

```
# td_info_051026.py
```

```
0001 ##### TD d'informatique de BCPST2B du 28 septembre 2026 #####
0002 ##### Séries, équivalent et régression linéaire #####
0003
0004 import numpy as np
0005 import random as rd
0006 import matplotlib.pyplot as plt
0007
0008 ## Exercice 1 : vérification d'un équivalent
0009 # On rappelle la méthode pour calculer une somme en informatique :
0010 # (initialement) s=0; for k in range(...): s = s+...
0011 # que l'on peut écrire avec l'instruction np.sum([liste en compréhension])
0012
0013 # On rappelle la méthode des moindres carrés (régression linéaire) pour évaluer
0014 # A et B tels que  $y \sim A \cdot x + B$  :  $A = \text{cov}(Lu, Lv) / \text{var}(Lv)$  et  $B = \text{moy}(Lu) - A \cdot \text{moy}(Lv)$ 
0015
0016 # Q1a : Vérifier que  $S_n \sim \beta \cdot n^a$  équivaut à  $\ln(S_n) - (A \cdot \ln(n) + \ln(\beta)) \rightarrow 0$ 
0017 # Q1b : écrire une fonction python renvoyant, en fonction de a et N
0018 # la liste [S1, S2, ..., SN] où  $S_n = \text{somme}\{k=1 \text{ à } n\} \text{ des } k^a$ 
0019 # la liste des  $\ln(k)$  où k décrit  $[1, n]$ 
0020
0021 def ex01b(a, n) :
0022     s, Ls = 0, []
0023     for k in range(1, n+1) :
0024         s += k**a
0025         Ls.append(s)
0026     return [np.log(s) for s in Ls], [np.log(k) for k in range(1, n+1)]
0027
0028 # Q1c : écrire une fonction moy calculant la moyenne d'une liste et une fonction
0029 # var calculant la variance (moyenne des carrés - carré de la moyenne) et une
0030 # fonction cov calculant la covariance de deux listes L1, L2 (moyenne des
0031 # produits  $L1[k] \cdot L2[k]$  - produit des moyennes)
0032
0033 def moy(L) :
0034     return np.sum(L) / len(L)
0035
0036 def var(L) :
0037     L2 = [x**2 for x in L]
0038     return moy(L2) - moy(L)**2
0039
0040 def cov(L1, L2) :
0041     Lp = [L1[k] * L2[k] for k in range(len(L1))]
0042     return moy(Lp) - moy(L1) * moy(L2)
0043
0044 # Q1d : effectuer une régression linéaire sur les listes des  $S_k$  et  $\ln(k)$  et
0045 # retrouver l'équivalence obtenue en cours par comparaison somme vs intégrale.
0046
0047 def verif(a, n) :
0048     Ls, Ln = ex01b(a, n)
0049     A = cov(Ls, Ln) / var(Ln)
0050     B = moy(Ls) - A * moy(Ln)
0051     # en cours on a trouvé  $A = a+1$  et  $\beta = 1/(a+1)$ 
0052     return A, np.exp(B)
0053
0054 # Q1e : tester la fonction verif avec a=2 et n qui vous semble pertinent.
0055
0056
0057 ## Exercice 2 : Dénombrement
0058 # En informatique, on peut dénombrer des objets en les fabriquant exactement ou
0059 # bien en fabriquant des "candidats" à tester pour sélectionner les vrais objets.
0060
0061 # Pour cela on a souvent recours à la fabrication de listes, y compris pour
0062 # compter des combinaisons, en considérant qu'il faut bien se mettre d'accord
0063 # sur une façon de les écrire (on adopte souvent l'ordre lexicographique).
0064
0065 # On peut d'abord penser à imbriquer des boucles pour parcourir des listes de
0066 # longueur 2 ou 3 : par exemple, pour parcourir les listes [x0, x1, x2] :
```

```

067 | # for x0... for x1... for x2...
068 |
069 | # Le procédé s'épuise vite et, pour parcourir ou fabriquer des listes plus
070 | # longues, on adopte un parcours d'arbre où on garde une pile (stock en cours)
071 | # de listes en cours de fabrication et on explique comment choisir un élément
072 | # de la pile et comment le prolonger d'un terme. Ensuite, on le replace dans la
073 | # pile sauf s'il est terminé, auquel cas on le compte, ou pas.
074 |
075 | # Exemple 1 :
076 | # Q2a : écrire une fonction testant tous les triplets [a,b,c] d'entiers positifs
077 | # et comptant combien sont croissants de somme égale à N (en fonction de N).
078 | def exo2a(N) :
079 |     # remarquons que les listes gagnantes sont constituées d'entiers <=N.
080 |     compteur = 0
081 |     for a in range(N) :
082 |         for b in range(N) :
083 |             for c in range(N) :
084 |                 if a<=b<=c and a+b+c==N :
085 |                     compteur += 1
086 |     return compteur
087 |
088 | # on peut proposer une variante qui stocke les listes gagnantes :
089 | def exo2abis(N) :
090 |     # remarquons que les listes gagnantes sont constituées d'entiers <=N.
091 |     Lgagn = []
092 |     for a in range(N) :
093 |         for b in range(N) :
094 |             for c in range(N) :
095 |                 if a<=b<=c and a+b+c==N :
096 |                     Lgagn.append( [a,b,c] )
097 |     return len(Lgagn),Lgagn
098 |
099 |
100 | ## Exercice 3 : Parcours
101 | # L'objectif est de parcourir "tous les cas possibles" afin de les compter
102 | # ou de compter, parmi eux, les cas dits "favorables"
103 |
104 | # Exemple 1 : parcourir toutes les permutations d'une liste L sans répétition
105 | # Le principe consiste à construire les permutations au fur et à mesure
106 | # en prolongeant à chaque fois "un peu" un déjà existante jusqu'à ce
107 | # qu'on en ait achevé que l'on compte (avec un compteur).
108 | # Pour cela, on stocke les permutations en cours de fabrication dans une
109 | # "pile"; à chaque tour, on en "dépille" une, à partir de laquelle on en
110 | # fabrique de nouvelles prolongées que l'on "repile".
111 | #
112 | # Le programme suivant est donné en guise d'illustration.
113 |
114 | def toutes_permut(L) : # on suppose L sans répétition non vide
115 |     N = len(L)
116 |     compteur = 0 # on va ici compter toutes les permutations possibles
117 |     pile = [ [x] for x in L ] # Q1 : que contient ici la pile ?
118 |     while pile != [] : # Q2 : quelle est la condition d'arrêt ?
119 |         l = pile.pop(-1) # Q3 : quel élément de la pile consulte-t-on ?
120 |         # Q4 : pourquoi utilise-t-on .pop() ?
121 |         if len(l)==N :
122 |             compteur += 1
123 |         else :
124 |             reste = [x for x in L if x not in l]
125 |             for x in reste :
126 |                 pile.append( l+[x] )
127 |     # Q5 : êtes-vous sûr que la boucle while s'arrête un jour ?
128 |     return compteur
129 |
130 | # Q3a : transformer le programme pour compter, lorsque L désigne une liste sans
131 | # répétition, toutes les listes de p éléments de L (p-arrangements de L)
132 |
133 | def tous_p_arrgmts(L,p) :
134 |     # on suppose L sans répétition non vide et 1<=p<=len(L)

```

```

135 |     compteur = 0
136 |     pile = [ [x] for x in L ]
137 |     while pile != [] :
138 |         l = pile.pop(-1)
139 |         if len(l)==p :
140 |             compteur += 1
141 |         else :
142 |             reste = [x for x in L if x not in l]
143 |             for x in reste :
144 |                 pile.append( l+[x] )
145 |     return compteur
146 |
147 | # Q3b : lorsque L est une liste non vide, sans répétition, d'entiers,
148 | #       compter tous les p-arrangements de L qui sont croissants
149 |
150 | def tous_p_arrgmts_croiss(L,p) :
151 |     # on suppose L sans répétition non vide et 1<=p<=len(L)
152 |     compteur = 0
153 |     pile = [ [x] for x in L ]
154 |     while pile != [] :
155 |         l = pile.pop(-1)
156 |         k = l[-1]
157 |         if len(l)==p :
158 |             compteur += 1
159 |         else :
160 |             reste = [x for x in L if x not in l and x>k]
161 |             for x in reste :
162 |                 pile.append( l+[x] )
163 |     return compteur
164 |
165 | ## Exercice 1bis : Voir le parcours
166 | # Au lieu de compter on va ici afficher et proposer une variante :
167 | # remplacer la ligne 25 par : l = pile.pop(0)
168 | def affiche_toutes_permut(L) : # on suppose L sans répétition non vide
169 |     N = len(L)
170 |     pile = [ [x] for x in L ]
171 |     while pile != [] :
172 |         l = pile.pop(-1)
173 |         if len(l)==N :
174 |             print(l)
175 |         else :
176 |             reste = [x for x in L if x not in l]
177 |             for x in reste :
178 |                 pile.append( l+[x] )
179 |
180 | def affiche_toutes_permut_variante(L) :
181 |     N = len(L)
182 |     pile = [ [x] for x in L ]
183 |     while pile != [] :
184 |         l = pile.pop(0)
185 |         if len(l)==N :
186 |             print(l)
187 |         else :
188 |             reste = [x for x in L if x not in l]
189 |             for x in reste :
190 |                 pile.append( l+[x] )
191 |
192 | # Q3c : tester les deux variantes et expliquer les deux résultats obtenus

```