TD d'informatique - résolution d'un problème de Sudoku

Le but de ce TD est de programmer un algorithme de résolution d'un problème de Sûdoku. On fera les tests sur le problème donné en annexe. Dans tout le programme, `i` *désigne un numéro de ligne entre 0 et 8,* `j` *un numéro de colonne entre 0 et 8, et* `nombre` *un entier entre 1 et 9.* `M` *sera la matrice que l'on traite. On créera aussi* `matrice_originale=M.copy()`*.*

1 Programmer une fonction `test_ligne(i,j,nombre,M` qui teste si le nombre `nombre` n'existe pas dans la ligne `i`, mis à part dans la colonne `j`, et qui retourne un booléen.
Tester cette fonction

2 Programmer une fonction `test_colonne(i,j,nombre,M)` qui teste si le nombre `nombre` n'existe pas dans la colonne `j`, mis à part dans la ligne `i`, et qui retourne un booléen.
Tester cette fonction

3 Programmer une fonction `test_carre(i,j,nombre,M)` qui teste si le nombre `nombre` n'existe pas dans le sous-carre auquel appartient la case `(i,j)`, mis à part dans cette case, et qui retourne un booléen. On rappelle que ce carré s'étend, pour une coordonnée, de $3X$ à $3X+2$, où X est le résultat de la division euclidienne de la coordonnée par 3.
Tester cette fonction

4 Programmer une fonction `test(i,j,nombre,M)` qui réalise la synthèse des trois tests précédents, et qui retourne un booleen.
Tester cette fonction

5 Programmer une fonction `protection(matrice_originale)`, qui retourne la liste `liste_protegee` des tuples des coordonnées des cases originellement remplies dans le problème.
Tester cette fonction et affecter sa valeur de retour à `liste_protegee`

6 Programmer une fonction `remplissage(i,j,M,liste_protegee,sens)` qui :

- si `(i,j)` est dans `liste_protegee`, retourne `sens` qui est un booléen indiquant si on était en train d'avancer (`True`) ou de reculer (`False`).
- sinon, crée `nombre=M[i,j]+1` et, tant que `nombre<10`, teste si `test(i,j,nombre,M)` est vrai.
  — si c'est le cas, la fonction affecte `nombre` à `M[i,j]` et retourne `True`
  — sinon, `nombre` est incrémenté de 1, et on refait le test
- si à l'issue du processus `nombre` atteint 10, alors aucune valeur n'était possible dans cette case. Dans ce cas :
  — on fait `M[i,j]=0`
  — on retourne `False`

Tester cette fonction avec `sens=True` **et la liste protégées créée plus tôt**

7 Programmer une fonction `avance_case(i,j)` qui retourne le tuple des coordonnées de la case suivante :

- `(i,j+1)` en général
- `(i+1,0)` si `j==8`

8 Programmer une fonction `recule_case(i,j)` qui retourne le tuple des coordonnées de la case précédente, suivant le même schéma que la fonction `avance_case(i,j)`

9 Programmer enfin la fonction `resolution(M)` qui :

- initialise les variables : `tournetourne=True`, `i=0`, `j=0`, `matrice.originale=M.copy()`,`liste_protegee=protection(matrice_originale)`, `sens=True`
- dans un `while tournetourne`
  - si `remplissage(i,j,M,liste_protegee,sens` retourne `True`, alors on a trouvé une valeur possible, et on a `(i,j)=avance_case(i,j)`, et `sens=True`
  - si `remplissage(i,j,M,liste_protegee,sens` retourne `False`, alors on a dépassé 9 sans succès, et on a `(i,j)=recule_case(i,j)`, et `sens=False`
  - si `i` atteint 9, alors le programme a dépassé la dernière case : on a résolu le problème, donc `tournetourne=False` et on affiche `M`
  - si `i` atteint -1, alors on a testé jusqu'à 9 sans succès la première case : le problème n'a pas de solution. `tournetourne=False` et on affiche que le problème n'a pas été résolu.

On peut afficher en début du `while` la matrice : ceci allonge considérablement le processus (on passe de quelques secondes à plus d'une minute) mais permet de voir comment le programme travaille.

10 Lancer le programme `resolution(M)`.

			1	2				
7						5		2
2				4	7			9
8	4		5			2		1
5		7			6		8	4
1			3	8				5
9		3						7
				5	2			

6	3	9	1	2	5	4	7	8
7	1	4	9	3	8	5	6	2
2	8	5	6	4	7	1	3	9
8	4	6	5	7	3	2	9	1
3	2	1	8	9	4	7	5	6
5	9	7	2	1	6	3	8	4
1	7	2	3	8	9	6	4	5
9	5	3	4	6	1	8	2	7
4	6	8	7	5	2	9	1	3

```
import numpy as np
M=np.zeros((9,9),int)
M[0,3]=1
M[0,4]=2
M[1,0]=7
M[1,6]=5
M[1,8]=2
M[2,0]=2
M[2,4]=4
M[2,5]=7
M[2,8]=9
M[3,0]=8
M[3,1]=4
M[3,3]=5
M[3,6]=2
M[3,8]=1
M[5,0]=5
M[5,2]=7
M[5,5]=6
M[5,7]=8
M[5,8]=4
M[6,0]=1
M[6,3]=3
M[6,4]=8
M[6,8]=5
M[7,0]=9
M[7,2]=3
M[7,8]=7
M[8,4]=5
M[8,5]=2
```