# Övningar

Slumpade gruppen

February 11, 2024

## Övning 1

### 1

För varje av våra $n$ vänner har vi $k$ alternativ, så vi kan välja vykorten på $k^n$ sätt.

### 2

Vi kan per definition välja vilka $n$ vykort vi skickar på $\binom{k}{n}$ olika sätt

### 3

För en given vän har vi $\frac{k(k-1)}{2}$ olika sätt att välja vykorten, $k$ val för det första och $k-1$ val för det andra. Vi delar med 2 för att inte dubbelräkna. Detta val görs $k$ gånger, så svaret är

$$(\frac{k(k-1)}{2})^n$$

## Övning 2

**i)** $\binom{n}{k}$

Det följer direkt från formel att

$$\binom{n}{k} = \frac{n!}{k!(n-k!)} = \frac{1}{k!}n(n-1)(n-1)...(n-(k-1))$$

vilket är ett polynom i $n$ för fixt $k$.

## Övning 3

Vi räknar antalet funktioner $f : [x] \to [n]$. Detta räknas av högra ledet, ty för varje element i $[n]$ finns det $x$ olika element de kan träffa, så det finns $x^n$ olika funktioner $f : [x] \to [n]$.

Betrakta nu $[n]/\sim_f$, där $a \sim_f b \Leftrightarrow f(a) = f(b)$. $\sim_f$ är uppenbarligen en ekvivalensrelation, så den partitionerar $[n]$. Varje funktion $f : [x] \to [n]$ ger upphov till en unik relation $\sim_f$. Om vi vet att $|\text{Im}(f)| = k$ kan $\sim_f$ per definition väljas på $\left\{{n \atop k}\right\}$ olika sätt. Givet $y \in [x]$, skriv $\bar{a} = \{a \in [n] | f(a) = y\}$ och $f(\bar{a}) = f(a)$. Om $\bar{a_1}, \bar{a_2}, ..., \bar{a_k}$ är en partitionering av $[n]$ kan $f(\bar{a_1})$ väljas på $x$ sätt. Sedan har vi brukat ett element i $[x]$, så $f(\bar{a_1})$ väljas på $x-1$ sätt, och så vidare, tills vi slutligen kan välja

$a\bar{}_k$ på $(x-k)+1$ sätt. Alltså finns det $\left\{ {n \atop k} \right\} x(x-1)...(x-k+1)$ olika funktioner $f : [n] \to [x]$ där $|\mathrm{Im}(f)| = k$ . Eftersom $k = 1$ *eller* $k = 2...$ *eller* $k = n$, finns det enligt additionsprincipen

$$\sum_{k=1}^{n} \left\{ {n \atop k} \right\} x(x-1)...(x-k+1)$$

olika funktioner $f : [n] \to [x]$. Slutligen, då $\left\{ {n \atop 0} \right\} = 0$ är

$$\left\{ {n \atop 0} \right\} + \sum_{k=1}^{n} \left\{ {n \atop k} \right\} x(x-1)...(x-k+1) = \sum_{k=0}^{n} \left\{ {n \atop k} \right\} x(x-1)...(x-k+1)$$

vilket avslutar beviset.

Föreknlat så partitionerar vi först mängden i $k$ delmängder, och sedan väljer vi vilket element i målmängden varje delmängd ska träffa under funktionen.

## Övning 4

### i

Påstående:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

Bevis:

Antag att vi har en mängd $M$ med $n+1$ element, och att ett av dessa elememt är $x$. Vi vet, enligt defintion, att denna mängd kan partitioneras på $B_{n+1}$ olika sätt. Mängden kan även partitioneras på följande sätt: Dela först upp mängden i två disjunkta delmängder. En av dessa kommer innehålla $x$. Om den mängden som inte innehåller $x$ innehåller $k$ element, kan dessa väljas på $\binom{n}{k}$ olika sätt (vi får ju inte välja $x$). Denna mängd kan sedan partitioneras på $B_k$ olika sätt, så för att givet $k$ finns det $\binom{n}{k} B_k$ olika sätt att partitionera mängden. $k$ är som minst 0, ifall vi valde att dela upp vår mängd $M$ i den ursprungliga mängden och den tomma mängden. $k$ är som högst $n$, ifall vi valde att dela upp $M$ i $\{x\}$ och $M \setminus \{x\}$. Eftersom $k = 0$ *eller* $k = 1$ *eller ... eller* $k = n$ finns det, enligt additionsprincipen,

$$\sum_{k=0}^{n} \binom{n}{k} B_k$$

olika sätt att partitionera $M$, vilket avslutar beviset.

### ii

Definiera

$$E(x) = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$$

vilket enligt vår rekursionsformel är

$$E(x) = \sum_{n=0}^{\infty} \sum_{k=0}^{n-1} \binom{n-1}{k} B_k \frac{x^n}{n!}$$

Vi plockar ut det första elementet $B_1 = 1$ och shiftar sedan index $n \to n-1$ vilket ger

$$E(x) = 1 + \sum_{n=0}^{\infty} \sum_{k=0}^{n} \binom{n}{k} B_k \frac{x^{n+1}}{(n+1)!}$$

Deriverar vi får vi

$$E'(x) = \sum_{n=0}^{\infty} \sum_{k=0}^{n} \binom{n}{k} B_k \frac{x^n}{n!} = \sum_{k=0}^{\infty} B_k \sum_{n=k}^{\infty} \binom{n}{k} \frac{x^n}{n!}$$

(Obs ingen aning om vi faktiskt får byta index sådär)

Shifta nu index $n \to n-k$ och få

$$E'(x) = \sum_{k=0}^{\infty} B_k \sum_{n=0}^{\infty} \binom{n+k}{k} \frac{x^{n+k}}{(n+k)!} = \sum_{k=0}^{\infty} B_k \sum_{n=0}^{\infty} \frac{(n+k)!}{n!k!} \frac{x^{n+k}}{(n+k)!} =$$

$$\sum_{k=0}^{\infty} B_k \sum_{n=0}^{\infty} \frac{x^k x^n}{k!n!} = \sum_{k=0}^{\infty} \frac{x^k}{k!} B_k \sum_{n=0}^{\infty} \frac{x^n}{n!} = e^x E(x)$$

från Tylorutvecklingen på $e^x$. Alltså har vi följade differentialekvation

$$E'(x) = e^x E(x)$$

vilket genom observtion har lösningen $E(x) = Ce^{e^x}$. Begynnelsevilkoret är $E(0) = B_1 = 1 \Rightarrow Ce^{e^0} = Ce = 1 \Rightarrow C = \frac{1}{e}$, alltså är $E(x) = \frac{1}{e} e^{e^x}$.

Nu måste vi bara Taylorutveckla så är vi klara

$$\frac{1}{e} e^{e^x} = \frac{1}{e} \sum_{n=0}^{\infty} \frac{(e^x)^n}{n!} = \sum_{n=0}^{\infty} \frac{1}{n!} \frac{1}{e} \sum_{k=0}^{\infty} \frac{(xk)^n}{k!} = \sum_{n=0}^{\infty} \frac{x^n}{n!} \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}$$

Jämför vi serierna termvis ser vi att

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{k!}$$

vilket avslutar beviset.

## iii

Vi vill visa att

$$B_n = \sum_{\substack{k_1, k_2, \ldots, k_n \geq 0 \\ k_1 + \ldots + n k_n = n}} \frac{n!}{\prod_{i=0}^{n} k_i! (i!)^{k_i}}$$

Vi vet att vänsta ledet räknar antalet partitioner av $[n]$. Vi visar att även högra ledet även gör det.

Vi kan tolka $k_i$ som antalet delmängder i vår partition som innehåller $i$ element. Då innehåller unionen av alla delmängderna $k_1 + 2k_2 + \ldots + nk_n = n$ element som önskat. Betrakta en fix uppsättning $k_1, k_2, \ldots, k_n$. Vi kan bilda delmängder av $[n]$ på följande sätt: Ta en permutation av $[n]$. Denna kan bildas på $n!$ olika sätt. Låt sedan de första $k_1$ elementen vara delmängder i sig,

låt efter dessa de följande $2k_2$ vara delmängder i par, och bilda delmängder på detta vis tills vi partitionerat hela $[n]$.

Här kommer ett förtydligande exempel för $n = 5$, $k_1 = 2$, $k_3 = 1$, $k_3 = k_4 = k_5 = 0$ och permutationen $(1, 5, 4, 2, 3)$. Då är vår partitionering $\{1\}\{5\}\{2, 3, 4\}$.

Nu måste vi dela bort alla partitioner som $n!$ räknar flera gånger. Dels spelar inte ordningen på våra delmängder med $i$ element roll. I vårt exempel så vill vi inte räkna $\{1\}\{5\}\{4, 2, 3\}$ och $\{5\}\{1\}\{4, 2, 3\}$ som olika paritioneringar. Således delar vi med $k_i!$ för att endast räkna dessa en gång. Elemnetets ordning i delmängden spelar inte heller någon roll, så för varje delmängd med $i$ element delar vi med $i!$ för. I exemplet vill vi inte räkna $\{5\}\{1\}\{4, 2, 3\}$ och $\{5\}\{1\}\{2, 3, 4\}$ som olika partitioneringar. Detta gör vi för alla $k_i$ delmängder, varför vi upphöjer $i!$ i $k_i$. Alltså finns det för vår givna uppsättning $k_1, k_2, ..., k_n$ exakt

$$\frac{n!}{\prod_{i=0}^{n} k_i!(i!)^{k_i}}$$

partitioneringar av $[n]$. Eftersom val av $k_1, k_2, ..., k_n$ är ömsesidigt uteslutande får vi den totala mängden partitioneringar genom att summera över alla dessa, d.v.s. vi får att antalat partitioneringar av $[n]$ är

$$\sum_{\substack{k_1,k_2,...,k_n \geq 0 \\ k_1+...+nk_n=n}} \frac{n!}{\prod_{i=0}^{n} k_i!(i!)^{k_i}}$$

vilket skulle visas.

## Övning 5

### Kombinatoriska biten

Vi räknar antalet demängder av $[n]$ som är har ett jämnt antal element. Det finns $2^n$ delmängder av $[n]$. Vi visar att hälften, d.v.s. $\frac{2^n}{2} = 2^{n-1}$ av dessa har ett jämnt antal element med snabbt induktionsbevis. Basfallet är $n = 1$, och $[1]$ har delmängderna $\{\}$ och $[1]$. Antag att det stämmer för $[m]$. Alla delmängderna till $[m + 1]$ är endera delmängder till $[m]$, och per antagande är hälften av dessa av jämn storlek, eller en delmängd till $[m]$, s innehåller $m + 1$, och hälften av dessa var udda innan vi lade till $m + 1$, så hälften är nu jämna. Detta avslutar beviset.

Å andra sidan, om en delmängd innehåller $2k$ element kan dessa väljas på $\binom{n}{2k}$ olika sätt. Eftersom $k$ kan variera från 0 till $\lfloor \frac{n}{2} \rfloor$ (vi golvar $\frac{n}{2}$ för att få ett heltal om $n$ är udda, och då motsvarar $2\lfloor \frac{n}{2} \rfloor$ mängden med alla förutom ett element), så kan alla delmängderna med ett jämnt antal element byggas på

$$\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k}$$

olika sätt, vilket avslutar beviset.

### Algebreiska biten

Notera att

$$\sum_{n=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} + \sum_{n=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n}{2k+1} = \sum_{n=0}^{\lfloor \frac{n}{2} \rfloor} [\binom{n}{2k} + \binom{n}{2k+1}] = \sum_{k=0}^{n} \binom{n}{k}$$

där $\binom{n}{2\lfloor \frac{n}{2} \rfloor + 1} = 0$ om $n$ är jämnt. Från binomialsatsen har vi att

$$\sum_{k=0}^{n} \binom{n}{k} = \sum_{k=0}^{n} \binom{n}{k} 1^{n-k} 1^k = (1+1)^n = 2^n$$

## Övning 6

Vi har att $n \geq 1$ är ett heltal med primtalsfaktorisering $n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_r^{\alpha_r}$.Låt antalet heltal mellan 1 och n som är relativt prima n beteckans $n^*$. Vi kan skriva $n^*$ som : $n^* = |\{x \leq n : SGD(n,x) = 1\}|$. Eftersom det finns totalt n positiva heltal mellan 1 och n kan vi skriva $n^* = n - |\{x \leq n : SGD(n,x) > 1\}|$. $SGD(n,x) > 1$ omm en primfaktor av n delar x, så vi får: $n^* = n - |\{x \leq n : p_1 \mid x \vee p_2 \mid x \vee ... \vee p_r \mid x\}|$. Låt för $1 \leq k \leq r, T_k = \{x \in \mathbb{Z}_+ : x \leq n, p_k \mid x\}$ vi får då

$$n^* = n - |T_1 \cup T_2 \cup ... \cup T_r| = n - \sum_{k=1}^{r} [(-1)^{k+1} \sum_{\substack{J \subseteq [r] \\ |J| = k}} |\bigcap_{j \in J} T_j|]$$

$$= n - \sum_{k=1}^{r} [(-1)^{k+1} \sum_{1 \leq i_1 < ... < i_k \leq r} \frac{n}{p_{i_1} p_{i_2} ... p_{i_k}}] = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_r})$$

## Programmeringsövningar

## Övning 1

proccessen kan simuleras med denna kod:

```python
import numpy as np
import matplotlib.pyplot as plt

def ball_pickup(red, blue):
    total_balls = sum([red, blue])
    probability_red = red / total_balls
    probability_blue = blue / total_balls


    chosen_ball = np.random.choice(['red', 'blue'], p=[probability_red,
    ↪  probability_blue])

    return chosen_ball
def poly_urn(start_vector, transition_matrix, steps):
    red = []
    blue = []
    ratio = []
    current_vector = start_vector
    for i in range(steps):
        ball = ball_pickup(current_vector[0], current_vector[1])
        if ball == 'red':
            current_vector = current_vector + transition_matrix[0] -[1,0]
            red.append(current_vector[0])
            blue.append(current_vector[1])
            ratio.append((red[-1])/(blue[-1] + red[-1]))
        elif ball == 'blue':
            current_vector = current_vector + transition_matrix[1] -[0,1]
            red.append(current_vector[0])
            blue.append(current_vector[1])
            ratio.append((red[-1])/(blue[-1] + red[-1]))
    return current_vector, red, blue, ratio
```

## Övning 2

### 1

Om våran startvektor är $[1, 1]$ har vi 1 röd och 1 blå boll i urnan. Om övergångsmatrisen ges av $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ kommer vi för varje röd boll vi plockar upp att addera en röd boll till urnan och för varje blå boll vi plockar upp att addera en blå boll till urnan. Vid första dragningen är sannolikheten att dra en röd boll och sannolikheten att dra en blå boll samma nämligen $\frac{1}{2}$. Säg att vi drar en röd boll i första dragningen då har vi totalt 3 bollar i urnan varav 2 är röda. Vid andra dragningen är det därför större sannolikhet att dra en röd kula igen, vi kan därför förvänta oss att antalet kulor i urnan inte är samma utan att det finns alltid mer utav en färg. Vi plottar antalet röda bollar mot totala antalet bollar i urnan 5 gånger efter 1000 dragningar:

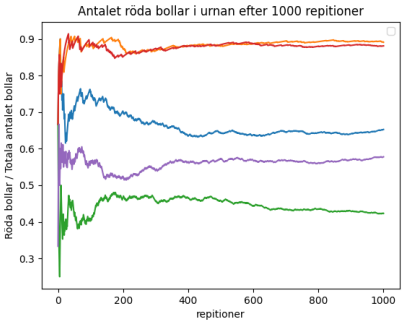


Figure 1: Antalet röda bollar mot totala antalet bollar simulerat 5 gånger efter 1000 dragninar

## 2

Vi låter $[n_0, m_0]$ vara vår startvektor och $\begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$ våran övergångsmatris. För varje röd boll vi plockar upp adderar vi 2 blåa bollar och för varje blå boll vi plockar upp adderar vi 2 röda bollar. Om $n_0 = m_0$ kan vi förvänta oss att antalet blåa och röda bollar är lika eftersom sannolikheten att dra en röd boll är ungefär lika stor som sannolikheten att dra en blå boll. Om vi plottar antalet röda bollar mot totala antalet bollar 5 gånger efter 1000 dragningar kan vi förvänta oss att antalet röda bollar närmar sig $\frac{1}{2}$. Om vi varier våra startförutsättningar så att $n_0 \neq m_0$ och låter det finnas fler röda bollar än blåa bollar får vi att utvecklingen av antalet röda bollar kommer vara större än antalet blåa bollar men om vi har ett stort tidssteg kommer till slut utvecklingen av röda och blåa bollar likna varandra. Vi låter vår startvektor vara $[10, 0]$ och plottar antalet röda bollar mot totala antalet bollar 5 gånger efter 10 dragningar.

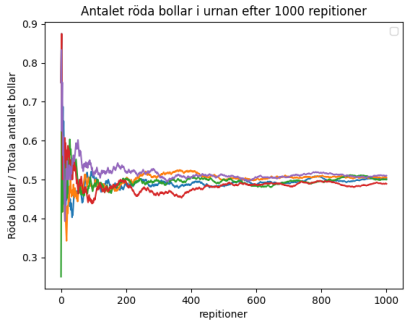


Figure 2: Antalet röda bollar mot totala antalet bollar simulerat 5 gånger med 1000 dragninar.

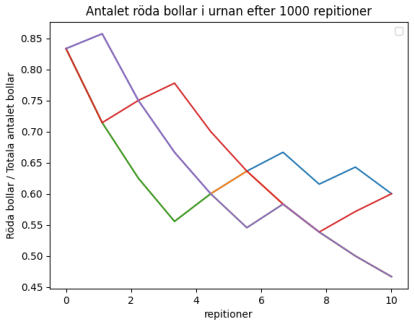


Figure 3: Antalet röda bollar mot totala antalet bollar simulerat 5 gånger med 10 dragninar

## 3

För att ta reda på hur många dragningar det krävs tills det bara finns röda bollar kvar kan vi modifiera våran kod från övning 1). Om vi väljer $[0, m_0]$ till $[0, 25]$ och kör programmet 100 gånger och tar snittet får vi att det tar 93 dragningar innan alla bollar är röda.

```python
import numpy as np

def ball_pickup(red, blue):
    total_balls = sum([red, blue])
    probability_red = red / total_balls
    probability_blue = blue / total_balls


    chosen_ball = np.random.choice(['red', 'blue'], p=[probability_red,
    ↪  probability_blue])

    return chosen_ball

def poly_urn(start_vector, transition_matrix):
    current_vector = start_vector
    counter = 0
    while current_vector[1]!=0:
        ball = ball_pickup(current_vector[0], current_vector[1])
        if ball == 'red':
            current_vector = current_vector + transition_matrix[0] -[1,0]
            counter += 1
        elif ball == 'blue':
            current_vector = current_vector + transition_matrix[1] -[0,1]
            counter +=1

    return current_vector, counter
start_vector = np.array([0,25])
transition_matrix = np.array([[1,0],
                              [1,0]])
lista = []
for i in range(101):
    a = poly_urn(start_vector, transition_matrix)[1]
    lista.append(a/100)

print(int(sum(lista)))
```

## 4

Vi har urna 1 med övergångsmatris $\begin{bmatrix} 6 & 1 \\ 0 & 2 \end{bmatrix}$ och urna 2 med övergångsmatris $\begin{bmatrix} 6 & 1 \\ 0 & 1 \end{bmatrix}$ samt att både urnorna har startvektorn $[1, 1]$. Då antalet blå bollar i urna 1 alltid ökar med 1 oavsett vilken boll vi drar så kommer den att växa snabbare än antalet blåa bollar i urna 2 som bara växer med 1 ifall vi drar en röd boll. Vi kan se detta om vi plottar antalet blåa bollar mot totala antalet bollar i båda urnorna mot varandra:

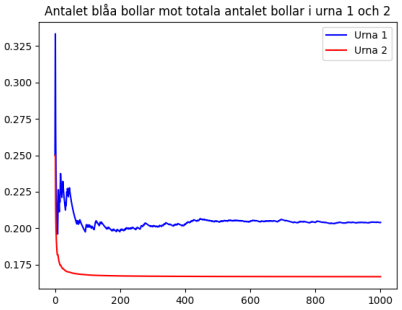


Figure 4: Antalet blåa bollar mot totala placeins antalet bollar i urna 1 och 2.

## Övning 3

Processen kan simuleras med hjälp av följande kod. Vi simulerar processen 1000 gånger och plottar ett historgam. Då vi lägger tillbaka en lapp utan sträck varje gång vi drar en lapp kan vi anta att det alltid kommer finnas flest av tomma lappar efter n dragningar. Därför kan en bra approximation för lappen med flest streck vara $\alpha_n \approx log(n)$. Om vi sedan kör denna simulering 100 gånger får vi att lappen med flest streck är $\approx log(n) + 3$

```python
import numpy as np
import matplotlib.pyplot as plt


def paper_pickup(lista):
    random_number = random.randint(0,len(lista)-1)
    random_paper = lista[random_number]
    lista.remove(random_paper)
    lista.append(random_paper + np.array([1]))
    lista.append([0])



def paper_urn(steps):
    lista = ([[0]])
    for i in range(steps):
        paper_pickup(lista)
    return lista
lista = []
for i in range(100):
    b = paper_urn(1000)
    flattened_b = [item[0] for item in b]
    c = max(flattened_b)
    lista.append(c)


d = sum(lista) / 100
print(d)
```

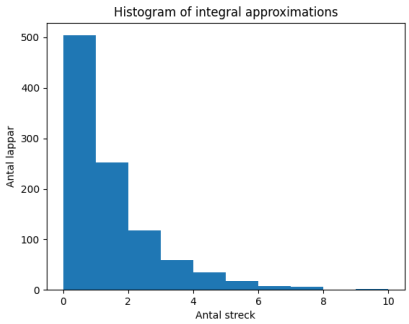


Figure 5: Histogram över antalet lappar med streck

## Övning 4

Dessa kvantiter kan räknas ut med hjälp av följande kod. Om vi låter n och k var små säg n = 3 och k = 2, n = 4 och k =2 kan vi för hand räkna ut hur många olika sätt det finns för att placera ut personerna i bussarna. Då n = 3 och k = 2 kan vi för hand räkna att det finns 4 olika sätt att placera ut personerna i bussarna och 2 sätt att placera personerna så att en buss bara innehåller en

person förutsatt att alla personer som kan sätta sig på en buss sätter sig på en buss. Då n = 4 och k =2 finns det 5 sätt att placera personerna på bussarna och 2 sätt att placera personerna så att en buss endast innehåller en person. Vi får även samma resultat om vi köra programmet. Vi testar sedan att köra programmet för stora värden på n och k. Om vi väljer n = 100 och k = 50 och kör programmet ser vi att sättet att placera personer så att en person sitter ensam på en buss närmar sig det total mängden av sätt man kan placera personerna på bussarna.

```
import numpy as np
from math import comb

def total_ways(n,k):
    return comb(n + k - 1, k -1)
def ways_buss_contain_atleast_one_person(n, k):
    total = total_ways(n, k)
    without_single_person = total_ways(n - k, k)
    return total - without_single_person
print(ways_buss_contain_atleast_one_person(3, 2))
print(total_ways(3,2))
print(ways_buss_contain_atleast_one_person(4, 2))
print(total_ways(4,2))
print(ways_buss_contain_atleast_one_person(100, 50))
print(total_ways(100,50))
```

## Övning 5

Intutivt känns det som att sannolikheten att någon av bussarna innehåller en ensam person kommer att öka med om k ökar och n är fixerat. Men vårna modell är endast definerad då $n > k$. Om vi kör programmet för n = 100 och simluerar för $0 < k < 100$ och printar en lista för alla sannolikheter ser vi att sannolikheterna närmar sig 1 då k ökar.

```
import numpy as np
from math import comb

def total_ways(n,k):
    return comb(n + k - 1, k -1)
def ways_buss_contain_atleast_one_person(n, k):
    total = total_ways(n, k)
    without_single_person = total_ways(n - k, k)
    return total - without_single_person

n = 100
lista = []
for i in range(1,100):
    probability = ways_buss_contain_atleast_one_person(n, i) / total_ways(n,i)
    lista.append(probability)
print(lista)

```