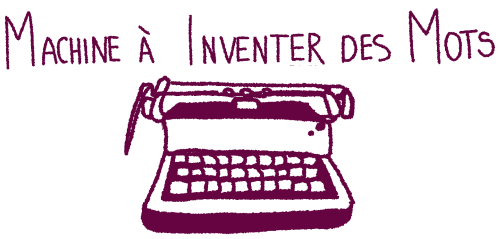


La machine à inventer des mots

Gasmi Rémy, Zenati Chloé, Zafy Karine, Braux Elie, Coron Matis
Darmon Sarah

2019-2020

L2 Double licence Maths-Physique 2019-2020
Projet Commando

Référents : Patrice Hello et Jérémy Neuveu

Université Paris-Saclay, Campus Orsay, UFR Sciences
Orsay, 91440

Table des matières

1 Introduction

En 2015, David Louapre, le présentateur de la chaîne youtube SciencesEtonnantes, décide de créer un algorithme qui permet d'inventer des mots se rapprochant énormément des mots de la langue française.
L'objectif de notre projet est de créer plusieurs algorithmes du même type qui vont nous permettre d'analyser des mots de la langue française et d'en créer de nouveaux, et de comparer leurs performances. Les banques de mots français utilisées sont d'une grande diversité afin de rendre notre étude la plus complète que possible. Leurs tailles varient de 835 à 411 430 mots, les banques les plus courtes ne visent qu'à offrir des mots courants, celles les plus longues visent à être les plus complètes possibles. Certaines banques présentent des mots déclinés, d'autres non. Leur étude comparative nous a permis de faire des statistiques sur les mots de la langue française, et de déterminer des longueurs de mot idéale et maximale pour inventer un mot français.

2 Programmes de génération de mots

2.1 Sans chaînes de Markov

2.1.1 Enchaînements de deux lettres

Le premier type d'algorithme est une reproduction de celui de David Louapre de SciencesEtonnantes : il utilise les probabilités qu'une lettre soit suivie d'une autre dans la langue française, probabilités qui sont stockées dans une matrice.
Une fonction ouvre un fichier `.txt` et le transforme en liste de caractères `file`. On définit préalablement un alphabet latin de taille 55 pour prendre en compte les lettres accentuées, et on associe à chaque lettre du dictionnaire un chiffre. On crée notre matrice de probabilité `P` en parcourant chaque mot du fichier. Lors du parcours d'un mot, on cherche les chiffres i de la lettre analysée, et j de la lettre qui suit (si la lettre qui suit existe), puis on ajoute 1 à `P[i][j]`. Pour obtenir la matrice des probabilités, on divise chaque élément de `P[i]` par le nombre total de combinaisons de deux lettres commençant par lettre codée par le chiffre i. Sur la carte de la matrice, les cases en bleu foncé représentent les enchaînements inexistants, comme 'qa', 'qb'. On remarque une case rouge sur la matrice 2D en 'qu' qui est la combinaison commençant par 'q' la plus courante.

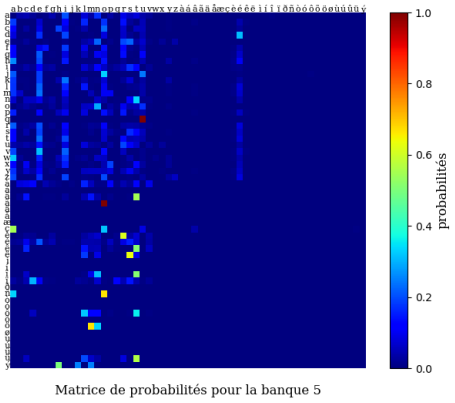


Figure 1 - Représentation 2D de la matrice de probabilités
des enchaînements de lettres des mots de la banque 5 (140 000 mots)

Pour générer un mot de taille n, on choisit une première lettre parmi les premières lettres les plus courantes en utilisant un tableau de probabilité stockant les probabilités qu'une lettre soit la première dans un mot. Via la commande `random.choice(dictionnaire,1,p=P)` de numpy, on choisit dans l'alphabet une lettre suivante en prenant en compte les probabilités qu'elle suive notre première lettre, stockées dans `P`. On réitère ce processus jusqu'à avoir la $n-$ème lettre du mot.

A droite sont donnés 10 mots de 10 lettres obtenus par le programme. Les résultats sont moyens. Si en effet, les enchaînements de deux lettres obtenus sont assez courants, les têtes et queues de mots sont composés d'enchaînements peu courants. Par exemple, $'ss'$ et $'tt'$ sont des combinaisons fréquentes, mais pas à la fin ou en début de mot. La fréquence de telles combinaisons fait qu'on obtient souvent des quadruples $'s'$ ou $'l'$ à la suite.

```
cherelithe
gouezoutyp
contaiotez
rmoglassis
dessauseet
bretauauti
dyerauoles
decaronnte
mentraresa
griegonstt
```

2.1.2 Enchaînements de trois lettres

Afin d'améliorer la qualité des mots sortants de notre programme, nous avons envisagé d'analyser les enchaînements trois lettres. Nous avons donc une matrice de probabilités `Pprime` de taille $55 \times 55 \times 55$. `P[5][0][19]` renvoie donc la fréquence d'apparition de l'enchainement $'eau'$ dans la langue. D'autre part, un tel code permet de résoudre le problème des triples et quadruples $'s'$.

Les mots obtenus ne sont pas d'une qualité très différente, la tête et la queue des mots n'étant pas contrôlées.

```
pintronspe
viraptites
mantilisse
chogueesti
aientinabl
ssespanaie
guassesorm
aferanavol
lsentereli
ciraissien
```

2.1.3 Contrôle sur le début et la fin des mots

Un contrôle sur la tête et la queue des mots générés peut enrichir notre programme et optimiser les mots obtenus. Pour cela, au lieu de chercher les probabilités d'enchaînements de lettres tout au long des mots du fichier, on ne les cherche qu'au début (respectivement, qu'à la fin) du mot. Nous implémentons alors ce contrôle dans le programme utilisant les enchaînements de deux (ou trois) lettres.

Ici avec le programme utilisant les enchaînements de trois lettres et un contrôle en début et fin de mots, nous obtenons des mots de meilleure qualité. On peut généraliser ce contrôle sur toute la longueur du mot en utilisant les chaînes de Markov.

```
avageausez
renciezoua
surisquise
debosteure
emeustorws
inersevole
garieraise
enctaierse
acessesont
pogittaite
```

2.2 Chaînes de Markov

L'objectif de cette partie est de prendre en compte la position de la lettre dans le mot. En effet, contrairement aux codes précédents, nous avons considéré que dans la langue française, certains enchaînements sont davantage placés au début d'un mot, ou en avant-dernière possition du mot. Par exemple $'im'$ dans 'important', ou encore $'er'$ de 'manger' en dernière position cette fois-ci. On évite ainsi de générer un mot commençant par $'ss'$ par exemple. On peut ici parler de processus de Markov à temps continu, en effet ici à chaque étape, nous construisons des matrices de passage prenant en compte la ou les deux lettres d'avant,stochastiques, et indépendantes. C'est à dire que seulement la ième matrice génère la ième lettre, ainsi la génération de la fin du mot ne sera pas influencé par le début.

2.2.1 Enchaînements de deux lettres

Ce premier code fait appel aux probabilités qu'une lettre soit suivie d'une autre en fonction de la place de la lettre dans le mot. Ces probabilités sont stockées dans plusieurs matrices : P1 pour les probabilités qu'une lettre en première position soit suivie d'une autre, P2 pour les probabilités qu'une lettre en deuxième position soit suivie d'une autre, etc. Par exemple :

```
P2[i][j]
```

renvoie la probabilité que la lettre de la case i, quand elle est en deuxième position dans un mot, soit suivie de la lettre de la case j.

On commence par générer une première lettre selon la distribution des premières lettres des mots français. Puis chaque nouvelle lettre en fonction de la probabilité de l'obtenir sachant la lettre d'avant et sachant sa position. Nous effectuons ce processus jusqu'à la dernière lettre, où nous ne regardons plus la 6ème, si le mot est à 6 lettres, mais la dernière lettre de tous les mots, pour terminer "proprement" le mot.
Les résultats sont assez satisfaisants à ceci près que les cas de quadruples consonnes identiques sont assez fréquents, ce qui n'existe en aucun cas dans notre langue.

```
hastissere
asitisisee
lilamieras
elastaphai
cinagupais
cracherais
trironndas
amaicuveue
jalollllle
conguiqura
```

2.2.2 Enchaînements de trois lettres

Pour remédier aux quadruples consonnes identiques, une solution a été de regarder non plus la lettre d'avant, mais les deux précédentes. Ainsi la probabilité d'obtenir trois consonnes identiques d'affilées étant nulle, il est ici impossible de générer un mot les contenant.
Les résultats sont bien plus satisfaisants que les autres programmes, les mots commencent et finissent plus proprement que précédemment.

Le programme est néanmoins beaucoup plus long que les précédents et fait chaque fois appel au parcours intégral du fichier lors de la génération d'un lettre. Le temps moyen de production d'un mot avec ce programme est de $t_{moy} = 1.85$ secondes.

```
cipineesse
repargesai
larintrepa
veraierats
soudamaite
renfraisai
meclention
plogentois
fachaiezat
trecquasse
```

3 Evaluation des programmes

L'objectif du programme réalisé dans cette partie est d'évaluer la qualité des mots sortant de chaque programme afin d'une part de voir quels programmes permettent au mieux d'inventer des mots ressemblant à des mots français, en respectant au maximum les contraintes imposées par la langue française, et d'autre part d'optimiser les mots inventés en ne conservant que les meilleurs. Dans cette partie nous utilisons la banque de mots mots.txt contenant environ 300000 mots français. L'idée est d'étudier les trigrammes (enchaînement de trois caractères). Pour ce faire, on crée différents fichiers txt dans lesquels on répertorie les trigrammes selon leur probabilité d'apparition dans la langue française : enchaînements impossibles (jamais vu en français donc probabilité = 0), très rares, rares, moyennement rares, courants, très courants (on obtient des mots français dans ce cas).

Tout d'abord, on crée une matrice $26 \times 26 \times 26$ qui répertorie toutes les combinaisons possibles de trigrammes. On réalise ensuite un programme qui sépare la liste de mot générée par les différents programmes précedents en une liste des trigrammes. On peut alors faire un programme qui mesure la fréquence d'apparition de chaque trigramme et répertorier cette valeur dans une matrice associée à la matrice $26 \times 26 \times 26$ de toutes les combinaisons de trigramme. On a donc une matrice des probabilité d'apparition de chaque trigramme dans la langue française. En fonction de cette probabilité, on crée cinq intervalles d'apparition (très rares, rares, fréquents, très fréquents, les plus fréquents). On peut alors créer un programme qui, pour une liste de mot donnée, classe ses trigrammes dans les intervalles. Enfin grâce à un programme de notation on peut juger le programme générant la liste de mot d'origine en fonction du classement de ses trigrammes et lui attribuer une note.

```
note sans Markov, enchaînements de deux lettres, sans contrôle:  9.752194114610223 /10
note sans Markov, enchaînements de trois lettres, sans contrôle:  9.887772552372809 /10
note sans Markov, enchaînements de deux lettres, avec contrôle:  9.75013150973172 /10
note sans Markov, enchaînements de trois lettres, avec contrôle:  9.9146110056926 /10
note avec Markov, enchaînements de deux lettres:  9.854268292682928 /10
note avec Markov, enchaînements de trois lettres:  9.942225826069329 /10
```

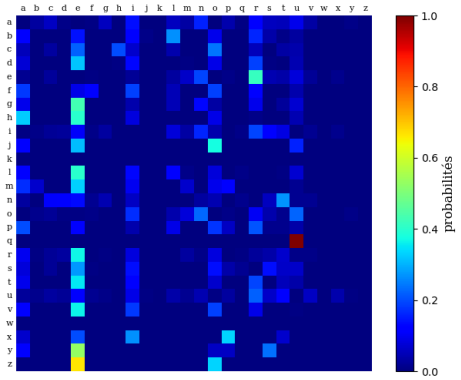
Les programmes présentent une majorité de trigrammes "très fréquents" ce qui montre la qualité de ces derniers. Au niveau des méthodes sans chaînes de Markov, on voit que les méthodes avec contrôle sont naturellement meilleures que celles sans. De plus, on constate que les meilleurs programmes sont ceux prenant en compte les trigrammes. Ils sont en effet plus puissants et plus précis que ceux prenant seulement en compte les bigrammes. Le meilleur programme étant celui combinant l'utilisation des enchaînements de trois lettres et des chaînes de Markov. A noter tout de même que les notes fluctuent en fonction des mots de sortie, pour avoir des notes encore plus précises il faudrait faire davantage de tirages, toutefois le temps d'exécution augmenterait d'autant plus.

Avec la martice des probabilités d'apparition des trigrammes, on peut alors s'amuser à trouver la probabilité d'apparition d'un mot de trois lettre en particulier. En effet, si l'on fixe le nombre de lettre des mots générés à trois, il suffit de regarder la probabilité du trigramme identique à ce mot de trois lettre dans la matrice des probabilités des trigrammes. Le mot eau correspondrait à l'élément d'indice $i = 4, j = 0, u = 20$ ce qui donne une probabilité de 0.0003633.

4 Les matrices de probabilités

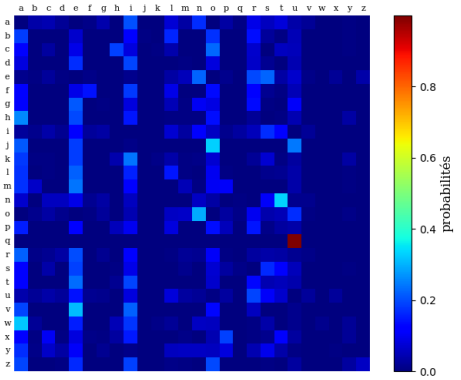
4.1 Carte des matrices des différentes banques de mots

Les différentes banques de mots utilisées donnent des matrices de probabilités variées générant des mots de structure différente selon la banque choisie. Nous pouvons analyser ces différences grâce à aux cartes des matrices. Nous n'étudions que les matrices tronquées de taille 26×26 représentant les enchaînements des caractères de a à z (sans les caractères spéciaux).



Matrice de probabilités pour la banque 3

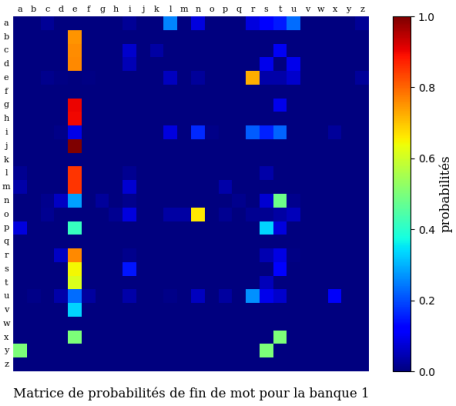
Figure 2



Matrice de probabilités pour la banque 5

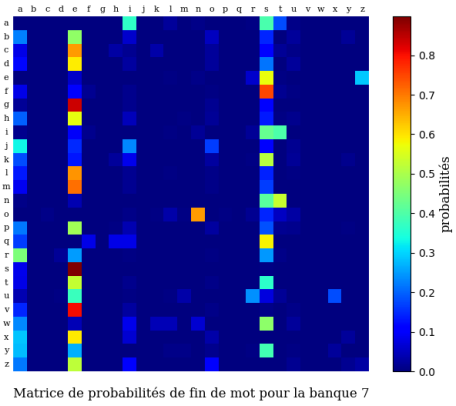
Figure 3

La comparaison des cartes des matrices de probabilités d'enchaînement de deux lettres pour une banque de petite taille de mots courants et celle de qu'une grande taille exhaustive nous montre de grandes divergences. Nous prenons la banque 3 (1500 mots) pour celle de petite taille, et la banque 5 (140000 mots) pour celle de grande taille. La première est plus contrastée que la seconde ; cela s'explique par le grand nombre d'enchaînements rares non présents dans la banque de petite taille, réprésentés par un bleu marine foncé sur la carte. Les pics sont plus marqués pour la banque de mots courants car les mots courants ont plus de chance de présenter des enchaînements courants qu'un mot au hasard de la langue n'en ait.



Matrice de probabilités de fin de mot pour la banque 1

Figure 4



Matrice de probabilités de fin de mot pour la banque 7

Figure 5

Les matrices de probabilités d'enchaînements de lettres à une position précise, utilisées lors du contrôle des débuts et fin de mots dans nos premiers types de programme, et exploitées tout au long du programme utilisant les chaînes de Markov, exhibent des différences encore plus marquées. Elles sont toutes plus contrastées que les matrices d'enchaînements de lettres à position quelconque, comme on peut le voir en comparant celles de la banque 7. Il y a beaucoup moins d'enchaînements de lettres possibles en fin de mot. Par exemple, les combinaisons $'qu'$, $'ss'$, $'tt'$, courantes en français comme on le voit dans la figure 3, sont très rares voire inexistantes en fin de mot, ce qui est représenté par un bleu foncé dans les cases de ces combinaisons de la matrice de fin. A l'inverse, la matrice de fin présente de forts pics pour les enchaînements se terminant par $'e'$ et $'s'$ (pluriels), très atténués dans la matrice régulière. A gauche est représentée la matrice de probabilités des enchaînements de lettres en fin de mots pour la banque 1. La différence majeure avec celle de la banque 7 est l'absence de pics pour une fin de mot en $'s'$. Cela s'explique par le fait que la banque 7 fournit des mots déclinés au pluriel alors que la banque 1 ne fournit que leur singulier.

4.2 Evolution des matrices en fonction de la taille de la banque

Les cartes des matrices de probabilités montrent de grandes variations pour des banques de mots de petite taille de grandes similarités pour des banques de mots de grande taille. On pourrait faire l'hypothèse que les matrices de probabilités "convergent" vers la matrice de taille 400 000 ayant analysé tous les mots français, et on peut visualiser cette convergence en regardant le comportement de certaines cases des matrices en fonction de la taille de la banque.

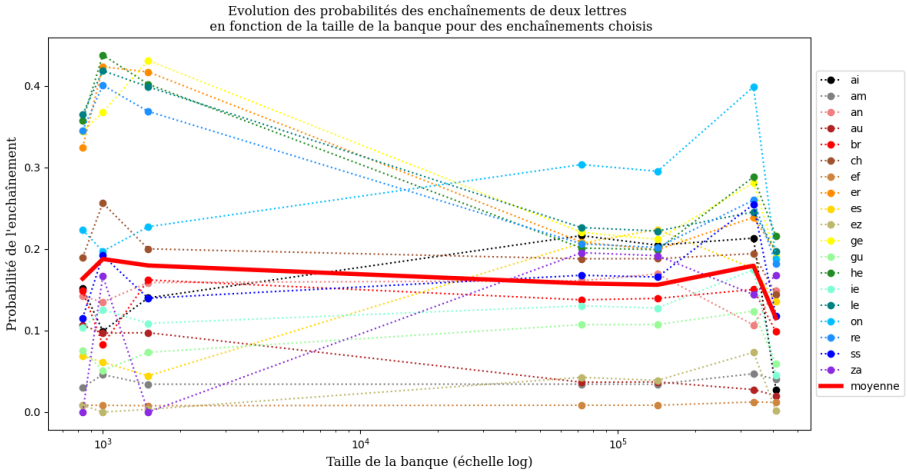


Figure 6

Sur le grahe ci-dessus d'évolution des probabilités des enchaînements de deux lettres en fonction de la taille de la banque pour certains enchaînements judicieusement choisis, on constate que les probabilités fluctuent jusqu'à un ordre de 0.1 au niveau des banques de petite taille. Ces probabilités se stabilisent au fur-et-à-mesure que la taille de la banque grandit.

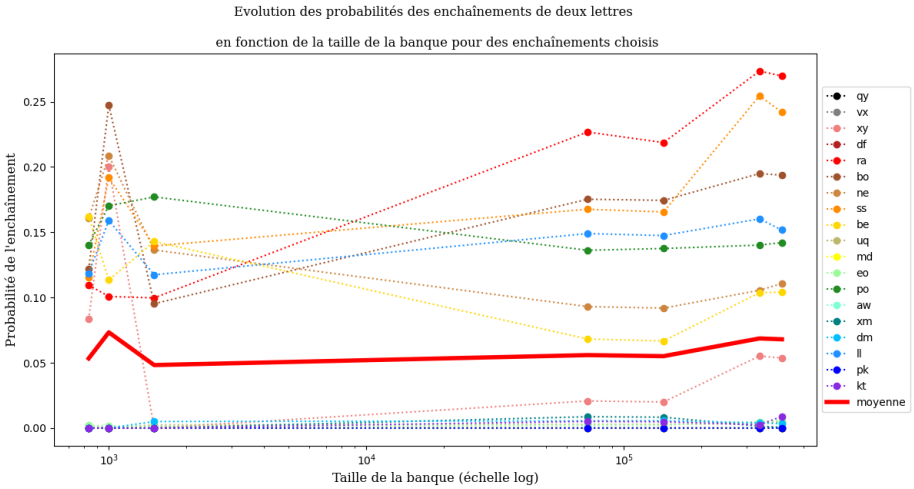
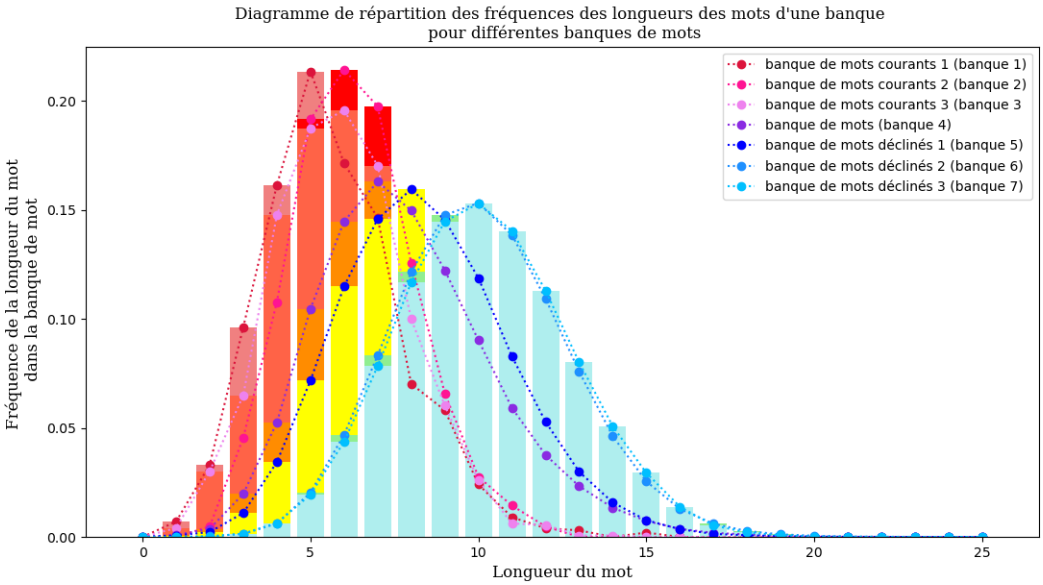


Figure 7

5 Statistiques

5.1 Détermination de n_{rec}

Nos programmes permettent de générer un mot de longueur n souhaitée. On peut suggérer une longueur recommandée n_{rec} pour faire en sorte que la longueur des mots générés soit proche de la longueur des mots de la langue française.



L'analyse de la longueur des mots français dans nos différentes banques de mots nous a permis de déterminer la répartition de ces longueurs et leur moyenne. Le diagramme de répartition des fréquences des longueurs des mots d'une banque pour différentes banques de mots nous confirme l'intuition que les mots courants (barres rouges)

sont plus courts que l'ensemble des mots (barres jaune), qui sont eux-mêmes évidemment plus courts que ces mots déclinés (barres bleues). Du calcul des moyennes des longueurs des mots pour chaque type de banque de mots, on déduit différents n_{rec} en fonction du type de banque avec laquelle on travaille.

type de banque	mots courants	tous les mots	tous les mots déclinés
n_{rec}	6	9	10

5.2 Détermination de n_{max}

On peut également se demander jusqu'à quel n_{max} il serait pertinent de faire tourner ces programmes, en particulier pour celui utilisant les chaînes de Markov dans lequel il est nécessaire de créer autant de matrices de probabilités qu'il est de lettres dans le mot à générer. Un mot de longueur trop grande n'aurait de plus, pas d'intérêt car sa longueur pourrait l'éloigner de la structure des mots de la langue française.

Une première solution simple serait de chercher la longueur maximale des mots de chaque type de banque. n_{max} serait cette longueur maximale. Pour évaluer la pertinence de cette méthode, on cherche le temps moyen t_{moy} que le programme utilisant les chaînes de Markov et les enchaînements de deux lettres prend pour génerer un mot de longueur n_{max} avec 50 mots.

type de banque	mots courants	tous les mots	tous les mots déclinés
n_{max}	14	25	25
$t_{moy} \times 10^3(s)$ sans Markov	0.136	1.21	1.84
$t_{moy} \times 10^3(s)$ avec Markov	1.19	\	\

Créer des mots de plus de 20 lettres avec la méthode de Markov génère beaucoup d'erreurs car les matrices de probabilités P_{21}, P_{22}, ... ont des colonnes et lignes nulles, et il n'est donc pas possible d'appliquer `random.choice` dessus, les mots d'une telle longueur étant rares même dans nos plus grandes banques. Même si on n'a pas ce problème avec la méthode sans chaîne de Markov, les mots obtenus ne sont pas très pertinents.

Une autre solution serait de déterminer un seuil α de sorte que $(1-\alpha) \times 100\%$ des longueurs des mots de la banque soient plus petites que n_{max}.

pour $\alpha = 5\%$

type de banque	mots courants	tous les mots	tous les mots déclinés
n_{max}	10	14	15

pour $\alpha = 2\%$

type de banque	mots courants	tous les mots	tous les mots déclinés
n_{max}	11	15	17

Les programmes de Markov ne tournent pas toujours avec $n_{max} = 17$.

Avec un seuil $\alpha = 5\%$ et un $n_{max} = 15$, on obtient un temps moyen de $t = 0.00116s$ pour la méthode sans chaînes de Markov et, un temps de $t = 0.00079s$ avec. Le programme tourne bien jusqu'à P_{15}.

6 Partie graphique

Pour donner une dimension de logiciel à notre programme, nous avons également décidé de réaliser une fenêtre graphique. L'objectif était que l'utilisateur puissent contrôler la générations de nouveaux mots en lui demandant combien de lettres et combien de mots il souhaitait. Pour cela, nous avons utilisé le module tkinter, module python parfait pour cet objectif.

La construction de la fenêtre est basée sur la méthode grid. Cette dernière permet de construire notre fenêtre ligne par ligne et colonne par colonne. On peut alors placer les éléments dans l'odre voulu de manière très simple. Tout a été construit pour que n'importe quel utilisateur puisse immédiatement comprendre et créer des mots. Il y a quand même un bouton information, afin d'éclaircir tout doute. Nous avons également prévu un programme qui utilise une bonne partie de nos banques de mots afin de laisser encore plus de choix à l'utilisateur.
Il suffit donc de sélectionner le nombre de lettres et de mots souhaités, puis de cliquer sur le bouton de la langue souhaitée afin de voir les nouveaux mots.(Ce programme est un peu plus lent car on utilise plus de banque de mots.)

Figure 9 - Fenêtre graphique multilangue

7 Conclusion et Perspectives

La réalisation du programme complet aura nécéssité l'utilisation de 9 modules python. Nous avons réussi à générer des mots cohérents avec la langue française et pu évaluer nos différents programmes de façon a en choisir un meilleur : le programme utilisant les chaînes de Markov et les enchaînements de trois lettres nous donnent les mots les plus optimisés.

De nombreuses applications à ce programme peuvent être envisagées. En effet, on pourrait créer un programme qui code un texte avec les mots inventés de notre programme, ou imaginer un programme qui créer des poèmes avec ces nouveaux mots.

8 Annexe

Nous avons, pour notre étude, utilisé sept banques de mots de caractéristiques et tailles variées. Par ordre de taille croissante :

- Banque 1 : une banque de 835 mots courants de la langue française offerte par le développeur web Boris Guéry sur GitHub : `https://gist.github.com/borisguery/6c94d67be8f531f986fc66e066236324`
  Les mots de cette banque sont principalement des noms.
  Les mots de cette banque ne sont pas déclinés.
- Banque 2 : une banque de 1 000 mots courants de la langue française offerte par le développeur web CodeBraeuer sur GitHub : `https://github.com/CodeBrauer/1000-most-common-words/blob/master/1000-most-common-french-words.txt`
  Cette liste de mots simples est destinée à l'apprentissage du français.
  Les mots de cette banque ne sont pas déclinés.
- Banque 3 : une banque de 1 500 mots les plus fréquents de la langue française offerte pas le ministère en 2010 et consitutée par le lexicologue Etienne Brunet. `https://www.ac-paris.fr/serail/jcms/s1_814789/fr/vocabulaire-liste-de-frequence-lexicale-de-1500-mots-mise-a-disposition-par-le-ministere-2010`
  Elle a été mise à disposition par eduscol afin de favoriser l'utilisation de certains mots.
  Les mots de cette banque ne sont pas déclinés.
- Banque 4 : une banque de 72 303 mots de la langue française offerte par le projet Lexique de Boris New et Christophe Pallier :
  `http://www.lexique.org/`
  Cette banque fait partie d'une base de données visant à recenser les mots de la langue et à regrouper diverses informations sur ces mots comme leurs fréquences d'occurrences, la représentation phonologique, la catégorie grammaticale, entre autres.
  Cette banque de mots non déclinés a été compilée à partir de la banque de mots plus grande de mots déclinés du même site, constituant notre banque suivante.
- Banque 5 : une banque de 142 695 mots de la langue française de la même provenance que la banque 4. `http://www.lexique.org/`
  Les mots de cette banque sont déclinés.
- Banque 6 : une banque de 336 531 mots de la langue française offerte par le Projet Gutemberg et compilés sur GitHub par Christophe Pallier. C'est celle utilisée par David Louapre dans sa machine à inventer des mots. `https://github.com/chrplr/openlexicon/tree/master/datasets-info/Liste-de-mots-francais-Gutenberg`
  Cette banque visait à être aussi exhaustive possible dans son regroupement des mots français.
  Les mots de cette banque sont déclinés.
- Banque 7 : une banque de 411 430 mots de la langue française de la huitième edition du dictionnaire officiel du Scrabble et accessible sur le site : `https://www.listesdemots.net/touslesmots.htm`
  Les mots de cette page ont été compilés grâce au code fournit sur ce blog : `https://blog.site2wouf.fr/2018/12/un-lexique-genre-ods7-en-txt.html`
  Cette banque visait également à l'exhaustivité.
  Les mots de cette banque sont déclinés.
- Banque 8 : une banque de 194 433 mots de la langue anglaise : `https://www.listesdemots.net/touslesmots.htm`
- Banque 9 : une banque de 60 454 mots de la langue italienne : `https://informationpoint.forumcommunity.net/?t=37000925`
- Banque 10 : une banque de 180 015 mots de la langue néerlandaise : `http://www.gwicks.net/dictionaries.htm`
- Banque 11 : une banque de 10 282 mots de la langue afrikaans : `https://github.com/oprogramador/most-common-words-by-language`
- Banques 12 à 21 : banques de 50 000 mots des langues, respecivement, albanaise, catalanne, danoise, estonienne, finnoise, norvégienne, polonaise, portugaise, serbe, suédoise : `https://github.com/oprogramador/most-common-words-by-language`