$$

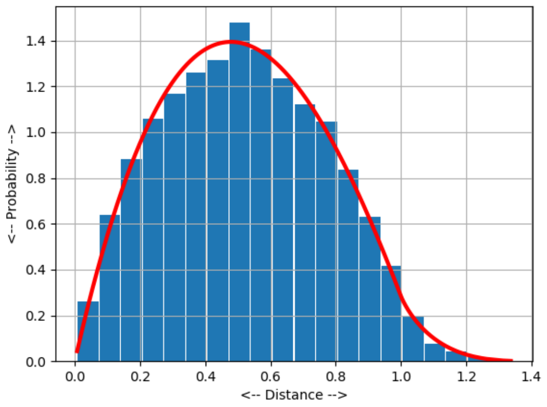
If we use an increasing number of sample points n, we can see that our estimated statistics gradually approach the exact mathematical values, although not monotonically!:

```
      n    dmin   dmean    dmax    dvar    dstd ||dmean-exact||

     10  0.2214  0.5277  0.9535  0.0741  0.2009  0.0063
    100  0.0444  0.5152  1.0284  0.0651  0.2287  0.0062
   1000  0.0113  0.5194  1.1999  0.0609  0.2443  0.0020
  10000  0.0050  0.5185  1.2643  0.0608  0.2482  0.0029

  Exact  0.0000  0.5214  1.4142  0.0615  0.2479
```

Because the region is a square, the frequency versus distance d behaves noticeably differently over the ranges $d < 1$ and $1 < d$. The exact formula for the probaility of a distance d is rather scary:

$$\text{pdf}(d) = \left\{ \begin{array}{ll} 2 \cdot d \cdot (d^2 - 4 \cdot d + \pi) & d \leq 1 \\ 2 \cdot d \cdot (4 \cdot \sqrt{d^2 - 1} - (d^2 + 2 - \pi) \; - 4 * \arctan(\sqrt{d^2 - 1})) & 1 \leq d \end{array} \right\}$$

To see how the probability curve changes for $1 < d$, we convert our frequency histogram into an estimated probabliity density function, and compare it to the exact formula:



It's easy to understand the histogram when it's measuring frequency, that is, how often a value of distance d fits into one of the bins. If we think of a histogram bin as having left and right limits, we could imagine the height of the histogram bar as counting the number of d values that fall in that range, which we could write in pseudo Python as:

```
ybin = sum ( ( dbin_left <= d[:] ) & ( d[:] < dbin_right ) )
```

If we convert our histogram into a PDF by adding the argument `density = True` to the `plt.hist()` call, then we can interpret the plot as follows. For any two distance values d_1 and d_2, the probability that we will observe a distance x between these values is

$$\text{prob}(d_1 <= x <= d_2) = \int_{d_1}^{d_2} \text{pdf}(x)\, dx$$

Thus, the PDF must integrate to 1, since the probability that d has some value in the range must be 1.

The fact that our data matches the exact PDF so well gives us some confidence that our simulation is correct, and that we have used about the right number of samples n in order to get a good estimate.

3 Sudden death

A deck contains 10 cards, numbered 0 through 9. A superstitious man named Allen has been told that, every morning, he must shuffle the deck and then turn over the top card. If it's the 0 card, then it's time for him to die. His sister Brenda finds out about this situation in the afternoon of day 6.

What are the chances that Allen is still alive? On what day does it first become more likely than not that Allen is dead? If Brenda did not interfere, how many days on average would Allen have lived before picking the 0 card? This number could be as low as 1 day, (very unlucky), and it is theoretically possible that Allen never picks the 0 card, although the probability of living a long time must decrease to an extremely low value.

We can answer these questions mathematically, but let's look at a computational approach. Let's start with the last question. How many times on average would Allen pick a card from the deck until drawing the 0?

To estimate the situation, we can imagine creating n copies of Allen, and having each one go through the card reading process until turning up the dreaded zero card.

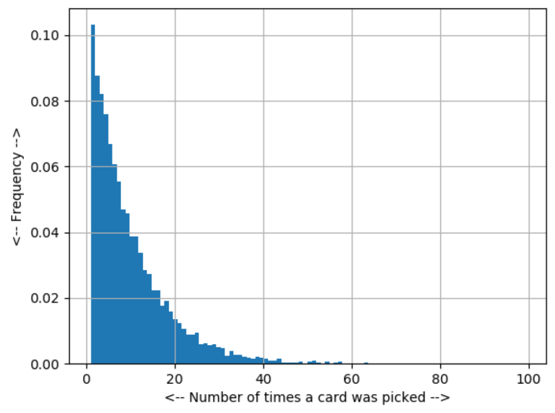
```
Input n, the number of simulations to be done.
Create the deck as a numpy array of the numbers 0 through 9.
For the i-th of the n copies of Allen,
  Start the i-th day count at 0.
  While Allen[i] is still alive:
    Increase day count[i] by 1.
    Shuffle the deck.
    Take the top card.
    If the top card is 0, break.

Return the array of n day counts.
```

Here are some results for various values of n:

```
 Estimate average wait until turning over card 0 out of 10.

       n  min      mean     max         std
      10    2       8.4      22         6.8
     100    1      10.1      36         9.4
    1000    1      10.0      81         9.3
   10000    1      10.1     101         9.5
  100000    1      10.1     103         9.6
```

It looks like our estimates for the mean and standard deviation have settled down. It's no surprise that the minimum is 1. It can be shown using algebra that the mean is exactly 10. It's not clear how we should describe the growth of the maximum wait, as n increases. Theoretically, this value has to go to infinity. *How* it goes to infinity as a function of n can be of interest to mathematicians!

If we look at the PDF version of the data we can see a smooth behavior, and it certainly seems unlikely for Allen to survive 20 days or more!



Instead of shuffling the entire deck and taking the top card, we could have modeled the repeated choosing of a card from a deck by using the function `np.random.randint(low=0,high=10,replace=True)`, since the card chosen is replaced in the deck after each pick.

4 Catching the Last Wolf

A virus is infecting the 300 wolves in Yellowstone Park. The virus is harmless to wolves, but deadly to cattle. Sometimes a wolf will wander out of the park into a nearby cattle herd. The local cattle ranchers demand that a fence be put up around the park (too expensive) or the wolves all be shot (politically controversial) or else every single one of the wolves must be vaccinated.

The decision is made to go with the vaccination option. Of course, the wolves are wild, and are not about to line up and get their shots in order. A park ranger is given the vaccination job. There is a small trap in the park which can be baited; every night, it catches one wolf at random, which can then be vaccinated, and released back into the park. It is an unrealistic part of our model that each time this process captures a wolf uniformly at random. Obviously, we can expect that after a while, we sometimes capture a vaccinated wolf. In fact, the longer we work, the less likely it becomes to capture the few remaining unvaccinated animals. Luckily, every wolf has a serial number tatooed on their neck, so the ranger doesn't vaccinate them twice. Moreover, by recording the serial numbers of the captured wolves, we will know when they have all been vaccinated.

Eventually the ranger will trap every wolf at least once, and complete the task. But can we tell the ranger an estimate for how many days will be needed to encounter every one of the wolves? In particular, because of wolf breeding patterns, the project needs to be completed within a year.

5 The Full Deck Problem

At this point, it helps to use analogy. Suppose once again that we have a deck of cards, and that we repeatedly draw one card, look at it, and then replace it. Suppose that our goal is to see every card in the deck. We could reasonably wonder how long it would take, on average, for us to do this. This problem is similar to the "sudden death" problem, except that now we don't complete our task until every card is exposed at least once. If we can adjust our sudden death model to this new problem, then a second simple adjustment in size will give us a tool for the wolf vaccination effort.

It is probably hard to decide what a reasonable estimate for the average waiting time would be. With a 10 card deck, our average wait to observe the 0 card was 10 draws, and with a 52 card deck it was 52 draws. But how is our waiting time changed when we need to see every card?.

It's time to try a simulation of the "full deck" problem.

```
Purpose:
  Simulate drawing and replacing a card at random until all have been seen.
Input:
  n = number of trials
  card_num = number of cards in deck
Output:
  count: array of size n, number of cards drawn for each trial
For 0 <= i < n
  count[i] = 0
  seen = array of length card_num = 52, initialized to 0
  While ( not all cards have been seen )
    count[i] = count[i] + 1
    Randomly select a card from the deck.
    Note j, the index of the card.
    Increment seen[j]
Return count
```

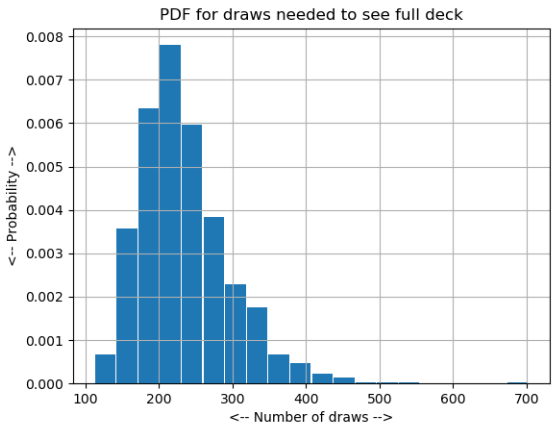
Let's carry out this experiment $n = 10$ times, for a deck of `card_num=52` and record the minimum, mean, and maximum entries in the `seen()` array, as well as the count of the total number of cards we had to pick. Our results show that guessing 100 as the average value of `count` was quite an underestimate!

Try	Min	Mean	Max	Count
0	1	4.79	9	249
1	1	5.65	11	294
2	1	5.21	10	271
3	1	4.40	12	229
4	1	5.62	12	292
5	1	4.15	11	216
6	1	5.29	11	275
7	1	3.60	8	187
8	1	5.85	11	304
9	1	4.21	10	219

We can try to see some regularity in the results by looking at increasing number of tries, up to $n = 10,000$ simulations. At this point, we can believe the estimate that the average value of `count` is about 236, with a typical range of between 170 and 300 cards for any single try.

```
       N  Min    Mean  Max    Std

      10  162  267.00  479   87.20
      20  157  244.65  350   61.07
      50  144  228.08  444   58.25
     100  121  233.26  390   54.67
    1000  113  241.56  557   65.82
   10000   99  236.59  851   65.81
```

Plotting the PDF for the total number of cards drawn shows that, while our most likely result is around 200, there's a pretty good chance of needing between 300 and 400 draws before we see the full deck. This problem has something of a "fat tail", that is, there is a small but significant chance of a very expensive result.



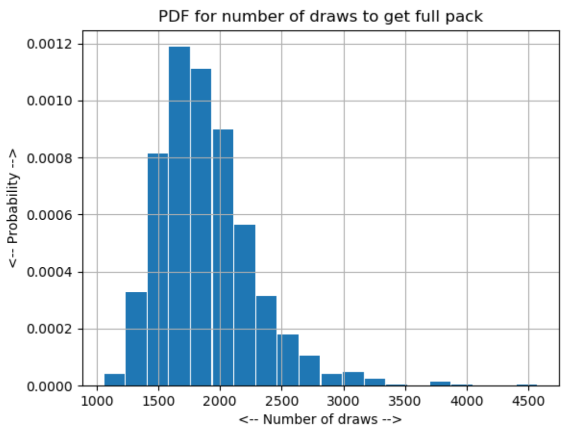
6 Back to the wolf pack!

Now it's a simple task to modify the full deck simulation so that we can interpret it as predicting the results for the wolf vaccination. Essentially, we simply replace the pack of 52 cards by a pack of 300 wolves. Again, we experiment with the number n of trials, hoping to see that our statistics begin to settle down.

```
      N   Min     Mean   Max     Std

     10  1378  1967.80  2910  481.49
     20  1217  1895.95  2749  407.37
     50  1312  1789.46  2880  302.47
    100  1240  1854.07  4252  390.00
   1000  1117  1874.83  3659  371.31
  10000  1082  1884.69  4449  380.79
```

and now we have to give the ranger the bad news that the vaccination project, as designed, is likely to take more than 5 years to complete. At that rate, it's very like the project will fail as new wolves are being born each spring, a factor we did not think we needed to take into account if we could complete the work within a year!

The PDF histogram shows us that the range of 1500 to 2000 days is very likely, but that much longer intervals are also possible!



7 Simulating a duel

In a duel to the death, two enemies alternate taking turns firing at each other until one is hit.

Let us suppose that Anne and Barbara are going to participate in a duel to the death. The duel begins when Anne fires at Barbara; if Anne misses, then Barbara gets a turn to fire at Anne, and the two enemies alternate turns until one of them is hit. Every time Anne fires, she has a probability of `pa` of hitting Barbara; similarly, Barbara's accuracy is `pb`.

Here are several questions we can consider, asuming we know the values of `pa` and `pb`:

- Who is likely to win this duel?
- What is the probablity of winning?
- On average, how many shots will be fired?
- How would these answers be different if Barbara had the first shot?
- If both participants have a 50% accuracy, what is the probability that the first person wins?
- What new values of `pa` and `pb` would guarantee a "fair" duel, in which each participant would have a 50% chance of winning?

Again, these questions can be answered exactly using mathematics and probability theory. However, the answers may involve summing an infinite seriers. And if any detail of the story is changed, a new infinite series would have to be treated. And in some cases, the problem can become too difficult to give an exact mathematical answer.

We consider simulating the duel as follows:

```
Name:
  duel_simulation ( p, n )
Purpose:
  Simulate a duel, returning survivor and number of shots
Input:
  p[2], contains pa and pb, the probability that the first and second
        shooters hit a target
  n, the number of trials
Compute:
  wins = np.array ( [ 0, 0 ] )
  shots = np.zeros ( n )

  for 0 <= i < n          # Perform n trials

    while ( True )

      for 0 <= j < 2      # Let player j take a shot

        r = random_number
        shots[i] = shots[i] + 1

        if r <= p[j]
          wins[j] = wins[j] + 1
          break

return wins, shots
```

Suppose that our accuracy values are `pa=0.25` and `pb=0.30`, so that Barbara is somewhat more accurate than Anne. We might run our simulation for a single trial, and discover that Anne misses, Barbara misses, and then Anne hits, becoming the winner with 3 shots fired.

A single trial doesn't really tell us much about the typical behavior of the duel, the likelihood of Barbara winning, or the typical number of shots fired. To be comfortable with making such statements, we might want to instead consider `n=100` duels, and average the results, as follows:

```
  n = 100
  p = np.array ( [ 0.25, 0.30 ] )
  wins, shots = duel_simulation ( p, n )
  print ( '  Anne winning probability    = ', wins[0] / n )
  print ( '  Barbara winning probability = ', wins[1] / n )
  print ( '  Average number of shots     = ', shots / n )
```

In that case we might get the following information:

```
  Probability Anne wins:          0.51
  Probability Barbara wins:       0.49
  Average number of shots fired = 3.57
```

Surprisingly, this duel seems almost perfectly fair, despite the difference in accuracy between the two players. Although we might suspect 100 duels is enough to estimate the statistics, we can repeat the simulation 1000 times and see what we get:

```
Probability Anne wins:    0.508
Probability Barbara wins: 0.492
Average number of shots fired = 3.794
```

So while the winning probabilities stay close, the number of bullets fired does seem to have been a little underestimated in the first study.

What happens if we switch the order of the two opponents?

```
Probability Barbara wins: 0.614
Probability Anne wins:    0.386
Average number of shots fired = 3.716
```

Although the accuracies of the two players were close, giving the more accurate player the first shot makes a big jump in the survival rates.

Suppose that instead of a duel, we had a `truel`, that is, a group of three people, who take turns firing a shot at one of their opponents. If they are required to fire at the "next" person in order, then it is a simple matter to model what happens. But suppose that we allow each person to choose their target. Suppose, further, that the accuracies of each shooter are known. Then it might make sense for the weaker players to always fire at the strongest one. This is a game in which there can be different strategies, and simulation can quickly show the advantages of choosing your target correctly!