I- Problem

The goal of this machine learning lab was to work on hidden markov models.
Markov models introduces the notion of time, arguing that where we want to go à t+1 depends only of where wz currently are (t).
As the name can suggest, the states Z are not visible to the observer, and only the output x (=sequence) is visible.

The problem that we needed to solve is directly in conjunction with what we call **"3 computational problems of HMMs"**

- **Evaluation problem:** What is the likelihood that my model, represented by some parameters that we will discuss later on, generated a particular sequence?
- **Decoding problem:** What is the single most likely sequence of states to have generated that sequence of observations?
- **HMM learning problem:** If the model is parametrized by some parameters, we need to train the model, learn the parameters and pick the ones that maximize the likelihood of getting that observation.

II- Methods

1) Parameters

Before going further in our report, we think that it is really important to explain the different parameters we used and especially what they represent:

- **Z_j**: hidden states that can take different values {z0, z1, z2, z3, z4}
- **$a_{(i,j)}$**: transition matrix that represents the probability to switch from state Zi at t to the state Zj at t+1.

$$\{a_{ij}\} = \begin{matrix} \\ z0 \\ z1 \\ z2 \\ z3 \end{matrix} \begin{matrix} \begin{matrix} z0 & z1 & z2 & z3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.2 & 0.3 & 0.1 & 0.4 \\ 0.2 & 0.5 & 0.2 & 0.1 \\ 0.7 & 0.1 & 0.1 & 0.1 \end{bmatrix} \end{matrix},$$

- **$b_{(j,k)}$**: emission probabilities that represents the probability that the sequence of observable states x is generated from the sequence of hidden state Zj.

$$\{b_{jk}\} = \begin{matrix} \\ z0 \\ z1 \\ z2 \\ z3 \end{matrix} \begin{matrix} \begin{matrix} x0 & x1 & x2 & x3 & x4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 0.4 & 0.1 & 0.2 \\ 0 & 0.1 & 0.1 & 0.7 & 0.1 \\ 0 & 0.5 & 0.2 & 0.1 & 0.2 \end{bmatrix} \end{matrix}.$$

- the sequence **x**

$$x = \{x_1, x_3, x_2, x_0\}$$

- **ßi(t)**: resukt from the HMM backward algorithm that we will introduce in the next part.

- **αj(t)**: result from the HMM forward algorithm that we will introduce in the next part

2) Evolution

In order to solve the evolution problem, so to find the probability to get exactly the same sequence of observables data (it means that x(t)=x(k)), we needed to implement the forward algorithm.
Basically, the algorithm needs 3 parameters to perform well:

- $a_{(i,j)}$
- $b_{(j,k)}$
- the sequence x

Because our model specifies that Z(0)=Z1, meaning that we have 100% chance of starting at state Z1, we can deduce that α1(0)=1, allowing us to start our algorithm.

We start the forward algorithm with=1, because we do not consider the initialization (anyway, the probability to get Z1 at t=0 is about 100%).
Then, the value each αj(t) equals to the sum of values we got from αi(t-1) times its transition probability times its emission probability.
Basically, we pretty much used the algorithm available in the course:

Algorithm 1: The HMM forward algorithm

Given: $\{a_{ij}\}$, $\{b_{jk}\}$, x;
Initialize: $\forall j,\ \alpha_j(0)$;
for $t = 1 : T$ **do**
  for $j = 0 : N$ **do**
    $\alpha_j(t) \leftarrow b_{jkx(t)} \sum_{i=0}^{N} \alpha_i(t-1) a_{ij}$;
$P(x|M) \leftarrow \alpha_0(T)$;
return $P(x|M)$;

3) Decoding

In this step, the goal was to determine **the path of hidden states**, thanks to our α. So, thanks to the forward algorithm (that computes α), we can determine the sequence of our hidden states that is the most probable.
Basically, our method involves crossing our α for each 't', and for each 't' taking only into consideration the hidden state associated to the highest probability at time 't'.
By doing that for each 't' of our sequence, we are actually selecting the most probable path, so finally the most probable sequence.
This result depends on our model defined by the transition matrix A and the emission probabilities b.

```
def decoding(A, b, alpha):
    hidden_sequence_probable=np.array([0, 0, 0, 0, 0])
    tmp=0 #variable that takes the maximum of each columns
    for t in range(0,alpha.shape[1]):
        for i in range(0,alpha.shape[0]):
            if alpha[i][t]>tmp:
                tmp=alpha[i][t]
                hidden_sequence_probable[t]=i #stores the index of the maximum
        tmp=0
    return hidden_sequence_probable
```

4) Learning

The learning part was especially hard to achieve. Indeed, we had troubles. By using the formulas available on the course, we noticed that we got some errors about the factor when we tried to compute the new a. Actually, sometimes it was equal to 0, provoking an error with the value “nan”. To overpass this issue, we actually divided the numerator only if the denominator was different from 0.
Then the strategy of that part was basically the same as all the king of E-M algorithms.

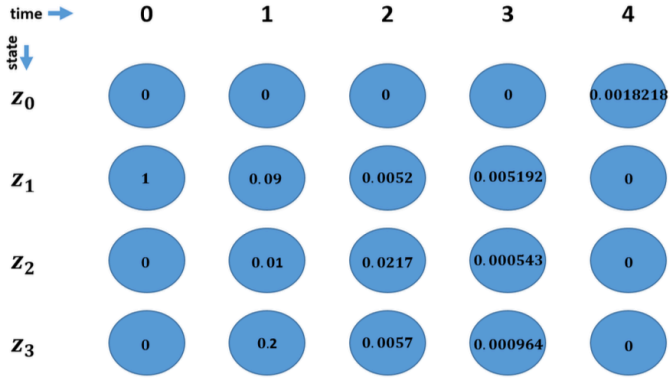
III- Results & discussion

1) Evolution

```
HMM forward diagram- Evolution:
[[  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.82180000e-03]
 [  1.00000000e+00   9.00000000e-02   5.20000000e-03   5.19200000e-03
    0.00000000e+00]
 [  0.00000000e+00   1.00000000e-02   2.17000000e-02   5.43000000e-04
    0.00000000e+00]
 [  0.00000000e+00   2.00000000e-01   5.70000000e-03   9.64000000e-04
    0.00000000e+00]]
```

Finally, thanks to our forward algorithm, we can see that using **our model** with its parameters, the probability to get the visible sequence x={x1, x3, x2, x0} is **0.0018218**.

The screen above shows our matrix **αj(t),** that can be represented by:



If we want to compute that visible sequence, we can largely conclude that finally that model is definitively not the best one. The parameters A (transition probabilities) and b (emission probabilities) are not optimized.
If we really want to improve our model with a higher probability, we must learn our model. The best way to compute that is to develop an E-M algorithm (Expectation-Maximization). In the Hidden Markov Model case, we can use the **Baum-Welch algorithm.**

2) Decoding

Now that we know what is the probability to get the visible sequence defined, we can decode the problem by finding the most probable hidden state of our model.

```
The most probable hidden states of our model:
Z1 Z3 Z2 Z1 Z0
```

So, in our case the path is: Z1 -> Z3 -> Z2 -> Z1 -> Z0

3) Learning

In this part, we had some issues for implementing the Baum-Welch algorithm. Indeed, we had troubles implementing particularly the new $\mathbf{b}_{(j,k)}$.
Actually, we got a different path from the one we computed before. Therefore, here is our path: Z1-> Z1 -> Z2->Z0
However, after having computed the algorithm 5 times, we have the sums of each lines of $\mathbf{a}_{(i,j)}$ and $\mathbf{b}_{(j,k)}$ that are equals to 1. So, the algorithms are respected.

```
new A
[[ 1.      0.      0.      0.    ]
 [ 0.499   0.002   0.02    0.479]
 [ 0.      0.965   0.034   0.    ]
 [ 0.      0.002   0.997   0.001]]
```

```
new B
[[ 1.      0.      0.      0.      0.    ]
 [ 0.      0.499   0.499   0.      0.002]
 [ 0.      0.      0.      0.965   0.035]
 [ 0.      0.      0.      0.002   0.998]]
```