Simulation
Mathematical Programming with Python

https://people.sc.fsu.edu/∼jburkardt/classes/mpp_2023/simulation/simulation.pdf

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

Simulation

- *A dynamic process is described by a state that varies over time;*
- *Typically, time is modeled by equally spaced discrete values;*
- *The state values at the next time depend on values at the previous time;*
- *State values typically also depend on some random influence;*
- *Randomness means a given solution is not determined;*
- *However, by simulating the process many times yields statistics such as average and variance of the state values at the end;*
- *Simulations can suggest properties of a system that might be provable mathematically.*

1 One fly lands on a plate

Suppose that a fly lands at a random spot on a plate P that has a radius of 1 unit. Let d represent the distance of the fly from the center of the plate. What is $\bar{d}$, the average value of d?

This is a question that can be answered with an integral

$$\bar{d} = \frac{\int_P r \cdot r dr d\theta}{\int_P r dr d\theta} = \frac{2}{3} \approx 0.666$$

We could also simulate this problem, if we can figure out how to generate a random point in a circle. It might seem that if r_1 and r_2 are two random values between 0 and 1, that we could represent the (r, θ) coordinates of a random point by $(r_1, 2\pi r_2)$. However, this is incorrect, as you can verify by using such a sampling technique and plotting 200 points. You will see too many points near the origin. The problem is that, going along the radial direction, the density of points is not uniform, like r_1. Instead, it grows like $\sqrt{r_1}$. If, instead, we represent a random point by $(\sqrt{r_1}, 2\pi r_2)$, our results will tend to be evenly scattered around the circle:

```
Purpose:
  Place a fly at random in a circle of radius 1
Compute
  r1, r2 = two uniform random values
  (r,theta) = ( sqrt(r1), 2*pi*r2 )
  (x,y) = (r*cos(theta),r*sin(theta))
Return
  x,y
```

Our algorithm actually computes the (x, y) coordinates of the fly. But in fact, for this problem, we simply want to know the distance of the fly from the origin, which is the value r. So for this special case, we can greatly simplify our computation.

We know that our simulated value d varies as we take n samples. To try to describe this variation, we may be interested in collecting some statistical information as we go. Typically, the minimum, mean, and maximum values are of interest, and we know how to compute them. One more statistical quantity of interest is the standard deviation, represented by σ. This is a measure of how tightly clustered the results are around the average. For instance, if the random quantity behaves according to a normal distribution, then about 68% of the samples would lie within one standard deviation of the average. We will ignore a mathematical detail and define the standard deviation as

$$\sigma(d) = \sqrt{\frac{\sum_{i=0}^{i<n}(\bar{d} - d_i)^2}{n}}$$

(The mathematical detail is whether we divide by n or by $n - 1$. For us, this won't matter.) Given a vector of values `d`, the `numpy()` library can compute statistical quantities such as `np.min(d), np.mean(d), np.max(d), np.std(d)`, where "std" is an abbreviation for the standard deviation.

Since we are estimating a quantity by looking at n samples, we have to expect that our results will only be approximate. Since we know the exact answer for this problem, we can actually compute the error and observe how it changes as we increase the sample size n. We can hope that the approximation generally improves, but this improvement can be somewhat irregular at the beginning (small n) and the end (very large n), when roundoff begins to limit achieving more accuracy.

2 Two flies land on a plate

The one-fly problem was not too difficult to solve exactly, and to approximate computationally. What happens if we assume that *two* flies land on the plate, and we want to know the average distance d between the pair of them? Now it's not so clear how to formulate an integral representation of what is happening. Presumably, we would need some sort of double integral, and we need to express each random point in terms of its (x, y) coordinates so that we can then compute their distance. All in all, it's tricky to think about, messy to express, and probably a little involved to evaluate the final expression, exactly.

On the other hand, the computational simulation almost as simple as for the one-fly example:

```
Purpose:
  Estimate average distance between two flies randomly landing on plate of 1 unit radius
Input:
  n, the number of simulations
Compute
  d = array of size n
  Iterate n times, on i
    Generate two random values, compute (r,theta) for fly 1, convert to (x1,y1)
    Generate two random values, compute (r,theta) for fly 2, convert to (x2,y2)
    d(i) = ||(x1,y1)-(x2,y2)||
Return
  min, mean, max, std of d
```

The exact value for the average distance between the two flies is $\frac{128}{45pi}$, a value which should make you suspect that getting this exact value would involve significantly more mathematical work than for the one-fly problem. However, knowing this exact answer, we can easily observe whether our simulated results converge quickly.

3 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 such a duel. Only because of the alphabet, Anne will get the first shot. We also happen to know that both of them have been practicing target shooting at the gun club, and so we have information about their average accuracy on one s hot, at the given distance used in the duel.

Here are several questions we can consider:

- Who is likely to win this duel?
- On average, how many shots will be fired?
- How would these answers be different if Barbara had the first shot?

We should admit here, immediately, that 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.

Even if we are willing to attack this problem mathematically, it would be helpful to have a quick way of estimating the answers to our questions. To do this, we turn to the idea of simulating the duel:

```
Purpose:
  Simulate a duel, returning survivor and number of shots
Compute:
  shots = 0
  A = 'Anne'
  A_accuracy = ?
  B = 'Barbara'
  B_accuracy = ?
  while ( True )

    r = random_number
    shots = shots + 1
    if r <= A_accuracy
      winner = A
      break

    r = random_number
    shots = shots + 1
    if ( r <= B_accuracy
      winner = B
      break

return winner, shots
```

Suppose that, on average, Anne and Barbara will hit a target 25% and 30% of the time, respectively. We can run our simulation, and discover that Anne misses, Barbara misses, and then Anne hits. So Anne is the winner, and 3 shots were fired.

This suggests our duel simulation is working, but it 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 100 duels, and average the results. In fact, 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, the 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 player 1 wins: 0.508
Probability player 2 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 player 1 wins: 0.614
Probability player 2 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.

If you write a program that simulates this dueling situation, you can easily experiment to explore the effects of differing levels of accuracy, the influence of going first or second, and ways of trying to adjust the rules of a duel to guarantee that both players have an even chance of winning.

4 A "truel", a three way duel, with strategy

Art, Bob and Cal are all enemies, and have decided to try to settle their differences with a three-way duel, which might be called a *truel*. In the order A, B, C, they will each have a chance to fire one shot, repeating the sequence as long as necessary until there is only one survivor.

As in the previous duel exercise, the three participants have been measuring their accuracy beforehand, and we can predict that Art's accuracy on a single shot is 2/6, Bob's is 4/6, and Cal is 5/6 (almost deadly!). In recognition of the importance of going first, the participants have agreed to fire in alphabetical order, so bad shot Art goes first.

We already have an element of randomness here, because no one is a perfect shot. But now we have a new factor: to start with, each person will have a choice of targets. Art might shoot at Bob, or at Cal. Assuming Art misses his target, Bob will also have two choices, and it's possible the same is true for Cal. Of course, eventually, there will only be two participants, and at that point we won't have the extra decision to make. So until one player is eliminated, each player will want a *strategy* to guide them in how to choose their target.

So before we can simulate this truel, we need to decide on the strategies of the players as long as all three are alive:

- #1: Fire at the best shooter: Art and Bob aim at Cal, Cal aims at Bob.
- #2: Fire "alphabetically": Art will aim at Bob, Bob at Cal, and Cal at Art.
- #3: Fire at random: each player flips a coin to choose which other player to aim at.

We assume all players follow the same strategy. But as we will see, strategy #4 specifies different behaviors for each player.

As with the previous duel, we can ask, for each strategy:

- Who is likely to win?
- On average, how many shots will be fired?

To keep things simple, we will run 1000 truels:

```
          Survival Rate
Strategy  Art  Bob  Cal  Shots
------------------------------
#1       0.29 0.58 0.13  3.95   This is Bob's favorite, and Art doesn't do too badly
#2       0.21 0.40 0.39  3.60
#3       0.17 0.35 0.48  3.58   This strategy suits Cal
```

The fact that strategy #3 works well for Cal suggests that in the other two strategies, he is at a disadvantage. In fact, this is obvious. For strategy #1, the other two players, as long as they survive, will always fire at him until he is eliminated. And this seems to make good sense for them. And in strategy #2, the second best shot always aims at Cal, until one of them is eliminated.

Bob likes strategy #1 because Art and he both gets a first shot at Cal before Cal can respond. Strategy #2 is not too bad, because initiallly Bob is fired on by the weakest player, and has a chance to eliminate the best player, at least for a shot or two.

Poor Art has to like strategy #1 because until Cal is eliminated, no one will fire at him. But it certainly seems that Art's poor accuracy dooms him to a low survival rate.

But now Art looks at the rules and discovers that, on his turn, he doesn't actually have to fire his gun at anyone. He can just fire into the air. So now, as long as there are three players, we let Art pass on his turn. Meanwhile, Bob will want to shoot at Cal, who is the best shot and his biggest threat. Similarly, Cal will prefer to shoot at Bob, who represents his biggest threat. This is the rule until someone is eliminated and we are down to a two player duel, in which case the two survivors simply trade shots at each other.

- #4: Until one player is eliminated, Art doesn't fire. Bob and Cal fire at their best opponent.

Actually, this seems very much like strategy #1, "Fire at the best opponent", but the difference is that Art doesn't fire at anyone until there is only one opponent. Let's investigate what effect this strategy will have on the results:

```
          Survival Rate
Strategy  Art  Bob  Cal  Shots
------------------------------
 #4       0.41 0.41 0.18  4.56   This is Art's best strategy!
```

By this simple change, Art dramatically increase his survival chance, while Cal's decreases. The difference is that, with strategy #1, there was some chance that on his first shot, Art would actually hit Bob; in that case, Cal would have the next shot, and Art would be in serious trouble. By instead withholding his fire, Art guarantees that Bob and Cal will shoot it out first, after which he will have the first shot at the survivor.

This surprising result, of course, depends significantly on the particular values we chose for the accuracy of each player, the order of their firing, and the strategy they chose. Because we are able to simulate the results for any choices we make, it is easy to explore many cases, hunting for interesting results like this one.

5 Percolation

Oil drillers are familiar with peculiar structures of underground rock. Often, they are drilling through a material that is a jumbled combination of rock and hollowed-out cavities which might contain oil, gas, or water. Depending on the size, density, and frequency of these cavities, a particular underground "lake" of oil might extend for a few hundred feet, or for miles.

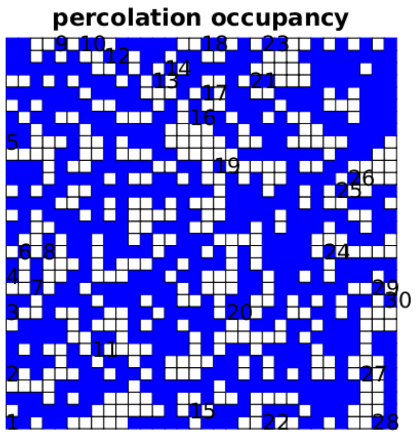
As another example, we might consider an object composed of two metals, one conducting and one not. We suppose the object involves many separate regions of each metal, which are touching, but not melted together. We know the approximate size of a typical "cell" or clump of each metal, and the proportion of the two metals. We ask for the probability that an electical signal can pass from the top to the bottom of the object. This will happen only if there is a some path, formed entirely of the conducting metal, that reaches from top to bottom.

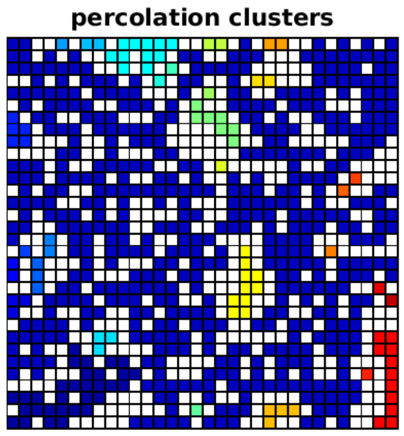
John Hammersley proposed an approach for studying such problems. He imagined a porous stone such as pumice, which is made from volcanic activity. Suppose such a stone is submerged in a bucket of water. Would the center of the stone end up wet or dry? This depends on whether there is at least one microscopic channel between the surface and the center. If the water reaches the center, we say it has done so by *percolation*. This is the same word used to describe one method of making coffee, by having boiling water percolate through a layer of coffee grounds.

A simple 2d model of percolation can be constructed by setting up an $m \times n$ grid of square cells. We will allow some cells to be obstructions (filled, or nonconducting, or solid) and others to be channels (open, conducting, or hollow). We will be looking for the existence of a path from top to bottom. To be a realistic model, the number and size of these cells would have to approximate those of a physical structure. For our learning purposes, though, we will be satisfied with rather coarse grids. We will look at several factors to vary in our problems:

- the total number of cells $m \times n$;
- the aspect ratio $\frac{m}{n}$;
- the probability p that a given cell is conducting;

Once we have chosen m, n, p, we can randomly set the status of each grid cell. The hard part now is to figure out whether the resulting pattern of conducting cells forms a path from top to bottom. If we plot the grid, we can usually spot a path if it exists, at least for our small example grids. We have to come up with a way for the computer to do the same thing, without the benefit of eyes.





The raw occupation grid, and the clustering pattern. $m = n = 32, p = 0.60$.

In the illustration on the left, we can see that a path exists. On the right, each connected "cluster" of cells is given its own color, and we can observe that there is an enormous group of blue cells which do indeed include a path from top to bottom. The idea of clustering will be the key to helping the computer to identify such connecting paths.

An interesting feature of these mathematical percolation simulations is that, if we fix the values of m and n, then as we vary p we observe a fairly sudden transition in behavior. Below a critical value of p, we almost never get a connecting path, which above it, such a path almost always exists. By analogy with how ice changes to water, mathematicians describe this behavior as a *phase transition.* A variety of other numerical problems have this behavior, include something known as the *subset sum* problem, in which a set of integers must be divided into two sets of equal sum.

For our example with $m = n = 32$, the phase transition seems to happen at about $p = 0.60$. In the exercises, you are asked to investigage this claim, as well as to look at results for grids of a different aspect ratio. For these problems, you should use the example program *percolation_simulation.py.*

6 The coughing passenger

Suppose you are a passenger on a plane, and you hear someone two rows ahead of you who is coughing repeatedly. You might begin to worry, because an airborne disease is most likely to spread to nearby neighbors of an infected person. If it was a really long flight, and the disease developed really rapidly, then it might spread first from the coughing passenger to the passenger in front of you, and then to you, taking two steps of infection.

In a simple model of disease propagation over time and space, we might have an $m \times n$ grid of people, who have agreed not to move during the extent of our experiment. We start with one infected person, at some random position (i, j). According to our simplified disease model, the other people are "susceptible", that is, they aren't sick, but they could become so. However, a susceptible person can only catch the disease if they are sitting next to an infected person (left, right, in front or in back). A susceptible person who sits next to an infected person for an hour has a probability p of getting infected during that time.

Now given what we have said so far, it is a certainty that if we wait long enough, everyone in the group must get infected; the one infected person we start with must eventually infect all their immediate neighbors, who must eventually infect all their neighbors, and so on.

This is not a very realistic model of disease. So let's add the idea that an infection only lasts for h hours, after which the person is no longer sick, and in fact becomes immune. An immune person cannot get sick again.

So our model starts with three types of people: $m \times n - 1$ susceptibles (S), 1 infective (I), and 0 recovereds (R). Over time, we expect the value of I to grow, but then it must eventually decreased to zero as the disease "burns out". The value of interest, then, is the number of people who became sick versus those who never got the disease. So we are interested in the value, at the end of the experiment, for the fractions $\frac{S}{m*n}$ and $\frac{R}{m*n}$. If everyone got sick, then these values will be 0 and 1, respectively, so the disease swept through the entire population. Higher values of $\frac{S}{m*n}$ indicate that the disease was less effective in spreading.

Using this model, and the value of $\frac{S}{m*n}$, we could also experiment with small changes to the problem, such as the location of the initial infected person, or the use of a vaccine to reduce the value of p, the transmissibility of the disease, or a treatment that reduces h, the dureation of the disease, or some means of isolating infected persons so they can't transmit the disease to others.

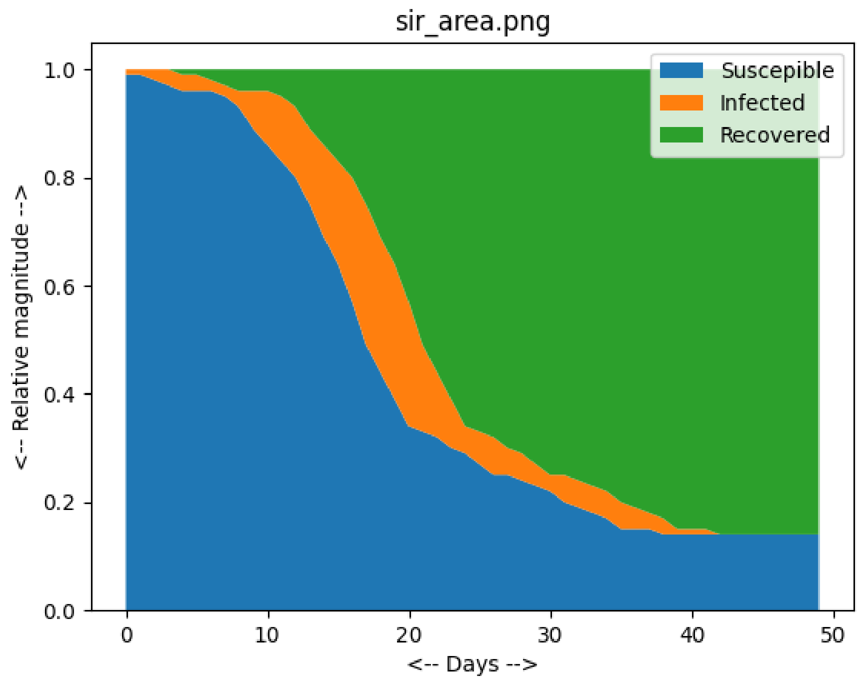
Here is an outline of an algorithm for the SIR model, assuming we are given values for m, n, p, h. The current status is stored in the array P, where $P(i, j) =$

- 1, if person (i,j) is susceptible;
- -k, if person (i,j) is infected for k more hours;
- 0, if person (i,j) is recovered;

We continually updated the array P hour by hour until there are no infected persons:

```
P = ones ( m, n )
i, j = indices of a random location in P.
P(i,j) = -h   # negative value; person will recover in h hours
hour = 0

while any P < 0

  hour = hour + 1

  for (i,j) in the grid

    Q = zeros ( m, n )
    if ( P(i,j) == 1 )
      for north, south, east, west infected neighbor, infect P(i,j) with probability p
      if infected, Q(i,j) = -h
    else if ( P(i,j) < 0 )
      Q(i,j) = P(i,j) + 1
      infected person has one less hour to be infected
    end

  end

  replace P by value of Q

end

print hour, duration of epidemic
print sum ( P ) / m / n, proportion of population that did not get sick
```

Here is a plot of the variation in the values of S, I, R for a model in which at the end of the simulation, S = 0.14 and R = 0.86:



The behavior of S, I, and R for $m = n = 10, p = 0.2$, $h = 4$.

7 Vaccinating the wolves

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 (political suicide) or else every single one of the wolves must be vaccinated.

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. Of course, sometimes a vaccinated wolf is caught in the trap a second time. Luckily, every wolf has a serial number tatooed on their neck, so the ranger doesn't vaccinate them twice. The ranger also knows how many wolves there are in the park.

So every morning there's a wolf in the trap, but often it's a wolf that's already been vaccinated. It seems clear that eventually the ranger will trap every wolf at least once, and complete the task. But can the ranger estimate how many days this is going to take?

We can model this problem as though we had a deck of cards. We repeatedly take a card at random, look at it, and return it to the deck. If the deck has 52 cards, how many times are we likely to have to draw a card before we have seen every card? Now we have a feeling for the problem. The answer has to be at least 52; a value of 60 or 70 seems low, and a value of 100 might seem almost right. Before we think about an exact answer, let's just do a simulation.

```
Purpose:
  Simulating drawing and replacing a card at random until all have been seen
Initialize:
  seen = zero array of length 52
```

```
Iterate:
  Draw a card at random with index i
  Increment seen(i)
  If all entries of seen() are greater than zero, break
Return
  Sum(seen)
```

Let's carry out this experiment 10 times, and record the minimum, mean, and maximum entries in the `seen()` array, as well as the total. Our results show that guessing 100 as the average wait was quite an underestimate!

Try	Min	Mean	Max	Total
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

If we carry out 10,000 simulations, the average number of times we have to draw a card is 236.

If we can simulate a deck of 52, we can do the same for a wolf pack of 300. But now we should expect that the number of times we have to trap a wolf in order to see every one of them is going to be, perhaps, 1000 or more.

8 Best of the Bunch

There's a new game show that gives away money, and you're the next player. Here's how it works. There are n bags, each containing a different number of gold coins. You are allowed to open a bag and see how many coins it contains; then you must decide to take this bag as your winnings and stop, or move on to the next bag.

Your ideal goal would be to choose the bag with the most number of coins. But there is no way to tell which bag that is, unless you look at every bag, and by then, you've almost certainly already rejected that bag.

Strategy #1 is to open the first bag and take it. Strategy #2 is to open every bag, taking the last one. Both strategies have only a $\frac{1}{n}$ chance that you will pick the most valuable bag. This seems the best you can do, since you have no additional knowledge about the arrangement of the bags.

But is there some strategy that gives you a better chance?

Strategy #3 says look at and reject the first three bags, but remember the maximum value you saw. Then take the very next bag whose value exceeds that number. Let's try these strategies a few times, and see how often we get the best bag. To keep things simple, we will assume the bags have 1 through 10 coins, in permuted order. In a real game, the values would vary in a more devious way.

```
                                      Strategy
       Money bags                    #1  #2  #3
-------------------------------     ----------
[ 7 10  3  4  2  9  5  1  8  6]      7   6   6
[ 9  3  8  2 10  4  7  6  1  5]      9   5  10
[ 2  7  4  1  3  8  9  6  5 10]      2  10   8
[ 8  6  9 10  7  5  2  4  3  1]      8   1  10
[ 4  8  3  1  5  6  9  2 10  7]      4   7   9
[ 5  7  6  8  3  2 10  4  9  1]      5   1   8
[ 3  5  4  1  9 10  6  8  7  2]      3   2   9
[ 8  4  9  7  2  5  6  3  1 10]      8  10  10
[ 6  1  4  7  9 10  5  3  2  8]      6   8   7
[ 1 10  5  6  9  4  8  7  2  3]      1   3   3
                                    --  --  --
                           Totals: 53  53  80
```

The average value of a bag in this setup is 5.5 units, and so the scores for the first two strategies are pretty much the same as picking any bag at random. The third strategy, however, is outperforming them somehow. What information does looking at the first 3 bags give you? (And what's so magical about 3?)

9 Equal Opportunity

Now let's change the "Best of the Bunch" example. An employer has a job to offer, and it does not require any special skills. The employer declares that every applicant will have an equal chance of getting the job. The employer accepts the resume of the first applicant. Then, one by one, as each applicant offers a resume, the employer can throw away the one being held and take this new one, or refuse it. The employer uses some strategy to decide how to do this correctly. When there are no more candidates in line, the person whose resume was kept gets the job.

What strategy does the employer use so that every candidate really has an equal chance of getting the job?

The same problem occurs if a computer program is reading a file, containing paragraphs of text separated by blank lines. The program only gets one chance to read this file, and has to select a paragraph at random. It doesn't know how many paragraphs of text there are. It wants every paragraph to have an equal chance of being selected. Once the program has read the file, it must print out the paragraph it selected.

While this might seem impossible at first, there is a strategy that will correctly choose from an initially unknown number of options, in such a way that each option has an equal chance of being selected.

We do this by making a series of temporary selections.

- Given the first option, we select it.
- If a second option is available, then with probability $\frac{1}{2}$, it replaces our old choice.
- If a third option is available, then with probability $\frac{1}{3}$, it replaces our old choice.
- We continue in this way as long as necessary.
- If the last option is number n, with probability $\frac{1}{n}$, it replaces our old choice.

At the end of this process, every available option had an equal chance of being the one chosen.

The algorithm is not hard to probram, but it can actually be tricky to come up with a good test that convinces that every option has equal probability, and so this is left as an exercise for you!

10 Exercises

1. In the one fly problem, it is claimed that we can greatly simplify the calculation of the distance of a random fly from the origin. In fact, we can compute the average distance of all n flies from the origin, using a single line of Python code. Can you work out what this line would be?

2. Modify the one fly problem so that the plate is square, with a side length of 2. Now what is the average distance of the fly from the center of the plate?
3. Computationally verify the average distance reported for the "one fly problem".
4. Computationally verify the average distance reported for the "two fly problem".
5. Sample the unit circle with (r, θ) points $(r_1, 2\pi r_2)$, where r_1 and r_2 are random numbers. Compute about 200 such points and plot them. Do the points seem well distributed?
6. In the duel example, allowing Anne to go first made for roughly equal survival chances for both players; but if we switched the order, Barbara was much more likely to survive. One way to try to decrease the advantage of going first would be to give the first player one shot, then the second player two shots. After that, the players would alternate taking two shots at a time. Implement this idea. Is it still roughly fair to have Anne go first? Now what happens if Barbara goes first?
7. In the truel example, suppose the players Art, Bob, and Cal adopt strategy #1 (shoot at the best opponent). The survival rates for this case are 0.29, 0.58, 0.13. Can you adjust the accuracies of Art, Bob, and Cal so that, even with this strategy, the survival rates become roughly equal?
8. For the percolation problem, verify the claim that the critical probability is roughly $p = 0.60$. For each of the values $p = 0.58, 0.59, 0.60, 0.61, 0.62$, simulate the problem 10 times and report the average number of connecting paths found.
9. For the percolation problem, estimate the critical value of the occupation probability p for a grid with the width n fixed at 32, but height set to the values $m = 4, 8, 16, 32, 64$.
10. For the percolation problem, try creating a function that accepts an $m \times n$ array A of 0's and 1's, and relabel the data into clusters of neighboring 1 values; Make a new array C where each 1 value is replaced by the index of the cluster it belongs to. You might use the the following 8×10 sample grid. Think about how you could quickly examine the C array and determine if there is a path from the top to the bottom.

```
          A                    C

0 1 0 0 1 1 1 1 1 1    0 1 0 0 2 2 2 2 2 2
1 1 1 0 1 1 0 0 0 0    1 1 1 0 2 2 0 0 0 0
0 0 1 1 0 0 1 1 1 0    0 0 1 1 0 0 1 1 1 0
1 1 0 1 1 1 1 0 1 0    3 3 0 1 1 1 1 0 1 0
1 0 0 1 0 0 0 0 1 0    3 0 0 0 0 0 0 0 1 0
1 0 0 0 0 1 1 1 1 0    3 0 0 0 0 1 1 1 1 0
1 1 0 0 1 1 1 0 0 0    3 3 0 0 1 1 1 0 0 0
1 0 0 0 1 0 0 0 1 1    3 0 0 0 1 0 0 0 4 4
```

11. For the SIR model with $m = 10, n = 10, p = 0.2, h = 4$, estimate the expected value of $\frac{S}{m*n}$ by running the simulation 50 times.
12. Consider the SIR model with $m = 10, n = 10, p = 0.2, h = 4$. Given the option of reducing the transmissibility to $p = 0.18$, or reducing h to 3, which option results in a higher expected value for $\frac{S}{m*n}$.
13. Write a program for the problem of vaccinating a pack of 300 wolves. Try 1000 simulations, and report your estimate for how many times a wolf must be trapped before all are seen. Compare your result to the exact value, which is $\sum_{i=1}^{299} \frac{1}{i}$.
14. Write a program for the "best of the bunch" example, in which you can vary the values of n (the number of bags) and k (the number of bags you open before choosing the next best one). First test your program on the small case of $n = 10$. Now move to the case $n = 100$. Compare the strategies using $k = 3$ and $k = 6$, by running your example 20 times. You should see that increasing k seems to improve the average winnings. In fact, for a problem with n bags, the best strategy is to use $k = \lfloor n/e \rfloor$, where e is our friend, the base of the natural logarithm system. Produce a table for $n = 100$ showing the average winnings over 20 games for $k = 3, 6, 10, \lfloor 100/e \rfloor$.
15. Write a program for the "equality opportunity" example. In order to test it, pick a value n at random.

Don't go crazy, choose a value of n no greater than 20. Then have your program "interview" a sequence of job applicants, without knowing how many there are. When the last applicant has been interviewed, the program returns its choice. By repeating this process many times, show that each applicant seems to have had a roughly equal chance of being selected. Remember, the important thing here is that the program does not know how many applicants there are until the line suddenly ends.