Etape 3 : Méthode de Métropolis

3.1. Il apparait que le déchiffrement réalisé à l'étape 2 ne soit pas très performant. En effet, le message est trop court pour que la fréquence des lettres dans le message ne soit représentative. Nous allons maintenant utiliser une méthode différente en regardant la **fréquence de bigrammes**, ie la **fréquence de 2 caractères consécutifs**. Nous allons créer une matrice de transition Mat dans laquelle seront stockées les fréquences du caractère i suivi du caractère j à la position Mat_{ij}.

Cette matrice de transition sera définie sous forme d'un tableau à 2 dimensions (array)

Ecrire la fonction **frequenceBigrammes** qui calcule et renvoie une matrice comportant les fréquences des bigrammes du texte passé en paramètre.

3.2. Afficher le résultat à l'aide de plt.imshow(). Vérifier que les fréquences sont cohérentes. (Pour davantage de lisibilité, vous pouvez choisir d'afficher la racine cubique de la matrice)

3.3. Grâce à la matrice de transition, nous allons définir la **mesure de plausibilité d'un message**. Soient c_1, c_2,… c_N la liste des caractères du message déchiffré. La mesure de plausibilité vaut :

$$\frac{1}{N}\sum_{i=1}^{N-1} \log Mat\ \ c_i c_{i+1}$$

Un message correspondant bien aux statistiques des bigrammes en français aura une plausibilité élevée. La méthode de Metropolis que nous allons utiliser consiste à chercher un code tel que le message déchiffré maximise la plausibilité (« algorithme de recuit-simulé »). De manière générale l'algorithme cherche une clé qui améliore la plausibilité du chiffrement en échangeant aléatoirement 2 caractères dans la clé de chiffrement courante. Si la nouvelle clé ainsi constituée améliore la plausibilité, on conserve l'échange de caractères. Sinon, on peut toujours accepter la clé avec une certaine probabilité qui dépend de l'augmentation de la plausibilité.

Ecrire la fonction **permute** qui :

- prend en paramètres les indices i et j, ainsi que la liste code
- et renvoie une liste dans laquelle les cases i et j ont été permutées.

3.4. Ecrire la fonction **calculePlausibilite** qui renvoie la plausibilité du texte passé en paramètre selon la matrice Mat elle aussi passée en paramètre.

3.5. Implémenter **la méthode de Métropolis** en vous appuyant sur l'algorithme suivant écrit en pseudo-code, afin de déchiffrer le message initial.

Vous pouvez tester avec une plausibilité de -2.1 et un nombre maximal d'itérations de 10000.


```
Metropolis
Entrées :
        Mat : matrice de transition
        texte : message à déchiffrer
        plausibiliteMax: valeur maximale de la plausibilité
        MAX_ITER  : nombre maximal d'itérations
#initialisation
codeCourant = générer code aléatoire
traductionCourante = dechiffrer texte avec codeCourant
scoreCourant = calculer la plausibilite de traductionCourante à l'aide de Mat
codeMax = copie de codeCourant
traductionMax = traductionCourante
scoreMax = scoreCourant
TANT QUE scoreMax < plausibiliteMax et nb_iter < MAX_ITER
        i,j = déterminer de manière aléatoire les 2 indices à permuter
        codeCandidat = permuter les caractères aux indices i et j de codeCourant
        traductionCandidat = dechiffrer texte avec codeCandidat
        scoreCandidat= calcul de la Plausibilite de  traductionCandidat avec  Mat
        SI scoreCourant > scoreMax ALORS
            codeCourant = copie de codeCandidat
            traductionCourante = traductionCandidat
            scoreCourant = scoreCandidat
            codeMax = copie de codeCourant
            traductionMax = traductionCourante
            scoreMax = scoreCourant
        SINON SI random() < exp( (scoreCandidat – scoreCourant) x taille(texte)) ALORS
            codeCourant = copie de )codeCandidat
            traductionCourante = traductionCandidat
            scoreCourant = scoreCandidat
        FINSI
        nb_iter = nb_iter +1
FINTANTQUE
RETOURNER codeMax, traductionMax
```