RESEARCH DIVISION - EXAMPLE INTERVIEW QUESTIONS 2016

SOLUTIONS BY MARTIN CITOLER

Problem 1. *Coin toss*

a. If you toss a coin until you see two heads in a row, how many tosses are required on average?

Solution. We assume that the coin might be unfair and shows a head with probability p. Let $X = \#$ tosses to see head-head. We find the expected value of X by conditioning on the outcome of the first toss

$$\begin{aligned}\mathbb{E}[X] &= p\mathbb{E}[X|head] + (1-p)\mathbb{E}[X|tail] \\ &= p\left(p\mathbb{E}[X|head, head] + (1-p)\mathbb{E}[X|head, tail]\right) + (1-p)\left(1 + \mathbb{E}[X]\right) \\ &= p\left(p2 + (1-p)\left(2 + \mathbb{E}[X]\right)\right) + (1-p)\left(1 + \mathbb{E}[X]\right).\end{aligned}$$

Now we only need to solve for $\mathbb{E}[X]$. A straightforward computation yields

$$\mathbb{E}[X] = \frac{1+p}{p^2},$$

which means that we expect 6 tosses with a fair coin.

b. If you toss a coin until you see a head followed by a tail, how many tosses are required on average?

Solution. We can solve this case in a similar fashion. Consider $X = \#$ tosses to see head-tail, we have

$$\begin{aligned}\mathbb{E}[X] &= p\mathbb{E}[X|head] + (1-p)\mathbb{E}[X|tail] \\ &= p\mathbb{E}[X|head] + (1-p)\left(1 + \mathbb{E}[X]\right),\end{aligned}$$

now observe that once we have tossed a head, the experiment reduces to tossing a coin until we see a tail. In other words, $\mathbb{E}[X|head] = 1 + \mathbb{E}[Y]$, where $Y = \#$ tosses to see a tail. We can compute this as follows

$$\mathbb{E}[Y] = p\mathbb{E}[Y|head] + (1-p)\mathbb{E}[Y|tail] = p(1 + \mathbb{E}[Y]) + (1-p),$$

which results in $\mathbb{E}[Y] = \frac{1}{1-p}$. Plugging this in the original equation we obtain

$$\mathbb{E}[X] = \frac{1}{p(1-p)},$$

which gives 4 tosses for a fair coin.

c. Why are these answers different, given that head-tail and head-head have an equal chance of appearing after two coin tosses?

Date: March 20, 2017.

Solution. The key fact is that we are not tossing the coins in blocks of two but we are doing it sequentially. Consider that we have just seen the first head. In case a., if we toss another head we are done and, similarly, in case b. if we toss a tail we are done. However, if we toss the undesired result, in case a. we need to "start over" because the head-head chain has been broken. In contrast, in case b. the sequence head-head only needs a tail afterwards.

Problem 2. *You want to create an open box with a square base and have only a fixed amount of material. How do you maximize the volume of the box, given the constraint on the total surface?*

Problem 3. *Given that A and B are covariance matrices, which of these are also covariance matrices?*

a. $A + B$
b. A^2
c. AB

Solution. content...

Problem 4. *Consider a circle and an equilateral triangle inscribed inside the circle. Write some code to compute the following by simulation:*

a. Take two points randomly on the circumference of the circle. Connect and measure the length of the chord through the two points. What is the probability that the chord is longer than the side of the triangle?
b. Take a random point inside a circle. This is the midpoint of only one chord. What is the probability that this chord is longer than the side of the triangle?
c. Take two points randomly inside a circle. Consider the chord that goes through the two points. What is the probability that the chord is longer than the side the triangle?

Solution. Note that the radius of the circle does not matter in any of the situations above, therefore, we can safely assume that it is equal to 1. We will write the answers using Python.

a. The length of a chord is uniquely determined by the angle that the endpoints form with the center of the circle. For example, it is easy to see that the side of an inscribed equilateral triangle is determined by $\frac{2\pi}{3}$. Further, choosing a random point on the circumference of the circle is equivalent to drawing a sample from a uniform distribution from 0 to 2π. Thus, the problem can be solved using the following code:

```
import numpy as np

TWO_PI = 2*np.pi
N = 100000
angle_equilateral = TWO_PI/3
success_count = 0
for _ in range(N):
    point1 = TWO_PI*np.random.rand()
    point2 = TWO_PI*np.random.rand()
    angle = min([abs(point1-point2), TWO_PI -abs(point1-point2)])
    if angle > angle_equilateral:
```

```
        success_count += 1
print(success_count/N)
```

This results in a probability of $\frac{1}{3}$.

b. In this case, the length of the chord is uniquely determined by the distance of the point to the center of the circle. Thus, the problem reduces to comparing the radius of a random point inside the circle and the corresponding one on the equilateral triangle, which is $\frac{1}{2}$. To sample a point uniformly on the inside of the circle, we can sample a point (x, y) where x, y are drawn from a uniform distribution from 0 to 1 and discard the point if it does not lie on the circle. Alternatively, since the area of a circle of radius R grows like R^2, to sample a radius from a uniform distribution we can sample a uniform distribution from 0 to 1 and then take the square root (up to some constant).

```
import numpy as np

def getPoint():
    x = np.rand()
    y = np.rand()
    while x**2 + y**2 > 1:
        x = np.random.rand()
        y = np.random.rand()
    return [x,y]

N = 100000
radius_equilateral = 0.5
success_count = 0
for _ in range(N):
    point = getPoint()
    radius = (point[0]**2 + point[1]**2)**0.5
    if radius > radius_equilateral:
        success_count += 1
print(success_count/N)
```

This gives a probability of $\frac{3}{4}$.

c. We can draw points uniformly from the inside of the circle as above, compute the length of the chord determined by them and compare it to the side of the triangle, which is $\sqrt{3}$.

```
import numpy as np

def getPoint():
    x = np.rand()
    y = np.rand()
    while x**2 + y**2 > 1:
        x = np.random.rand()
        y = np.random.rand()
    return [x,y]
```

```
def getLengthChord(point1, point2):
    # get the line generated by the points: line(t) = v*t + point1
    v1, v2 = point1[0]-point2[0], point1[1]-point2[1]
    def line(t):
        v = np.array([v1, v2])
        p = np.array(point1)
        return t*v + p
    # get intercepts with the circle
    a = (v1**2 + v2**2)
    b = (2*v1*point1[0] + 2*v2*point1[1])
    c = point1[0]**2 + point1[1]**2 -1
    t1 = (-b + (b**2 - 4*a*c)**0.5)/(2*a)
    t2 = (-b - (b**2 - 4*a*c)**0.5)/(2*a)
    x = line(t1)
    y = line(t2)
    return np.linalg.norm(x-y)

N = 100000
side_equilateral = 3**0.5
success_count = 0
for _ in range(N):
    point1 = getPoint()
    point2 = getPoint()
    len_chord = getLengthChord(point1, point2)
    if len_chord > side_equilateral:
        success_count += 1
print(success_count/N)
```

This gives a probability of ~ 0.59.

Problem 5. *Given two words, the* edit distance *between them is defined as the number of operations needed on the first word to convert into the other, where the allowed operations are insertion, removal and replacing of letters. How do you write a function that computes this distance between two words?*

Solution. content...

Problem 6. *A dynamic model is defined as follows:*

- *$State(n, m, t)$ is the state of the node (n, m) at time t. This value is either 0 or 1.*
- *A neighborhood of node (n, m) consist of 8 nodes around it. Denote $LifeAround(n, m, t)$ as the number of nodes in the neighborhood of (n, m) at time t that are in state 1.*
- *Evolution:*
  - *If $State(n, m, t) = 0$ (dead), then $State(n, m, t + 1) = 1$ (alive) if and only if $LifeAround(n, m, t) = 3$.*
  - *If $State(n, m, t) = 1$ (alive), then $State(n, m, t + 1) = 0$ (dead) if and only if $LifeAround(n, m, t) < 2$ or $LifeAround(n, m, t) > 3$.*

a. *Write a function that starts with a given state and develops the model through n steps on a fixed grid with zeros outside the boundary.*
b. *Write a similar function that has no boundary.*