```
1
2   # coding: utf-8
3
4   # In[1]:
5
6
7   import numpy as np
8
9
10  # # 1. *Génération d'un processus caché et d'un processus observable*
11  #     Cette fonction retourne, dans l'ordre:
12  #     0)Le vecteur de probabilités des états cachés générés X
13  #     1)Le vecteur des états cachés générés X (Sain ou Fievreux)
14  #     2)La version numérique du vecteur des états cachés X (0 -> Sain et 1 ->
    Fievreux)
15  #     3)Le vecteur de probabilités des états observables Y
16  #     4)Le vecteur des états observables générés Y (Etourdi, Rhume, ou Normal)
17  #     5)La version numérique du vecteur des états observables Y  0(-> Etourdi, 1 ->
    Rhume, 2->Normal)
18
19  #
20
21  # In[2]:
22
23
24  def generation(n):
25      #Génération d'un processus caché composé de n états
26      #0 -> Sain et 1 -> Fievreux
27      prbX=np.random.choice(10,n)
28      initial=prbX[0]
29      etatsX=[]
30      etatsXnum=[]
31      if initial<6:
32          etatsX.append("Sain")
33          etatsXnum.append(0)
34      else:
35          etatsX.append("Fievreux")
36          etatsXnum.append(1)
37      for i in range(1,n):
38          if (etatsX[i-1]=="Sain"):
39              if prbX[i]<7:
40                  etatsX.append("Sain")
41                  etatsXnum.append(0)
42              else:
43                  etatsX.append("Fievreux")
44                  etatsXnum.append(1)
45          else:
46              if prbX[i]<6:
47                  etatsX.append("Sain")
48                  etatsXnum.append(0)
49
50              else:
51                  etatsX.append("Fievreux")
52                  etatsXnum.append(1)
53      #Génération du processus observable composé de n états
54      #0-> Etourdi, 1 -> Rhume, 2->Normal
55      prbY=np.random.choice(10,n)
56      etatsY=[]
57      etatsYnum=[]
58      for i in range(n):
59          if (etatsX[i]=="Sain"):
60              if prbY[i]<1:
61                  etatsY.append("Etourdi")
62                  etatsYnum.append(0)
63              elif prbY[i]<5:
64                  etatsY.append("Rhume")
65                  etatsYnum.append(1)
66              else:
67                  etatsY.append("Normal")
68                  etatsYnum.append(2)
69          else:
70              if prbY[i]<1:
```

```
                etatsY.append("Normal")
                etatsYnum.append(2)
            elif prbY[i]<4:
                etatsY.append("Rhume")
                etatsYnum.append(1)
            else:
                etatsY.append("Etourdi")
                etatsYnum.append(0)
    print("Les états cachés X :")
    print(etatsX)
    print("\nLes états observables Y :")
    print(etatsY)
    return(prbX,etatsX,etatsXnum,prbY,etatsY,etatsYnum)


# ## Création des matrices psi, Q et rho
# * psi est la matrice d'émission (lien entre le domaine caché et le domaine observable)
# * Q est la matrice de transition (passage d'un état caché x à l'état suivant x'
# * rho est le vecteur de la loi initiale, c'est la probabilité de l'état x0

# In[3]:


Q=np.array([[0.7,0.3],[0.4,0.6]])
psi=np.array([[0.1,0.4,0.5],[0.6,0.3,0.1]])
rho=[0.6,0.4]


# # 2. *Implémentation de l'algorithme "Forward"*
# On cherche à connaître la loi de vraisemblance de l'ensemble des états observables jusqu'au temps k. On a seulement besoin du vecteur Y de 0 à k. L'indexation commençant à 0, on doit donner en paramètre un k strictement inférieur à n (car il y a exactement n états dans le vecteur)

# In[27]:


def algoForward(k,etatsYnum):
    #on a besoin d'une matrice de (k+1) lignes car on doit calculer les probabilités de 0 à k
    alpha=np.eye(k+1,2)
    def algoForwardRec(k,alpha):
        #n -> nombre d'états de la chaîne
        #k -> étape de l'algo (0<k<n)
        if (k>0):
            #on stocke dans une variable la valeur obtenue par récursivité
            tmp=algoForwardRec(k-1,alpha)[k-1]
            for x in range(2):
                #xprime=0
                a1=Q[0,x]*tmp[0]
                #xprime = 1
                a2=Q[1,x]*tmp[1]
                alpha[k,x]=round(psi[x,etatsYnum[k]]*(a1+a2),5)
            return(alpha)
        else:
            #lorque k=0
            alpha[0,0]=round(rho[0]*psi[0,etatsYnum[0]],5)
            alpha[0,1]=round(rho[1]*psi[1,etatsYnum[0]],5)
            return(alpha)
    alpha=algoForwardRec(k,alpha)
    return(alpha)


# In[30]:


def vraisemblance(alpha,k):
    return(sum(alpha[k,]))


# # 3. *Résolution du problème de filtrage*
```

```
# In[5]:


def filtrage(k,etatsYnum):
    alpha=algoForward(k,etatsYnum)[k]
    #initialisation de la matrice
    denom=alpha[0]+alpha[1]
    filtrage=alpha/denom
    return (filtrage)


# # 4. *Implémentation de l'algorithme "Backward"*

# In[6]:


def algoBackward(k,etatsYnum):
    #Création d'une matrice de (k+1) lignes et 2 colonnes
    beta=np.eye(k+1,2)
    def algoBackwardRec(l,beta,k):
        #n -> nombre d'états de la chaîne
        if (l<k):
            #on stocke dans une variable la valeur obtenue par récursivité
            tmp=algoBackwardRec(l+1,beta,k)[(l+1),]
            for x in range(2):
                #xprime = 0
                b0=Q[x,0]*psi[0,etatsYnum[l+1]]*tmp[0]
                #xprime = 1
                b1=Q[x,1]*psi[1,etatsYnum[l+1]]*tmp[1]
                beta[l,x]=b0+b1
            return(beta)
        else:
            #lorsque l=k
            beta[l,]=[1,1]
            return(beta)
    beta=algoBackwardRec(0,beta,k)
    return(beta)


# # 5. *Résolution du problème de lissage*

# In[41]:


def lissage(l,etatsYnum):
   #on récupère les dernières lignes des matrices forward et backward
   alpha=algoForward(l,etatsYnum)[l,]
   beta=algoBackward(l+1,etatsYnum)[l,]
   denom=alpha[0]*beta[0]+alpha[1]*beta[1]
   return((alpha*beta)/denom)


# # 6. *Implémentation de l'algorithme de Viterbi*

# In[37]:


def viterbi(k,etatsYnum):
    #Création d'une matrice de (k+1) lignes et 2 colonnes
    w=np.eye(k+1,2)
    def ProgDynaRec(k,w):
        #permet de calculer w(0:k)
        #n -> nombre d'états de la chaîne
        #k -> étape de l'algo (0<k<n-1)
        if (k>0):
           tmp=ProgDynaRec(k-1,w)[k-1]
           for x in range(2):
               m=max(Q[0,x]*tmp[0],(Q[1,x]*tmp[1]))
               w[k,x]=m*psi[x,etatsYnum[k]]
           return(w)
        else:
```

```
            #lorsque k=0
            w[0,0]=round(rho[0]*psi[0,etatsYnum[0]],5)
            w[0,1]=round(rho[1]*psi[1,etatsYnum[0]],5)
            return(w)
    w=ProgDynaRec(k,w)
    #l'estimateur xchapeauk avec le k fixé
    xchapeauk=np.argmax(w[k,])
    xchapeaul=np.eye(k+1,1)
    def viterbiRec(l,xchapeaul):
        if (l<k):
            tmp=int(viterbiRec(l+1,xchapeaul)[l+1])
            #x=0
            c0=Q[0,tmp]*w[l,0]
            #x=1
            c1=Q[1,tmp]*w[l,1]
            argsup=[c0,c1]
            xchapeaul[l]=np.argmax(argsup)
            return(xchapeaul)
        else:
            #on s'arrête quand on a l=k
            xchapeaul[l]=xchapeauk
            return(xchapeaul)
    xchapeaul=viterbiRec(0,xchapeaul)
    return(xchapeaul)


# # 7. *Tests des algorithmes*

# In[43]:


#On peut créer un processus aléatoire avec la fonction de génération
gen=generation(10)
etatsYnum=gen[5]
print("\nLes états observables en version numérique :")
print(etatsYnum)
print("\nRésultats de l'algorithme forward :")
print(algoForward(9,etatsYnum))
print("\nLa vraisemblance est :")
print(vraisemblance(algoForward(9,etatsYnum),9))
print("\nRésolution du problème de filtrage :")
print(filtrage(9,etatsYnum))
print("\nRésultats de l'algorithme backward :")
print(algoBackward(9,etatsYnum))
print("\nRésolution du problème de lissage pour l'état X4 :")
print(lissage(4,etatsYnum))
print("\nRésultats de l'algorithme de Viterbi :")
print(viterbi(9,etatsYnum))


# In[42]:


#Mais on peut également tester les algorithmes avec un vecteur qu'on a créé
etatsYnum=[2,1,0]
print("Les états observables en version numérique :")
print(etatsYnum)
print("\nRésultats de l'algorithme forward :")
print(algoForward(2,etatsYnum))
print("\nLa vraisemblance est :")
print(vraisemblance(algoForward(2,etatsYnum),2))
print("\nRésolution du problème de filtrage :")
print(filtrage(2,etatsYnum))
print("\nRésultats de l'algorithme backward :")
print(algoBackward(2,etatsYnum))
print("\nRésolution du problème de lissage pour l'état X1 :")
print(lissage(1,etatsYnum))
print("\nRésultats de l'algorithme de Viterbi :")
print(viterbi(2,etatsYnum))
```