Alexandre Verzura Victor Vannobel

Compte rendu projet de statistique computationelle

Avril 2023

I) Intervalles de confiance et prédictions par le bootstrap

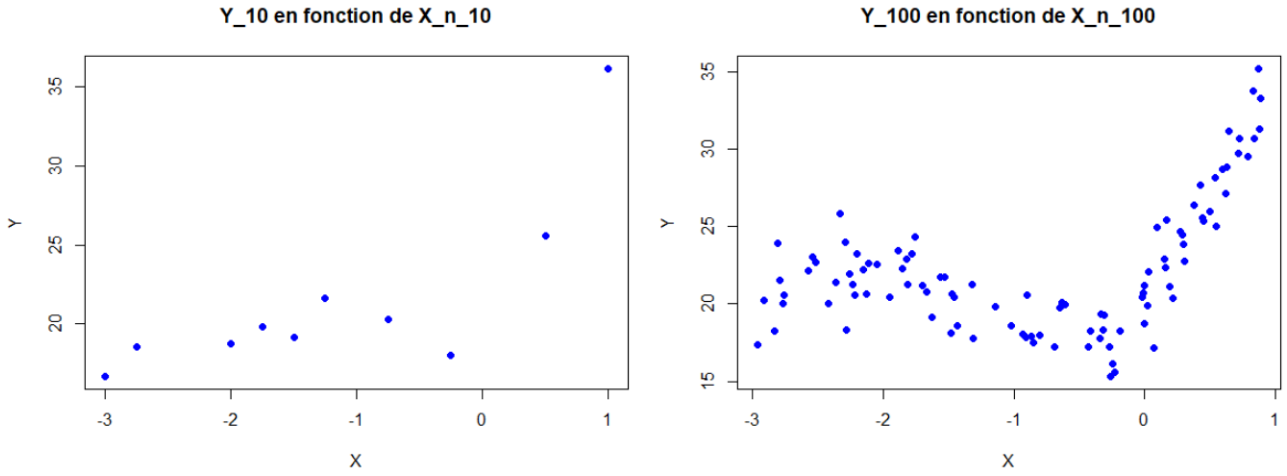
Introduction

Dans cet exercice, nous allons comparer deux approches de type bootstrap pour la construction d'intervalles de confiance en régression et évaluer leurs performances en termes de précision et de couverture.

Dans tout l'exercice, $n \in \{10, 100\}$ selon le cas. Nous posons $\theta = (a, b, c, d, \sigma^2)$ et nous utiliserons $\theta^* = (20, 7, 8, 2, 1.75^2)$.

Question 1 :

Dans chaque cas, on a généré selon la seed 2023 $Y_i = a + bX_i + cX_i^2 + dX_i^3 + \epsilon_i$ avec $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, puis on a tracé Y_i en fonction de X_i, ce que représente la figure 1. Ce sont avec ces jeux de données que nous allons ajuster notre modèle, puis nous créerons des valeurs de test dites vraies valeurs en question 3 pour tester les performances de notre régression.



(Figure 1) Affichage des jeux de données.

Question 2 :

Il s'agit en théorie de maximiser une certaine fonction de vraisemblance pour obtenir les estimateurs de a, b, c, d et σ^2, ce que fait la fonction lm de R en plus de fournir des intervalles de confiance à 95% sur les 4 coefficients de notre modèle. Pour l'intervalle de confiance sur σ^2, on utilise le fait que l'estimateur sans biais de la variance $\hat{\sigma^2_n} = \frac{1}{n-4}\sum_{i=1}^{n} \hat{\epsilon_i}^2 \sim \frac{\sigma^2}{n-4}\chi^2(n-4)$. L'estimateur par maximum de vraisemblance de θ dans les deux cas est affiché sur la figure 2, avec de gauche à droite les EMV de a, b, c, d et σ^2.

```
(Intercept)      X_n_10[, 1] I(X_n_10[, 1]^2) I(X_n_10[, 1]^3)
  21.039626         6.853860         6.239891         1.491737         2.398283
```

n=10

```
(Intercept)      X_n_100[, 1] I(X_n_100[, 1]^2) I(X_n_100[, 1]^3)
  20.270112          7.364491          7.680231          1.787213          2.805410
```

n=100

(Figure 2) EMV de θ.

La figure 3 met en évidence les intervalles de confiance à 95% obtenus grâce à la fonction lm pour chaque composante de θ, dans les deux cas.

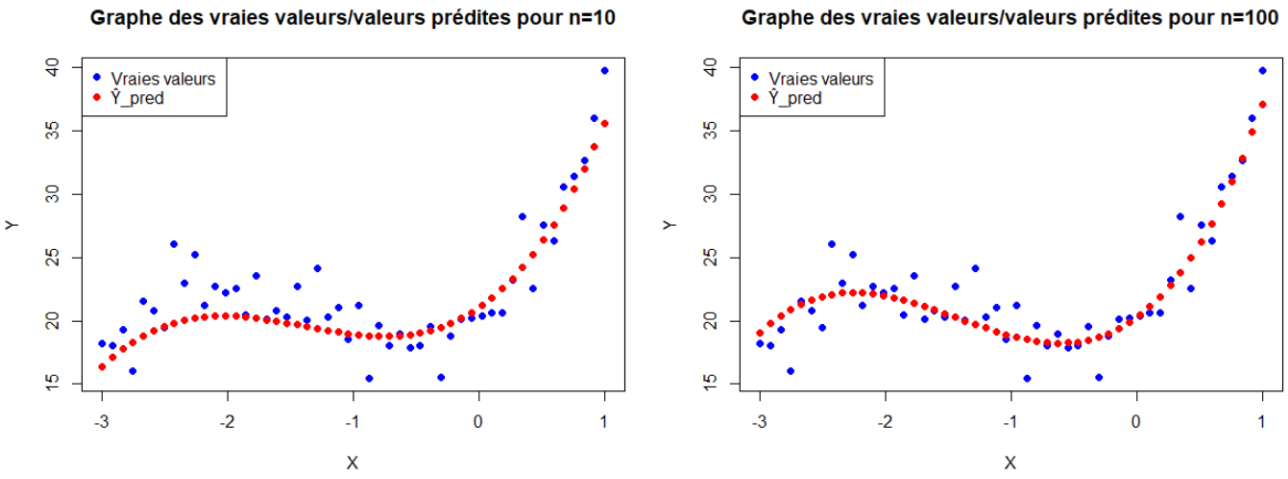
```
                2.5%     97.5%                2.5%     97.5%
a         18.4559341 23.623318 a          19.716266 20.823958
b          4.9798135  8.727906 b           6.692786  8.036197
c          3.5477604  8.932021 c           6.791235  8.569226
d          0.6791203  2.304354 d           1.503785  2.070640
sigma^2    0.9958701 11.629504 sigma^2     2.154554  3.804869
```

n=10 n=100

(Figure 3) Intervalles de confiance à 95% pour les composantes de θ.

Question 3 :

Nous créons un polynôme du 3-ème degré avec pour coefficients les EMV des paramètres de θ que l'on applique à un vecteur X_{pred} de 50 valeurs réparties uniformément dans l'intervalle [-3,1] pour obtenir le vecteur des valeurs prédites $\hat{Y}_{pred}$. Nous allons comparer nos valeurs prédites avec de nouvelles valeurs obtenues à partir de X_{pred}, de θ^* et d'un nouveau bruit, appelées vraies valeurs ou valeurs de test. Les résultats sont présentés dans la figure 4.

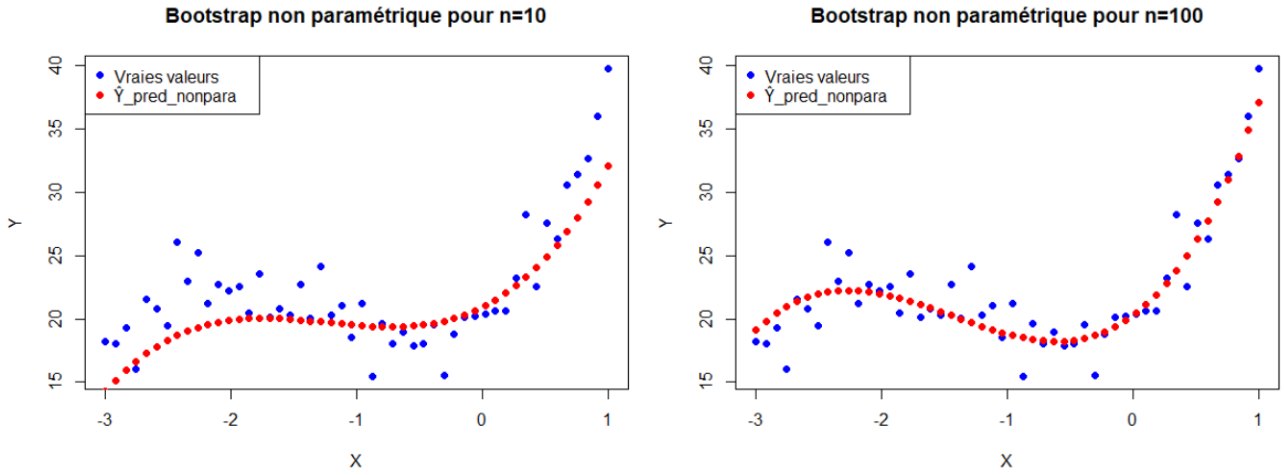


(Figure 4) Visualisation des valeurs prédites par EMV dans chaque cas.

La notation "pour n=..." signifie le cas selon que notre modèle a été adapté avec des échantillons de taille 10 ou 100.

Question 4 :

Cette fois-ci, nous partons des couples (X_i, Y_i) générés à la question 1 auxquels nous allons effectuer un ré-échantillonnage par bootstrap non paramétrique de la manière suivante : Pour $1 \leq b \leq B = 2000$, on effectue à chaque itération n tirages uniformes et avec remise parmis les couples (X_i, Y_i) pour obtenir de nouveaux couples $(X^*_{i,b}, Y^*_{i,b})$ auxquels nous allons appliquer la fonction lm pour obtenir les estimateurs $\hat{a}^*_b$, $\hat{b}^*_b$, $\hat{c}^*_b$, $\hat{d}^*_b$ et $\hat{\sigma^2}^*_b$. Enfin nous faisons la moyenne des estimateurs des B échantillons pour obtenir les estimateurs bootstraps des paramètres de θ, ce qui nous permet d'obtenir les valeurs prédites par bootsrap non paramétrique (figure 5), que nous comparons aux vraies valeurs de la question 3. Par ailleurs, nous avons choisi $B = 2000$, car au delà les calculs étaient plus longs pour pas beaucoup plus d'amélioration des résultats.



(Figure 5) Visualisation des valeurs prédites par bootstrap non paramétrique par rapport aux vraies valeurs dans chaque cas.

Pour les intervalles de confiance, nous utilisons les intervalles de confiance du boostrap classiques donnés par $\hat{IC}(0.95) = \left[2\hat{\theta} - \hat{\theta}^*_{\lceil B*0.975 \rceil}, 2\hat{\theta} - \hat{\theta}^*_{\lceil B*0.025 \rceil}\right]$. Les résultats sont présentés ci-dessous en figure 6.

	2.5%	97.5%
a	15.7636446	27.057339
b	5.8399183	27.268755
c	2.1204457	21.679895
d	-0.7873571	4.716670
sigma^2	1.8249522	4.795282

n=10

	2.5%	97.5%
a	19.677678	20.897781
b	6.683725	7.963249
c	6.705230	8.646269
d	1.477493	2.092580
sigma^2	2.147121	3.645107

n=100

(Figure 6) Intervalles de confiance du bootstrap non paramétrique classiques à 95% pour les composantes de θ.

Ce n'était pas demandé mais nous avons remarqué que puisque les échantillons utilisés pour ajuster le modèle étaient de taille relativement petite, les intervalles de confiance du boostrap censés être à 95% ne capturaient les vraies valeurs des paramètres moins de 95% du temps. Nous avons ensuite utilisé la fonction boot.ci de R (bibliothèque "boot") avec comme mode de calcul des intervalles de confiance le mode "bca" pour "Bias-Corrected-Accelerated" ce qui a permis d'obtenir des intervalles de confiance plus précis (figure 7).

	2.5%	97.5%
a	(19.64,	20.87)
b	(6.716,	7.996)
c	(6.684,	8.634)
d	(1.456,	2.066)
sigma^2	(2.204,	3.853)

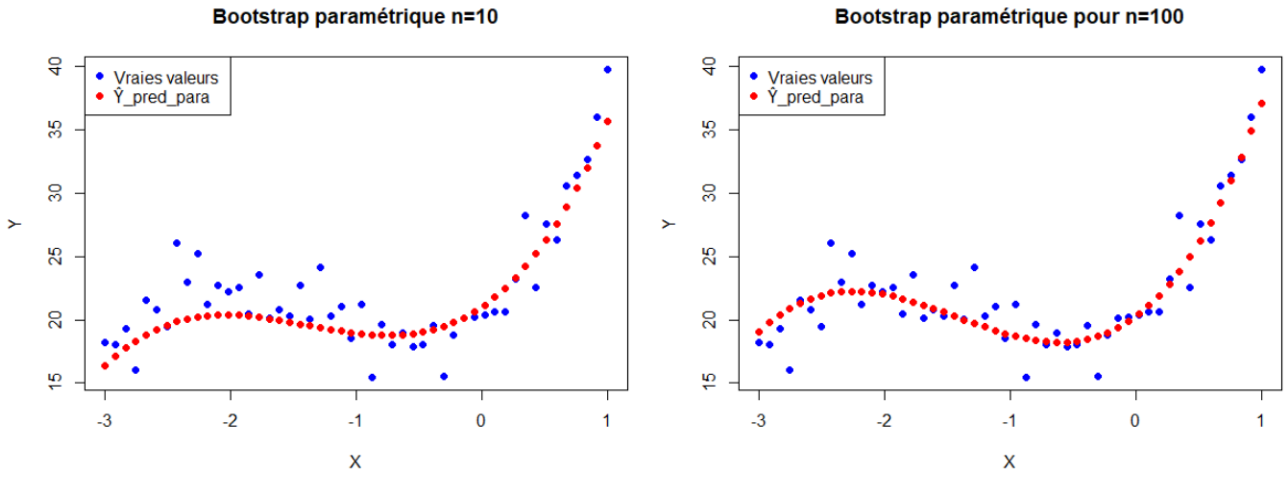
n=100

(Figure 7) Intervalles de confiance du bootstrap non paramétrique BCA à 95% pour les composantes de θ dans le cas n=100.

Nous avons choisi de ne pas afficher dans le compte rendu les intervalles de confiance à 95% des valeurs prédites par bootstrap des questions 4 et 5 mais ils sont accessibles dans le code et représentés graphiquement dans la question 6.

Question 5 :

Nous allons ici faire un ré-échantillonnage par bootstrap paramétrique, c'est-à-dire ré-échantillonner les résidus $\hat{\epsilon}_i$ obtenus à la question 2 pour obtenir nos échantillons bootstraps et nous appliquerons la même procédure qu'à la question précédente.
Nous obtenons alors nos valeurs prédites par boostrap paramétrique que nous comparons aux vraies valeurs créées en question 3 (figure 8) ainsi que les intervalles de confiance du bootstrap paramétrique classiques à 95% pour les paramètres de θ (figure 9).



(Figure 8) Visualisation des valeurs prédites par bootstrap paramétrique par rapport aux vraies valeurs dans chaque cas.

	2.5%	97.5%
a	19.427469	22.548965
b	5.698073	7.937270
c	4.588031	7.846420
d	1.004941	1.994212
sigma^2	1.918880	4.473294

n=10

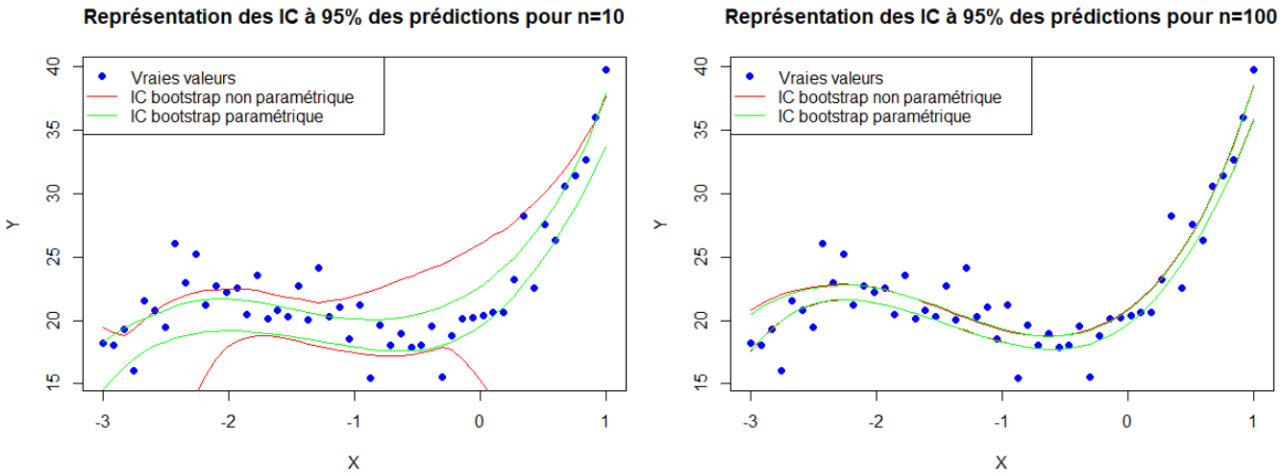
	2.5%	97.5%
a	19.768147	20.816253
b	6.753963	8.005560
c	6.870144	8.525802
d	1.527952	2.067717
sigma^2	2.182204	3.588465

n=100

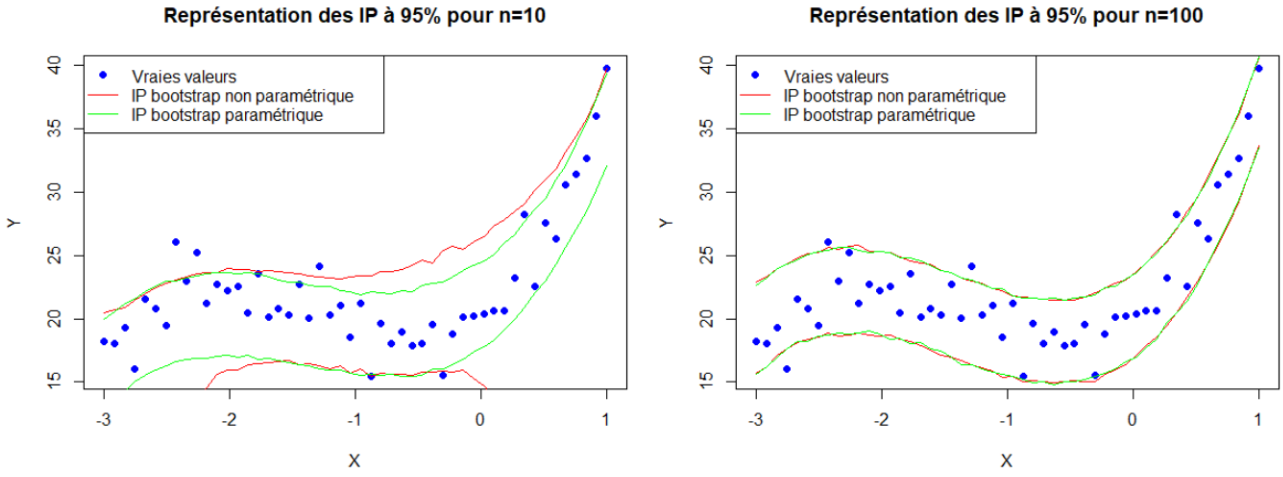
(Figure 9) Intervalles de confiance du bootstrap paramétrique classiques à 95% pour les composantes de θ.

Question 6 :

Nous allons comparer graphiquement les intervalles de confiance des prédictions en les superposant aux vraies valeurs de la question 3 (figure 10) ainsi que les intervalles de prédiction aux vraies valeurs (figure 11). Les intervalles de confiances à 95% des valeurs prédites ont été obtenus en créant les vecteurs $\hat{Y}_{pred}$ associés aux B échantillons bootstraps et en appliquant la fonction quantile aux B valeurs de chaque $\hat{Y}_{pred,j}$ pour $j \in \{1,...,50\}$. Pour obtenir les intervalles de prédiction, nous ajoutons un bruit gaussien $\mathcal{N}(0,\hat{\sigma_n^2})$, avec $\hat{\sigma_n^2}$ l'estimateur de la variance obtenu en question 2 selon chaque cas, aux B vecteurs $\hat{Y}_{pred}$ et nous appliquons la même méthode.



(Figure 10) Comparaison graphique des différents intervalles de confiance pour les valeurs prédites dans chaque cas.



(Figure 11) Représentation des intervalles de prédiction dans chaque cas.

Question 7 :

D'après la figure 10, nous remarquons graphiquement que pour un échantillon initial de petite taille (n=10) servant à adapter le modèle, la méthode du bootstrap paramétrique est plus adaptée que celle du bootstrap non paramétrique. En effet, les bornes de l'intervalle de confiance du bootstrap non paramétrique des valeurs prédites s'écartent sur les extrémités (cas n=10). Cela signifie que le modèle est instable et ne peut pas fournir des valeurs prédites fiables.
En revanche, pour un échantillon inital de taille plus grande (n=100), les deux méthodes sont plus ou moins équivalentes quand à la précision et à la fiabilité de prédiction de valeurs car les deux intervalles de confiance des valeurs prédites sont relativement étroits et les intervalles de prédiction sont tous les deux précis et capturent bien les valeurs de test.

II) Algorithmes MCMC et déchiffrage de message codé

Introduction

Nous nous intéressons dans cet exercice au déchiffrage de message codé. Le but est de déchiffrer la phrase suivante : "Ula gadilba qddgitpfqrpsa q cq nilla xuabrpil q naqueiud dcub oa sqcaug xu'ula gadilba dgaepba q cq fqusqpba xuabrpil". Nous savons que la méthode de cryptage est de type substitution, c'est-à-dire que chaque lettre de l'alphabet est remplacée par une autre. Nous allons mettre en pratique un algorithme de Monte-Carlo Markov Chain (MCMC) pour explorer efficacement l'ensemble des codes possibles et ainsi déchiffrer le message codé.

Question 1 :

Nous nous intéressons à la quantité de code par substitution possible. Un code par substitution peut être représenté comme une fonction bijective de l'ensemble alphabet dans lui même, permettant un réagancement des lettres. En raisonnant par dénombrement, on en déduit qu'il y a 26! bijections possibles d'un alphabet de 26 lettres dans un autre. Ainsi, en retirant la bijection identité qui ne représente pas un code, nous avons 26!-1 codes possibles.

Question 2 :

Dans un premier temps, nous avons créé deux fonctions, codage et decodage prenant en entrée une chaîne de caractères x ainsi qu'un code c (l'objet code est une matrice de taille 26×2 contenant dans sa première colonne les caractères de l'alphabet dans l'ordre alphabétique et dans sa deuxième colonne les caractères de l'alphabet substitués par une bijection). Les deux fonctions renvoient respectivement la chaîne de caractères x codée par le code c et la chaîne x décodée en utilisant le code c.

Questions 3 à 5 :

L'objectif de notre algorithme est d'explorer l'ensemble des codes de manière plus efficace. Pour ce faire, nous allons créer une matrice de transition des couples de lettres dans la langue française notée A. En effet, on fait l'hypothèse qu'une chaîne de caractères dans la langue française peut être considérée comme une chaîne de Markov. A sera donc une matrice de taille 27×27, avec $A_{i,j}$ la probabilité que le couple de caractères formé de la i-ème lettre de l'alphabet suivie de la j-ème soit présent dans la langue française. Le 27-ème caractère est l'espace, qui doit être pris en compte pour calculer les probabilités d'avoir un mot composé d'une seule lettre ainsi que la probabilité qu'un mot commence ou finisse par une certaine lettre.
Pour construire cette matrice, nous avons utilisé un texte écrit dans la langue française. (Le texte doit préalablement être "nettoyé", c'est-à-dire réduit à ses caractères utilisés pour construire la matrice. On retire donc les ponctuations ainsi que les accents et les caractères spéciaux pour ne garder que les lettres et les espaces). En parcourant l'ensemble du texte on construit l'élément de la matrice A d'indice i,j comme suit :

En notant $N_{i,j}$ le nombre de fois que le duet d'indice i,j est présent dans le texte et N le nombre de duets total, on pose : $A_{i,j} = \frac{N_{i,j}}{N}$. On a bien que, la somme des éléments de A vaut 1.
Prenons un exemple pour mieux visualiser. Si le texte considéré était "Qui est-ce?", le texte "nettoyé" serait "qui est ce " et le premier duet serait "[espace]q", le second "qu" puis "ui", "i[espace], "[espace]e, et ainsi de suite. Le nombre de duets total de ce texte serait 11.
Finalement, on peut créer une fonction nommée "dlettresfr" prenant en entrée une chaîne de caractères de longueur deux et qui renvoie la probabilité du duet dans la langue française, à savoir l'élément de A qui correspond au duet d'entrée. Pour calculer la probabilité P qu'un texte de longueur quelconque notée N soit dans la langue française on peut utiliser la fonction dlettresfr. Par indépendance sur chaque duet, on a que la probabilité P est le produit de la probabilité qu'un espace soit suivi de la première lettre du texte et des probabilités de chaque duet. On note i_n $n \in \{1, ..., N\}$ l'indice du $n_{ième}$ caractère, on a $P = A_{27,i_1} \times \prod_{n=1}^{N-1} A_{i_n,i_{n+1}}$

Questions 6 et 7 :

On va maintenant implémenter un algorithme de Metropolis-Hastings (noté MCMC) pour décoder le texte initial. Dans toute la suite du compte rendu, par abus de notation, on ne fera pas la différence entre "la probabilité d'un code X_i" et "la probabilité que le message "Ula gadilba qddgitpfqrpsa q cq nilla xuabrpil q naqueiud dcub oa sqcaug xu'ula gadilba dgaepba q cq fqusqpba xuabrpil" de l'énoncé préalablement décodé grâce au code X_i soit en vrai français".
Dans un premier temps on définit un code initial noté X_0 que l'on doit choisir tel que la probabilité de X_0 est différente de 0. Pour simplifier la recherche de ce code initial, on peut pré décoder le code en remarquant certaines lettres. En effet, dans la phrase "Ula gadilba qddgitpfqrpsa q cq nilla xuabrpil q naqueiud dcub oa sqcaug xu'ula gadilba dgaepba q cq fqusqpba xuabrpil" la lettre q est seule, on peut raisonnablement faire l'hypothèse que la lettre "a" de l'alphabet est codée par la lettre "q". On remarque aussi que la chaîne "xu'ula" code probablement l'expression "qu'une". Ainsi, nous avons décidé d'initialiser notre algorithme avec le code :

$$X_0 = \begin{pmatrix} a & b & e & l & n & q & x \\ q & b & a & e & l & x & n \end{pmatrix}^T$$

(Les lettres non modifiées par X_0 ne sont pas représentées ci-dessus.)
On notera que la probabilité issue de ce code est de 0 car la probabilité de certains duets est de 0 (i.e $\exists\ (i,j) \in [\![1,27]\!]^2$ tel que $A_{i,j} = 0$). L'algorithme devra donc préalablement créer un code proche de X_0 dont la probabilité qui en est issue n'est pas nulle.

Pour chaque nouvelle étape n de l'algorithme MCMC on doit générer un candidat de code Y_n selon une loi $q(.|X_{n-1})$. On crée cette fonction de sorte à ce qu'elle soit symétrique ($q(a|b) = q(b|a)$). On implémente donc une fonction nommée prop prenant en entrée une matrice de taille 26×2 et renvoyant la même matrice à ceci près que deux éléments de la deuxième colonne ont été intervertis.
L'objectif de cette fonction est de générer un candidat Y_n selon une loi pré-définie (ici une loi uniforme comme tirage sans remise de deux indices parmi les 26 indices possibles). Donc la loi pour générer Y_n est symétrique.

Question 8 :

On se propose dans cette question d'implémenter l'algorithme MCMC. L'algorithme est basé sur les chaînes de Markov.
On rappelle le principe général :
Initialisation : On initialise la chaîne de Markov avec X_0
Boucle itérative au rang n :
- On génère un candidat $Y_n \sim q(.|X_n)$
- On pose :

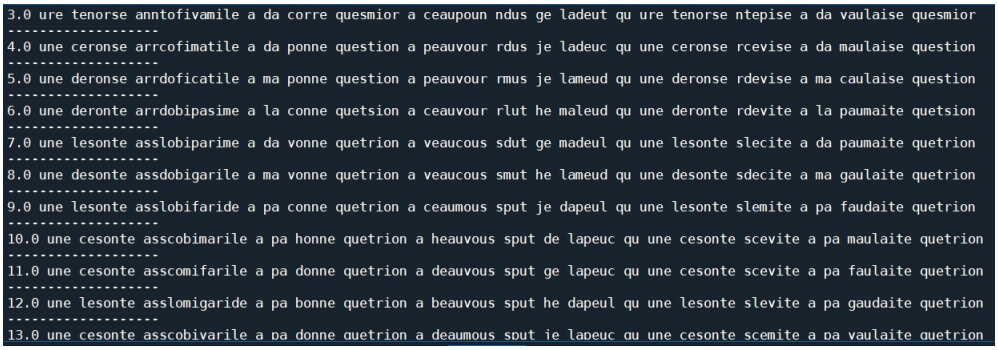
$$X_n = \begin{cases} Y_n & \text{avec une probabilité } \alpha(X_{n-1}, Y_n) \\ X_{n-1} & \text{avec une probabilité 1-}\alpha(X_{n-1}, Y_n) \end{cases}$$

Avec $\alpha(x, y) = \min\left(1, \frac{f(y)q(x|y)}{f(x)q(y|x)}\right)$

Pour implémenter correctement l'algorithme, on définit préalablement notre espace d'état E, la loi de proposition (notée q()) ainsi que la loi cible (notée f()) et la probabilité d'acceptation (notée α).
Dans le cas présent, l'espace d'état E est l'ensemble des codes possibles, pour cet algorithme on considère aussi le "code identité". E est donc l'ensemble des bijections d'un ensemble de 26 éléments dans lui même.
La loi de proposition q() est celle décrite dans le paragraphe précédent, une loi uniforme comme tirage de deux éléments d'un ensemble fini sans remise. On a donc bien que q est symétrique.
La loi cible notée f() est la loi P() des questions 3 à 5. La probabilité d'acceptation notée α est égale dans ce cas au rapport : $\alpha = \frac{P(Y_n)}{P(X_n)}$. Car la loi de proposition q() est symétrique.

Question 9 :

On peut finalement utiliser l'algorithme expliqué au paragraphe précédent pour décoder la phrase :"Ula gadilba qddgitpfqrpsa q cq nilla xuabrpil q naqueiud dcub oa sqcaug xu'ula gadilba dgaepba q cq fqusqpba xuabrpil". On l'implémente avec 20000 itérations en l'initialisant au X_0 spécifié à la question 6. L'algorithme suggère à la 20000-ème itération que le message décodé est "une lesonte asslofimarive a pa donne quetrion a deaucous sput ge vapeul qu une lesonte slecite a pa mauvaite quetrion"
On remarque que le texte décodé est proche du français sans l'être vraiment. On décide donc de relancer l'algorithme avec 10^5 itérations et d'afficher le résultat de l'algorithme toutes les 500 itérations (figure 12).



(Figure 12) Exemple de résultats de l'algorithme MCMC (avec matrice A)

L'algorithme MCMC semble tendre vers des mots inexistants mais dont l'enchaînement de lettres est fort courant dans la langue française. En comparant les différents résultats on arrive néanmoins à décrypter le message. L'algorithme à notamment affiché le message : "une deronse arrdowimative a la ponne question a peaucour rlus je valeud qu une deronse rdecise a la mauvaise question" lors d'une de ses itérations. L'algorithme ne converge donc pas vers une réponse exacte mais permet de générer des candidats très probables pour résoudre le problème. Le message codé semble donc être : "une reponse approximative a la bonne question a beaucoup plus de valeur qu'une reponse precise a la mauvaise question". La fonction de codage c serait donc similaire au code ci-dessous (figure 13) :

$\mathscr{A}$	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
$\mathscr{C}$	q	n	e	o	a	v	m	h	p	j	k	c	f	l	i	d	x	g	b	r	u	s	w	t	y	z

(Figure 13) Exemple de code par substitution possible

Le message décodé ne comporte pas l'ensemble des lettres de l'alphabet. Les lettres f,g,h,j,k,w,y et z ne sont pas dans le message décodé, ainsi la fonction de codage c peut interchanger les éléments correspondants à ces lettres. En notant I_n, $n \in [\![1,8]\!]$ les éléments interchangeables du code avec les I_n à valeurs dans {v,m,h,j,k,w,y,z} et tel que les I_n soit différents deux à deux. On en déduit que la fonction codage est de la forme du tableau ci-dessous (figure 14) :

$\mathscr{A}$	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
$\mathscr{C}$	q	n	e	o	a	I_1	I_2	I_3	p	I_4	I_5	c	f	l	i	d	x	g	b	r	u	s	I_6	t	I_7	I_8

(Figure 14) Forme de la fonction codage

La longueur du message a un impact sur la qualité du déchiffrage. Cependant, le message à décoder est d'une longueur de 117 caractères (espaces inclus), ceci est suffisant pour avoir un début de décodage.
On remarque que le code varie beaucoup au début de la boucle itérative puis que certaines lettres se stabilisent et ne varient quasiment plus. Le poids de probabilité de certains duets est donc très important, l'algorithme les considère comme "trop probables" pour les rejeter. On peut prendre l'exemple du duet "ss" présent dans le troisième mot (voir figure 12). En effet, en vérifiant dans notre matrice de transition, on voit que la probabilité d'avoir le duet "ss" est de 3.61×10^{-3}, tandis que la probabilité du duet "pp" n'est que de 6.40×10^{-4}. Il est donc raisonnable du point de vue algorithmique de considérer la lettre "s" plutôt que

la lettre "p" (nous n'avons pas vérifié l'influence des lettres "p" ou "s" dans les autres parties du message codé (autres que la double lettre du troisième mot), ce calcul n'est pas formel mais vise à comprendre la stabilité de certaines lettres dans le code). On remarque aussi que certaines lettres varient quasiment à chaque affichage. Nous avons donc calculé le quotient de la probabilité que le texte ci-dessus soit en vrai français avec la probabilité que le texte décodé par le code en sortie de notre algorithme MCMC soit en français. On obtient un quotient de $4.3752650157538684 \times 10^{-5}$
La probabilité de la phrase décodée étant plus de mille fois plus forte que la probabilité de la vraie phrase, on se questionne quand à l'exactitude de notre matrice de transition A.
On choisit de construire une nouvelle matrice de transition notée A_1 pour visualiser l'impact du biais de la matrice sur le décodage. On calcule ensuite les taux de variation entre les deux matrices (comme somme de la valeur absolue de la différence des matrices terme à terme). On obtient un taux de 0.18678125733365086.
On affiche enfin les résultats de l'algorithme MCMC avec la nouvelle matrice A_1 (figure 15) pour les comparer avec les résultats de l'algorithme avec la matrice A. (figure 12)

```
4.0 une ceronse arrcovipatile a da fonne question a feaubour rdus he ladeuc qu une ceronse rcebise a da paulaise question
------------------
5.0 une seronte arrsomifacile a da jonne quetcion a jeauvour rdut ge ladeus qu une seronte rsevite a da faulaite quetcion
------------------
6.0 une desonte assdofibarile a pa conne quetrion a ceauvous sput me lapeud qu une desonte sdevite a pa baulaite quetrion
------------------
7.0 une cesonte asscofimarile a pa gonne quetrion a geauvous sput de lapeuc qu une cesonte scevite a pa maulaite quetrion
------------------
8.0 une cesonte asscobimarile a pa donne quetrion a deauvous sput fe lapeuc qu une cesonte scevite a pa maulaite quetrion
------------------
9.0 une desonte assdomigarile a pa conne quetrion a ceauvous sput fe lapeud qu une desonte sdevite a pa gaulaite quetrion
------------------
10.0 une cesonte asscofigarime a pa lonne quetrion a leauvous sput de mapeuc qu une cesonte scevite a pa gaumaite quetrion
------------------
11.0 une cesonte asscofimarile a pa donne quetrion a deauvous sput be lapeuc qu une cesonte scevite a pa maulaite quetrion
------------------
12.0 une cesonte asscohiparime a la donne quetrion a deauvous slut je maleuc qu une cesonte scevite a la paumaite quetrion
------------------
13.0 une cesonte asscofivarile a pa monne quetrion a meaudous sput he lapeuc qu une cesonte scedite a pa vaulaite quetrion
```

(Figure 15) Exemple de résultats de l'algorithme MCMC (avec matrice A_1)

On remarque que les lettres fixées sont les mêmes pour les deux figures et que l'algorithme tend vers les mêmes zones de l'espace d'état E quelque soit la matrice considérée. On en déduit que le biais de la matrice de transition n'a vraisemblablement pas de gros impact sur l'algorithme. On notera aussi que les mots "valeur" et "precise" ne sont pas très probables avec le calcul de probabilité présenté à la question 5.

Conclusion

L'algorithme MCMC explicité dans les paragraphes précédents est un bon outil qui nous a permis de déchiffrer partiellement le message codé. L'exactitude de la matrice de transition considérée n'a finalement que peu d'impact sur l'algorithme. Le modèle utilisé pour calculer la probabilité qu'une chaîne de caractères soit en langue française pourrait être discuté.
Le message une fois déchiffré est : "Une réponse approximative à la bonne question a beaucoup plus de valeur qu'une réponse précise à la mauvaise question".