1 Correction du TD ??, page ??

On revient ici sur les questions 7 et 8.

7)

S'il y a n femmes autour de la table, il est au plus possible d'insérer deux hommes entre deux femmes, soit $2n$ hommes (FHH FHH FHH ...) . A l'inverse, il est au plus possible d'avoir deux femmes entre deux hommes (HFF HFF HFF ...), soit $\frac{n}{2}$ si n est pair ou $\frac{n+1}{2}$ si n est impair. Il ne faut oublier pour ce nombre minimum que la table est ronde.

8)

1.1 Comment observer un problème à l'aide d'un programme informatique

On suppose maintenant qu'on ne connaît plus la personne assise à chaque place de la table, on sait que la probabilité que ce soit une femme ou une homme est égale à $\frac{1}{2}$. On se propose dans un premier de répondre à la même question mais sans tenir compte que la table est ronde, c'est-à-dire sans tenir compte des extrémités.

Le premier problème est de savoir s'il est plus facile de déterminer la probabilité que la table vérifie les conditions d'Harmonie ou celle qu'elle ne les vérifie pas, c'est-à-dire la probabilité d'avoir trois hommes ou trois femmes consécutifs. Afin de répondre à cette question, on écrit un programme informatique comptant le nombre de dispositions bonnes ou mauvaises en fonction de la taille de la table.

```
def trois_pareil (table) :
    """table est une suite booleens, on retourne True
    s'il n'y en a pas trois consecutifs"""
    for i in xrange (len (table)-2) :
        if table [i] == table [i+1] == table [i+2] :
            return False
    return True

def compte (n) :
    """on parcours toutes les tables possibles et on compte les
    bonnes configurations"""
    table = [ False for i in range (0,n) ]
    nb = 0
    bon = 0
    while table [0] != None :
        if n == 5 and trois_pareil (table) : print table
        nb += 1
        if trois_pareil (table) : bon += 1
        for i in xrange (len (table)-1, -1, -1) :
            if not table [i] :
                table [i] = True
                break
            else :
                table [i] = False
                if i == 0 : table [0] = None
```

```
    return nb, bon

for n in range (3, 15) :
    nb, bon = compte (n)
    print n, " nb = ", nb, "  bon = ", bon, " mauvais = ",
    print " proba = ",
    print float (bon) / float (nb), "          \t",
    print bon, "/", nb
```

Le programme permet d'obtenir les résultats suivants :

n	**dispositions**	**bonnes**	**mauvaises**
3	8	6	2
4	16	10	6
5	32	16	16
6	64	26	38
7	128	42	86
8	256	68	188
9	512	110	402
10	1024	178	846
11	2048	288	1760
12	4096	466	3630
13	8192	754	7438
14	16384	1220	15164

La suite des mauvaises dispositions ne semble pas obéir à une règle particulière tandis que la suite des bonnes dispositions est une suite de Fibonacci. Si on note u_n le nombre de bonnes dispositions, alors cette suite vérifie :

$$\left\{ \begin{array}{lcl} u_3 & = & 6 \\ u_4 & = & 10 \\ u_n & = & u_{n-1} + u_{n-2} \ \ \forall n \geqslant 5 \end{array} \right. \tag{1.1}$$

Il reste à comprendre pourquoi cette suite vérifie cette propriété. Il faudrait dénombrer toutes les possibilités de continuer une table de sortir qu'elle vérifie les contraintes d'Harmonie. On suppose alors qu'on dispose d'une table de longueur n se terminant par une femme, on essaye de la continuer :

...F FH
...F HF
...F HH

Ce schéma est exactement l'opposé si la table se termine par un homme. On serait tenté de dire que : $n_{n+2} = 6 * \frac{u_n}{2} = 3u_n$. Mais ce n'est pas ce qu'on observe. Etant donné qu'on a pas compté de dispositions en double ni oublié, on a forcément compté de fausses dispositions. L'examen des bonnes dispositions pour $n = 5$ nous montre l'erreur :

```
[False, False, True, False, False]  .... groupe de 3
[False, False, True, False, True]   .... conforme au raisonnement
```

```
[False, False, True, True, False]

[False, True, False, False, True]  .... groupe de 3 attendu
[False, True, False, True, False]  .... conforme au raisonnement
[False, True, False, True, True]

[False, True, True, False, False]  .... groupe de 2 non attendu
[False, True, True, False, True]   .... non conforme au raisonnement

[True, False, False, True, False]
[True, False, False, True, True]

[True, False, True, False, False]
[True, False, True, False, True]
[True, False, True, True, False]

[True, True, False, False, True]
[True, True, False, True, False]
[True, True, False, True, True]
```

Il est impossible d'écrire de continuer une table $...F\,FH$ si la table est en fait $...FF\,FH$. Les n premières places de cette table suivent les contraintes mais pas sa $n+1$ place. Autrement dit, il n'est pas possible de faire intervenir une récurrence en se servant uniquement de n et $n+2$. Il faut donc prolonger une table de longueur n et une autre de longueur $n+1$ en prenant soin qu'aucune configuration ne soit identique à une autre.

Si une table se termine par une femme, on est sûr que si on la continue avec un ou deux hommes, elle vérifiera toujours les contraintes. On applique la règle opposée si la table se termine par un homme.

table de longueur n	...F	H	H	il y en a $\frac{u_n}{2}$
table de longueur n	...H	F	F	il y en a $\frac{u_n}{2}$
table de longueur $n+1$		...F	H	il y en a $\frac{u_{n+1}}{2}$
table de longueur $n+1$		...H	F	il y en a $\frac{u_{n+1}}{2}$

Pour tout n, u_n est forcément pair puisque si une disposition est bonne, en inversant homme et femme, cette autre disposition est aussi bonne. On peut donc diviser u_n par 2. On vérifie bien qu'aucune disposition n'est identique à une autre. Il faut maintenant vérifier qu'on n'en a pas oublié[1]. Pour répondre à cette question, on retourne chacune des tables.

H	H	F...
F	F	H...
H	F...	F ou H...
F	H...	F ou H...

Il n'est pas possible de trouver un autre début pour une table vérifiant les contraintes d'Harmonie. On a donc bien dénombrer toutes les manières possibles de terminer une table de longueur $n+2$. La suite (u_n) des bonnes dispositions vérifie donc bien :

[1] Apparemment oui, car la somme des dispositions trouvées est égale au résultat escompté.

$$\begin{cases} u_3 &= 6 \\ u_4 &= 10 \\ u_n &= u_{n-1} + u_{n-2} \ \ \forall n \geqslant 5 \end{cases} \tag{1.2}$$

On peut aussi exprimer u_n en fonction de n seulement :

$$u_n = A\left(\frac{1-\sqrt{5}}{2}\right)^{n-3} + B\left(\frac{1+\sqrt{5}}{2}\right)^{n-3} \tag{1.3}$$

Avec :

$$\begin{cases} A + B &= 6 \\ A\left(\frac{1-\sqrt{5}}{2}\right) + B\left(\frac{1+\sqrt{5}}{2}\right) &= 10 \end{cases} \tag{1.4}$$

$$\implies \begin{cases} A + B &= 6 \\ \sqrt{5}(-A+B) &= 14 \end{cases} \tag{1.5}$$

$$\implies \begin{cases} A &= -1 - 3\sqrt{5} \\ B &= 3\sqrt{5} + 7 \end{cases} \tag{1.6}$$

La probabilité P_n qu'une table droite (pas ronde) vérifie les contraintes est donc :

$$P_n = \frac{-\left(1+3\sqrt{5}\right)\left(\frac{1-\sqrt{5}}{2}\right)^{n-3} + \left(7+3\sqrt{5}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{n-3}}{2^n} \tag{1.7}$$

Il reste à répondre à la même question lorsque la table est ronde. On modifie le programme de manière à compter les bonnes tables rondes (voir à la fin de cette section).

n	**dispositions**	**bonnes droites**	**bonnes rondes**
3	8	6	6
4	16	10	6
5	32	16	10
6	64	26	20
7	128	42	28
8	256	68	46
9	512	110	78
10	1024	178	122
11	2048	288	198
12	4096	466	324
13	8192	754	520
14	16384	1220	842

Prolonger la table n'est plus aussi évident qu'avant. Il y a les bonnes configurations :

table de longueur n	F	...	...	F	H	H	il y en a α_n
table de longueur n	H	...	...	H	F	F	il y en a α_n
table de longueur $n+1$	H	F	...	...	F	H	il y en a β_n
table de longueur $n+1$	F	H	...	...	H	F	il y en a β_n
table de longueur $n+1$	F	H	...	...	F	H	il y en a γ_n
table de longueur $n+1$	H	F	...	...	H	F	il y en a γ_n
table de longueur $n+1$	F	F	...	...	F	H	il y en a δ_n
table de longueur $n+1$	H	H	...	...	H	F	il y en a δ_n

Et les mauvaises :

table de longueur n	H	...	...	F	H	H	il y en a η_n
table de longueur n	F	...	...	H	F	F	il y en a η_n
table de longueur $n+1$	H	H	...	...	F	H	il y en a ζ_n
table de longueur $n+1$	F	F	...	...	H	F	il y en a ζ_n

On pourrait chercher à estimer les nombres α_n, β_n, ... Mais cette approche paraît plutôt fastidieuse, d'autant plus qu'on ne sait pas si on aura la chance de retomber sur une suite de Fibonacci. Peut-être vaut-il mieux aborder le problème selon un angle différent et avec des mathématiques différentes.

1.2 Chaînes de Markov

Les chaînes de Markov sont souvent utilisées pour étudier des séquences de nombres, des séquences d'ADN. On peut calculer la probabilité d'une lettre dans une séquence d'ADN. On peut aussi calculer la probabilité d'un couple de lettres consécutives, d'un triplet, ... En langue anglaise, c'est souvent l'association d'un verbe et de sa préposition qui ont du sens et non l'un ou l'autre pris séparément.

On considère tout d'abord une table droite de sept personnes dont on ne retient que les genres des prénoms : HFFHHHF. Et au lieu de la représenter comme une succession de lettres, on la représente comme une succession de cinq triplets de lettres :

HFF
FFH
FHH
HHH
HHF

Dans cette séquence de triplets, une table d'Harmonie ne doit pas contenir un seul triplet HHH ou FFF. L'objectif est donc de calculer le nombre de tables ne contenant pas ces deux triplets. L'inconvénient maintenant est qu'il n'est pas possible d'avoir n'importe quel triplet après un autre : il faut que les deux dernières lettres du premier soient égales aux deux premières lettres du suivant. On note t_1 le premier triplet, t_2 le second triplet et $\mathbb{P}(t_2 \mid t_1)$ la probabilité que t_2 suivent t_1. Par exemple :

$$\begin{array}{ll} \mathbb{P}(t_2 = FFF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = FFH \mid t_1 = FFH) = 0 \\ \mathbb{P}(t_2 = FHF \mid t_1 = FFH) = \frac{1}{2} & \mathbb{P}(t_2 = FHH \mid t_1 = FFH) = \frac{1}{2} \\ \mathbb{P}(t_2 = HFF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = FHH \mid t_1 = FFH) = 0 \\ \mathbb{P}(t_2 = HHF \mid t_1 = FFH) = 0 & \mathbb{P}(t_2 = HHH \mid t_1 = FFH) = 0 \end{array}$$

En fait ces probabilités sont les mêmes qu'on soit au milieu de la table ou au début : elles ne dépendent pas de la position n, on les notera dans la suite : $\mathbb{P}(t_{n+1} \mid t_n)$.

On construit ensuite l'ensemble $E = \{e_i \mid 1 \leqslant i \leqslant 8\}$ où :

$$\begin{array}{ll} e_1 = HHH & e_2 = HHF \\ e_3 = HFH & e_4 = HFF \\ e_5 = FHH & e_6 = FHF \\ e_7 = FFH & e_8 = FFF \end{array}$$

On construit aussi la matrice $M = \left(\mathbb{P}\left(t_{n+1} = e_j \mid t_{n+1} = e_i\right)\right)_{\substack{1 \leqslant i \leqslant 8 \\ 1 \leqslant j \leqslant 8}}$, on obtient[2] :

$$M = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{1.8}$$

Tout l'intérêt de cette modélisation tient dans le calcul des puissances de la matrice M. A quoi correspond M^2 ? Par exemple, on note m_{ij}^d le coefficient ij de la matrice M^d, alors :

$$\begin{aligned} m_{ij}^2 &= \sum_{k=1}^{8} m_{ik}^1 \, m_{kj}^1 \\ &= \sum_{k=1}^{8} \mathbb{P}\left(t_{n+1} = e_k \mid t_{n+1} = e_i\right) \; \mathbb{P}\left(t_{n+1} = e_j \mid t_{n+1} = e_k\right) \\ &= \sum_{k=1}^{8} \mathbb{P}\left(t_{n+1} = e_j \mid t_{n+1} = e_k\right) \; \mathbb{P}\left(t_{n+1} = e_k \mid t_{n+1} = e_i\right) \\ &= \mathbb{P}\left(t_{n+2} = e_j \mid t_n = e_i\right) \end{aligned} \tag{1.9}$$

La matrice M^2 contient les probabilités d'apparition non pas du triplet suivant mais du second triplet suivant. De la même manière, on démontrer que m_{ij}^d est la probabilité d'apparition du triplet e_j à la position $n + d$ sachant que le triplet à la position n est e_i.

Est ce que cela nous permet de résoudre notre problème ? Pas encore, il reste à écrire d'une manière ou d'une autre qu'on ne doit pas rencontrer à aucune position le triplet $e_1 = HHH$ ou $e_8 = FFF$.

Il existe une autre manière d'entrevoir le problème. Tous les triplets de la table doivent être différents de e_1 et e_8, autrement dit, on doit aller du premier triplet au dernier triplet sans passer par e_1 et e_8. Imaginons que si jamais, on passe par un des ces deux triplets, on reste bloqué, comme si les triplets e_1 et e_8 étaient des culs-de-sac : une fois qu'on passe par le triplet e_1, on y reste. La probabilité de rester dans ce triplet est égale à 1. Ceci aboutit à la matrice de transition M' suivante :

[2] Afin de rendre la lecture plus facile, les "." signifie 0.

$$M' = \begin{pmatrix} 1 & . & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & . & . & . & . & . & . \\ . & . & \frac{1}{2} & \frac{1}{2} & . & . & . & . \\ . & . & . & . & \frac{1}{2} & \frac{1}{2} & . & . \\ . & . & . & . & . & . & . & 1 \end{pmatrix} \tag{1.10}$$

De la même manière que pour M, M'^d contient les probabilités d'apparition du triplet d connaissant le triplet de départ. Et si ce dernier triplet est différent de e_1 et e_8, alors il est impossible qu'on y soit passé puisque dans ce cas, on y serait resté jusqu'à la fin. Tous les triplets acceptables pour une table d'Harmonie sont dans l'ensemble $E' = \{e_2, ..., e_7\}$. Toutes les tables droites de longueur n vérifiant les contraintes d'Harmonie partent d'un triplet de E' pour arriver dans E'. La probabilité d'une table d'Harmonie de longueur n constituée des triplets $\{t_1, \ldots, t_{n-2}\}$ s'exprime comme suit :

$$\begin{aligned} P_n &= \mathbb{P}\left(t_1 \in E'\right) \mathbb{P}\left(t_{n-2} \in E' \mid t_1 \in E'\right) \\ &= \sum_{i=2}^{7} \mathbb{P}\left(t_1 = e_i\right) \left[\sum_{j=2}^{7} \mathbb{P}\left(t_{n-2} = e_j \mid t_1 = e_i\right) \right] \\ &= \sum_{i=2}^{7} \frac{1}{8} \left[\sum_{j=2}^{7} m'^{n-3}_{ij} \right] = \frac{1}{8} \sum_{i=2}^{7} \sum_{j=2}^{7} m'^{n-2}_{ij} \end{aligned} \tag{1.11}$$

Il ne reste plus qu'à en déduire la probabilité d'une table d'Harmonie ronde en introduisant la probabilité de passer du dernier triplet au premier. Par exemple, pour la séquence HFFHHFF. On avait construit pour une table droite la séquence :

HFF
FFH
FHH
HHF
HFF

Mais pour une table ronde, il faut ajouter le fait de revenir au début tout en respectant les règles de passages entre deux triplets : les deux dernières d'un triplet sont les deux premières du triplet suivant. On est alors obligé d'ajouter deux triplets qui permettent de faire la liaison entre le début et la fin.

HFF
FFH
FHH
HHF
HFF
FFH
FHF

Ensuite seulement, il est possible de passer du dernier triplet au premier, ce qui donne une probabilité P^r d'avoir une table ronde de longueur n respectant les critères d'Harmonie :

$$P_n^r = \sum_{i=2}^{7}\sum_{j=2}^{7} m_{ij}'^{n-1} m'_{ji} \tag{1.12}$$

Dans cette expression (1.12), par rapport la probabilité (1.11), le terme $\mathbb{P}(t_1 = e_i)$ puisqu'il n'y a plus de début de la table, le premier triplet dépend du dernier, ce que dit le terme m'_{ji}. De même, comme la séquence contient deux triplets de plus, on utilise les coefficients de la matrice M'^{n-1}.

L'expression obtenue n'est pas aussi aisément calculable que celle (1.7) obtenue à l'aide des suites de Fibonacci mais pour une table droite. Il faudrait pour cela obtenir la forme de la matrice M'^{n} pour tout n. Il faudrait pour cela découvrir une récurrence entre les coefficients ou alors appliquer une méthode plus systématique comme la diagonalisation de la matrice M'.

1.3 Le programme

```
1  ##############################################################################
2  #                              questions 1 a 5
3  ##############################################################################
4
5  def lit_fichier (nom) :
6      """cette fonction lit le contenu d'un fichier
7      et retourne une liste contenant ses lignes"""
8      f = open (nom, "r")
9      li = f.readlines ()
10     f.close ()
11     r =[]
12     for l in li :
13         s = l.replace ("\n", "")
14         s = s.replace ("\r", "")
15         r.append (s)
16     return r
17
18 def genre (prenom, femme, homme) :
19     """retourne le genre d'un prenom"""
20     if prenom in femme : return True
21     if prenom in homme : return False
22     return None
23
24 def contrainte (table, femme, homme) :
25     """cette fonction verifie que la table
26     obeit aux contrainte d'Harmonie :
27     il ne peut y avoir trois personnes voisines du meme sexe
28     qui se suivent"""
29     ge = [ genre (t,femme, homme) for t in table ]
30     for i in range (0,len (ge)) :
31         if ge [i] == ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
32             return False
33     return True
34
```

```
35  def dominique (table, femme, homme) :
36      """cette fonction dit si on peut placer dominique,
37      deux schemas de figures possibles :
38        1-  H F dominique H F
39        2-  F H dominique F H
40      """
41      ge = [ genre (t,femme, homme) for t in table ]
42      for i in range (0,len (ge)) :
43          if  ge [i]                != ge [ (i+1) % len (ge) ] and \
44              ge [(i+2) % len (ge)] != ge [ (i+3) % len (ge) ] and \
45              ge [i]                == ge [ (i+2) % len (ge) ] :
46              return True
47      return False
48
49  femme = lit_fichier ("femme.txt")
50  homme = lit_fichier ("homme.txt")
51  table = lit_fichier ("table.txt")
52  r = contrainte (table, femme, homme)
53  print "la table suit les desirs d'Harmonie : ", r
54  r = dominique (table, femme, homme)
55  print "peut-on ajouter Dominique : ", r
56
57
58  ###############################################################################
59  #                                question 6
60  ###############################################################################
61
62  def contrainte_dominique (table, femme, homme) :
63      """meme fonction que contrainte mais on tient aussi du fait que
64      le genre peut etre indetermine (None), toute configuration
65      incertaine est eliminee"""
66      ge = [ genre (t,femme, homme) for t in table ]
67      for i in range (0,len (ge)) :
68          if ge [i] == ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
69              return False
70          if ge [i] == None and ge [ (i+1) % len (ge) ] == ge [(i+2) % len (ge) ] :
71              return False
72          if ge [i] ==  ge [ (i+1) % len (ge) ] and ge [(i+2) % len (ge) ] == None :
73              return False
74          if ge [i] ==  ge [ (i+2) % len (ge) ] and ge [(i+1) % len (ge) ] == None :
75              return False
76          if ge [i] == None and ge [ (i+1) % len (ge) ] == None :
77              return False
78          if ge [i] == None and ge [ (i+2) % len (ge) ] == None :
79              return False
80          if ge [ (i+2) % len (ge) ] == None and ge [ (i+1) % len (ge) ] == None :
81              return False
82      return True
83
84  def place_dominique (table, femme, homme) :
```

```
85      """on cherche a inserer autant de dominique que possible"""
86      nb = len (table)
87      fin = False
88      while not fin :
89          fin = True
90          for i in range (0, len (table)) :
91              table.insert (i, "DOMINIQUE")
92              r = contrainte_dominique (table, femme, homme)
93              if r :
94                  fin = False
95                  break
96              else :
97                  del table [i]
98      return len (table) - nb
99
100 print "-------------------------------------------------------------"
101 nb = place_dominique (table, femme, homme)
102 print "nombre de dominique ", nb
103 for p in table :
104     print p
105
106
107 ###############################################################################
108 #                               question 7
109 ###############################################################################
110 #
111 #  nombre minimal : partie entiere de (n+1)/2
112 #  nombre maximal : 2n
113 #
114 ###############################################################################
115 #                               question 8
116 ###############################################################################
117
118 def trois_pareil (table) :
119     """table est une suite booleens, on retourne True
120     s'il n'y en a pas trois consecutifs"""
121     for i in xrange (len (table)-2) :
122         if table [i] == table [i+1] == table [i+2] :
123             return False
124     return True
125
126 def trois_pareil_rond (table) :
127     """table est une suite booleens, on retourne True
128     s'il n'y en a pas trois consecutifs"""
129     for i in xrange (len (table)) :
130         if table [i] == table [ (i+1) % len (table)] == table [(i+2) % len (table)] :
131             return False
132     return True
133
134 def compte (n) :
```

```
135       """on parcours toutes les tables possibles et on compte les
136       bonnes configurations"""
137       table = [ False for i in range (0,n) ]
138       nb = 0
139       bon = 0
140       while table [0] != None :
141           nb += 1
142           if trois_pareil_rond (table) : bon += 1
143           for i in xrange (len (table)-1, -1, -1) :
144               if not table [i] :
145                   table [i] = True
146                   break
147               else :
148                   table [i] = False
149                   if i == 0 : table [0] = None
150
151       return nb, bon
152
153   print "-----------------------------------------------------------------"
154   print "calcul en parcourant toutes les tables possibles"
155   print "-----------------------------------------------------------------"
156   for n in range (3, 15) :
157       nb, bon = compte (n)
158       print n, " nb = ", nb, "  bon = ", bon,
159       print " proba = ", float (bon) / float (nb), "          \t",
160       print bon, "/", nb
161
162   ###############################################################################
163   #                               question 8
164   #                                   markov
165   ###############################################################################
166   # module pour le calcul matriciel
167   import scipy_base as SCI
168   import copy
169
170
171   def construit_matrice () :
172       """construit la matrice de transition,
173       le vecteur des probabilites d'entrees dans chaque etat"""
174       # construction des etats
175       etat = []
176       for i in range (0,2) :
177           for j in range (0,2) :
178               for k in range (0,2) :
179                   etat.append ( (i,j,k) )
180       # construction de la matrice
181       mat = []
182       for i in range (0, len (etat)) :
183           l = []
184           for j in range (0, len (etat)) :
```

```
185                 if etat [i] [1:3] == etat [j] [0:2] :
186                     # (i,j,k) --> (j,k,0) ou (j,k,1)
187                     l.append (1)
188                 else : l.append (0)
189             # on renormalise sur chaque ligne
190             s = sum (l)
191             for j in range (0, len (etat)) : l[j] /= float (s)
192             mat.append (l)
193 
194         entree = [ 0.125 for i in range (0,8) ]
195         return etat, SCI.mat (mat), SCI.mat (entree)
196 
197     def calcul_probabilite_droite (n, etat, mat_, entree) :
198         """calcule les probabilites pour une table droite"""
199         pos     = [ etat.index ( (0,0,0) ), etat.index ( (1,1,1) ) ]
200         temp    = copy.deepcopy (entree)
201         mat     = copy.deepcopy (mat_)
202         for p in pos :
203             for i in range (0,8) :
204                 mat [ (p,i) ] = 0.0
205             mat [ (p,p) ] = 1.0
206 
207         m = mat ** (n-1)
208 
209         sum = 0.0
210         for i in range (0,8) :
211             for j in range (0,8) :
212                 if i not in pos and j not in pos :
213                     sum += m [ (i,j) ]  * mat [ (j,i) ]
214         return sum
215 
216 
217 
218     print "-----------------------------------------------------------------"
219     print "calcul avec les chaines de Markov"
220     print "-----------------------------------------------------------------"
221     etat,mat,entree = construit_matrice ()
222 
223     for n in range (3,15) :
224         proba = calcul_probabilite_droite (n, etat, mat, entree)
225         print n, "  bon ", proba
```

fin correction TD ?? □