Inversion numérique de fonctions de répartition

MAT-3600 Projet de fin d'études
Sous la supervision de Philippe-André Luneau
et Jean Deteix

Julien April
536 860 609

Baccalauréat en Mathématiques

Département de mathématiques et de statistique
Université Laval, Québec
21 avril 2023

Résumé

La fonction de répartition d'une variable aléatoire réelle X est la fonction F_X qui, à tout réel x, associe la probabilité que X prenne une valeur inférieure ou égale à x. Cette fonction est caractéristique de la loi de probabilité de cette variable aléatoire réelle. En statistique, on a parfois besoin de réaliser des simulations. On peut notamment y tester et y comparer des estimateurs statistiques. Toutefois, ces simulations nécessitent généralement l'obtention de nombres aléatoires d'une certaine distribution, pas nécessairement uniforme. Or, avec la simple observation que $X = F_X^{-1}(U)$, où U est la loi uniforme sur $[0, 1]$, on est en mesure de générer des échantillons de réalisations de la variable X à condition de connaître sa fonction de répartition.

Dans le cas particulier des variables aléatoires réelles non-uniformes, il arrive que la fonction de répartition inverse ne puisse pas être exprimée en termes de fonctions élémentaires. Des méthodes numériques comme la méthode de Newton et l'interpolation linéaire peuvent alors être utilisées pour en faire l'approximation. Toutefois, ces méthodes présentent plusieurs inconvénients. Les splines cubique d'Hermite semblent en fait mieux convenir à la problématique, car il s'agit d'une méthode locale nécessitant relativement peu d'évaluations de la fonction de répartition. Dans ce travail, la méthode d'inversion numérique par interpolation dévelopée par Hörmann et Leylold [5] et son ordre de convergence sont étudiées. L'algorithme est également implémenté en C++. Les fichiers du programme sont disponibles à l'adresse suivante: `https://github.com/julapril/PFE_Inv_num_fdr.git`

Table des matières

1 Introduction **1**

1.1 Quelques notions de statistiques 1

1.2 Motivations 3

2 Méthode de la transformée inverse **5**

2.1 Définitions et exemples 5

2.2 Considérations générales 7

2.3 Inversion numérique par méthodes itératives 9

2.4 Inversion numérique par interpolation 10

2.5 Interpolation cubique d'Hermite de F^{-1} 13

2.6 Choix des sous-intervalles 15

2.7 Monotonicité 17

2.8 Densité s'annulant sur le domaine 20

3 Analyse de l'erreur d'interpolation théorique **21**

3.1 Retour sur l'erreur-u 21

3.2 Borne d'erreur théorique 23

4 Algorithme **25**

4.1 Développement 25

4.2 Implémentation 25

5 Résultats empiriques **29**

6 Conclusion **33**

Références **34**

A Résultats de l'article étudiée **36**

1 Introduction

On introduit d'abord quelques notions de statistiques [7] avant de motiver le projet.

1.1 Quelques notions de statistiques

Définition 1.1.1. Une *variable aléatoire* X est une fonction définie sur l'ensemble des résultats possibles d'une expérience aléatoire Ω vers l'ensemble des nombres réels.

Définition 1.1.2. La *fonction de répartition* d'une variable aléatoire X, abréviée par fdr, est définie, pour tout $x \in \mathbb{R}$, par

$$F_X(x) \overset{\text{déf}}{=} \mathbb{P}(\{\omega \mid X(\omega) \leq x\}),$$

où $\mathbb{P}$ est un application définie sur l'ensemble des parties de Ω à valeurs dans $[0, 1]$ telle que $\mathbb{P}(\Omega) = 1$. Pour simplifier l'écriture de la fonction de répartition, on écrit $F_X(x) = \mathbb{P}(X \leq x)$ pour la suite.

La proposition suivante donne une propriété importante des fonctions de répartition. Elle est acceptée sans démonstration.

Proposition 1.1.3. *La fonction de répartition d'une variable aléatoire X est croissante.*

Définition 1.1.4. Soit X une variable aléatoire continue, c'est-à-dire une variable aléatoire pouvant prendre un nombre infini de valeurs réelles, de fdr F_X. La *fonction de densité* de X est définie par

$$f_X(x) \overset{\text{déf}}{=} \frac{d}{dx} F_X(x).$$

Lorsque le contexte est clair, on peut laisser tomber l'indice X pour alléger la notation. Voici maintenant une méthode pour trouver la loi d'une fonction d'une variable aléatoire, qui sera utile pour la démonstration du théorème 2.1.1.

Règle 1.1.5 (Méthode de la fonction de répartition)**.** Soit une variable aléatoire X et $g : \mathbb{R} \to \mathbb{R}$ une fonction donnée. On pose $Y = g(X)$. Alors, la fonction de répartition de Y est

$$F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(g(X) \leq y) = \mathbb{P}(X \in \{x \mid g(x) \leq y\}).$$

Puisqu'on utilise la loi uniforme à de multiples reprises dans ce travail, on la définit ci-dessous.

Définition 1.1.6. On définit la loi uniforme sur l'intervalle (a, b), notée $U(a, b)$, par la fonction de densité

$$f_{U(a,b)} = \frac{1}{b-a} \mathbb{1}_{(a,b)}(x),$$

où $\mathbb{1}_{(a,b)}$ est la fonction indicatrice sur (a, b).

Remarque 1.1.7. Avec les définitions 1.1.2 et 1.1.4, on trouve

$$F_{U(a,b)} = \frac{x}{b-a} \mathbb{1}_{(a,b)}(x).$$

Remarque 1.1.8. On utilisera principalement le cas particulier de la loi uniforme $U(0, 1)$, en raison de son lien avec les fonctions de répartition qu'on montre au théorème 2.1.1.

1.2 Motivations

Étudions maintenant un exemple tiré du chapitre 2 de l'ouvrage *Modélisation et évaluation quantitative des risques en actuariat*, d'Etienne Marceau. [8]

En actuariat, on est souvent intéressé à décrire le comportement aléatoire d'un risque, représenté par une variable aléatoire X. En effet, cela s'avère très utile dans des domaines tels que les assurances automobile, habitation et maladie. Dépendamment du contexte d'application et du type de risque, on pourra choisir des approches différentes pour modéliser le risque. Voici un exemple de modèle général pour X.

Définition 1.2.1. On définit la v.a. X par une somme aléatoire

$$X = \begin{cases} \sum_{k=1}^{M} B_k, & \text{si } M > 0 \\ 0, & \text{si } M = 0 \end{cases},$$

où M est une v.a. représentant le nombre de sinistres et B_k une v.a. positive représentant le montant du $k^{\text{ième}}$ sinistre. On suppose que les v.a. $B_1, B_2, \ldots, B_M$ sont indépendantes et identiquement distribuées.
On dit alors que X suit une *loi composée*. M est appelée *v.a. de fréquence* et B_k est appelé *sévérité*.

Avec cette définition, on peut montrer le résultat suivant. [8]

Proposition 1.2.2. *Si on note par F_X la fonction de répartition de X et par p_M la fonction de probabilité de M, on a:*

$$F_X(x) = p_M(0) + \sum_{k=1}^{\infty} p_M(k) F_{B_1+\cdots+B_k}(x)$$

$$f_X(x) = \sum_{k=1}^{\infty} p_M(k) f_{B_1+\cdots+B_k}(x)$$

Exemple 1.2.3 (Loi de Poisson composée de sévérité Gamma)**.** Supposons que $M \sim Poisson(\lambda)$ et $B_k \sim Gamma(\alpha, \beta)$. Alors,

$$X = \begin{cases} \sum_{k=1}^{M} B_k, & \text{si } M > 0 \\ 0, & \text{si } M = 0 \end{cases}$$

suit une loi de Poisson composée de sévérité Gamma.
Comme $B_1 + \cdots + B_k \sim Gamma(k\alpha, \beta)$, on a :

$$F_X(x) = e^{-\lambda} + \sum_{k=1}^{\infty} \frac{e^{-\lambda}\lambda^k}{k!} \frac{\gamma(k\alpha, \beta x)}{\Gamma(k\alpha)} \quad \text{et} \quad f_X(x) = \sum_{k=1}^{\infty} \frac{e^{-\lambda}\lambda^k}{k!} \frac{\beta^{k\alpha} x^{k\alpha-1} e^{-\beta x}}{\Gamma(k\alpha)}$$

où $\gamma(n, x)$ est la fonction gamma incomplète.

Obtenir des résultats analytiques sur la fiabilité d'un modèle statistique comme celui du dernier exemple peut être difficile, voire impossible. Dans ces cas, la simulation statistique peut permettre d'obtenir des résultats empiriques. Elle consiste à générer des résultats aléatoires selon le modèle statistique et à analyser les résultats obtenus. Ainsi, lorsqu'on réalise des simulations, on a besoin d'une façon de générer des nombres aléatoires suivant une loi donnée.

Afin d'atteindre cet objectif, on étudie la méthode de la transformée inverse par interpolation développée dans l'article *Continuous Random Variate Generation by Fast Numerical Inversion* de Wolfang Hörmann et Josef Leylold (2003). [5]

2 Méthode de la transformée inverse

Dans ce deuxième chapitre, on introduit la méthode de la transformée inverse. Puis, on considère les contraintes particulières du problème de génération de nombres suivant des lois non uniformes.

2.1 Définitions et exemples

La méthode de la transformée inverse se base sur ce résultat.[11]

Théorème 2.1.1. *Soit $F(x)$ une fonction de répartition continue et U une variable aléatoire de loi uniforme sur l'intervalle $(0,1)$. Alors, la fonction de répartition de la variable aléatoire $X = F^{-1}(U)$ est F. De plus, si la fonction de répartition de la variable aléatoire X est F, alors $F(X)$ suit une loi uniforme sur $(0,1)$.*

Démonstration. On utilise la méthode 1.1.5. Calculons d'abord la fdr de $F^{-1}(U)$. Pour tout $x \in \mathbb{R}$,

$$\mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x).$$

La première égalité suit du fait que F est croissante, par la proposition 1.1.3. Cela montre que la fdr de $F^{-1}(U)$ est F. Calculons maintenant la fonction

de répartition de $F(X)$.

$$\begin{aligned}\mathbb{P}(F(X) \leq u) &= \begin{cases} 0 & \text{si } u \leq 0 \\ \mathbb{P}(X \leq F^{-1}(u)) & \text{si } 0 < u < 1 \\ 1 & \text{si } u \geq 1. \end{cases} \\ &= \begin{cases} 0 & \text{si } u \leq 0 \\ F(F^{-1}(u)) & \text{si } 0 < u < 1 \\ 1 & \text{si } u \geq 1. \end{cases} \\ &= \begin{cases} 0 & \text{si } u \leq 0 \\ u & \text{si } 0 < u < 1 \\ 1 & \text{si } u \geq 1. \end{cases}\end{aligned}$$

Notons qu'on a l'égalité $\mathbb{P}(F(X) \leq u) = \mathbb{P}(X \leq F^{-1}(u))$ lorsque $0 < u < 1$, encore une fois par la proposition 1.1.3. On reconnaît la fonction de répartition de la loi uniforme sur $(0, 1)$. Cela montre que $F(X) \sim U(0, 1)$.

□

Ce résultat mène à un algorithme, appelé méthode de la transformée inverse, pour générer une réalisation d'une variable aléatoire X de fonction de répartition $F(X)$. Voici l'algorithme dans sa forme la plus simple.

Algorithme 2.1 Méthode de la transformée inverse

Entrée: L'inverse de la fonction de répartition $F^{-1}(u)$.

Sortie: Une réalisation x de la variable aléatoire X de fdr F.

1: Générer une réalisation u de loi $U(0, 1)$.
2: Poser $x \leftarrow F^{-1}(u)$.
3: **Retourner** x

Exemple 2.1.2. Appliquons cet algorithme à la loi exponentielle.
Soit $X \sim \mathcal{E}(\theta)$. On a que $F(x) = (1 - e^{-\theta x})\mathbb{1}_{\mathbb{R}_+}(\theta)$. On peut calculer directement F^{-1}. On obtient $F^{-1}(u) = -\frac{1}{\theta} log(1-u)$. On génère un u de loi $U(0, 1)$, puis on calcule $F^{-1}(u)$. Le résultat correspond, par le théorème 2.1.1, à une

réalisation de X.

Exemple 2.1.3. Tentons d'appliquer cet algorithme à la loi de Poisson composée de sévérité Gamma. À l'exemple 1.2.3, on avait obtenu la fonction de répartition de cette loi.

$$F_X(x) = e^{-\lambda} + \sum_{k=1}^{\infty} \frac{e^{-\lambda}\lambda^k}{k!} \frac{\gamma(k\alpha, \beta x)}{\Gamma(k\alpha)}$$

Dans ce cas, il est impossible d'obtenir une expression analytique de F^{-1}. On ne peut donc pas appliquer directement l'algorithme.

Malheureusement, en général, il est rare qu'on puisse calculer directement la fonction de répartition inverse. C'est le cas aussi pour des lois plus simples, comme les lois normales, gamma et beta. On peut alors utiliser des méthodes numériques pour approximer F^{-1}.

2.2 Considérations générales

Pour la suite du rapport, on travaillera sous les hypothèses suivantes.

Hypothèses 2.2.1. On suppose que :

(H_1) On souhaite générer plusieurs réalisations d'une variable aléatoire continue X de distribution donnée.

(H_2) On connaît la fonction de répartition F et la fonction de densité f de cette distribution.

(H_3) L'ensemble des valeurs que peut prendre X, appelé support de X, est borné.

(H_4) On a à notre disposition une source de nombres aléatoires uniformes de résolution limitée.

On donne maintenant une série de remarques utiles pour le reste du rapport.

Remarque 2.2.2. Puisque X est une variable continue (H_1), on sait que F est continue et croissante.

Remarque 2.2.3. En pratique, on ne connaît pas toujours la fonction de répartition F et la fonction de densité f. En effet, on a souvent seulement accès à la fonction de densité. Cependant, on peut en déduire la fonction de répartition, car la définition 1.1.4 de la densité implique que

$$F(x) = \int_{-\infty}^{x} f(t)dt.$$

Ainsi, on peut approximer la fdr à l'aide de l'intégration numérique. Cependant, la procédure devient en général très coûteuse lorsqu'on doit le faire pour un grand nombre de valeurs de x. C'est en fait le principal désavantage de la méthode de résolution numérique traitée dans la section 2.3.

Remarque 2.2.4. En pratique, (H_3) n'est pas toujours respectée. Cependant, pour des distribution avec des domaines non bornés, il est possible d'ignorer les queues des distributions. Il suffit en effet de choisir a_0 et b_0 tels que $F(a_0)$ et $1 - F(b_0)$ soient petits relativement à l'erreur d'approximation tolérée. Par exemple, si X suit la loi normale centrée réduite, le support de X correspond aux réels. Toutefois, si on prend $a_0 = -6$ et $b_0 = 6$, on obtient $F(a_0) = 1 - F(b_0) = 9.865877e - 10$, ce qui est de l'ordre de la précision machine.

Plusieurs pistes sont désormais possibles pour réaliser l'objectif de générer des nombres aléatoires de sorte à optimiser la précision et/ou le temps d'exécution et/ou la mémoire utilisée. Dans un premier temps, on affrontera les méthodes itératives et l'interpolation. Puis, on comparera les types d'interpolation et de sous-intervalles pouvant être utilisés. Notons au passage qu'il existe d'autres méthodes pour générer des nombres aléatoires de lois non-uniformes

qui ne se basent pas sur la méthode de la transformée inverse. Elles ne seront pas étudiées dans ce travail. Cependant, l'ouvrage de Devroye [4] ainsi que celui de Hörmann, Leylold et Derflinger [11] présentent certaines de ces méthodes.

2.3 Inversion numérique par méthodes itératives

En analyse numérique, les méthodes itératives sont des techniques pour résoudre des problèmes mathématiques en effectuant des itérations successives jusqu'à ce qu'une solution suffisamment précise soit obtenue. Lorsqu'il est difficile ou impossible de calculer l'inverse de la fonction de répartition, on peut utiliser de telles méthodes pour obtenir une approximation de $F^{-1}(u)$ à l'étape 2 de l'algorithme 2.1.1. On tente en fait de résoudre l'équation:

$$F(x) = u.$$

Pour trouver une solution approximative, Hörmann, Leylold et Derflinger [11] suggèrent d'utiliser la méthode *regula falsi*, car elle s'avère plus stable que la méthode de Newton dans la plupart des cas. Cependant, si la fonction de répartition n'est pas lisse, *regula falsi* peut converger très lentement. Il est alors proposé de recourir à la méthode de bissection dans les cas où *regula falsi* converge lentement. Une convergence au moins linéaire est ainsi garantie.

Bien qu'elle soit simple et qu'elle requiert peu de mémoire, l'inversion par méthodes itératives présente des désavantages importants. D'abord, elle nécessite un grand nombre d'évaluations de la fdr, ce qui est assez coûteux pour la plupart des distributions standards comme souligné dans la remarque 2.2.3. Aussi, elle peut converger lentement lorsque la pente de la fdr est très

élevée et qu'on doit utiliser la méthode de la bissection. On propose d'utiliser l'interpolation pour obtenir une méthode plus rapide.

2.4 Inversion numérique par interpolation

L'inversion numérique par interpolation se distingue de la résolution numérique par méthodes itératives. Plutôt que de résoudre l'équation $F(x) = u$ pour tous les u générés, on tente de construire une approximation de l'inverse de la fonction de répartition, F^{-1}. Puis, pour chaque u généré, il suffit d'évaluer la fonction approximative construite pour obtenir les réalisations x de fdr F. Voyons un peu plus en détails comment cette méthode fonctionne.

D'abord, considérons le problème d'interpolation. Étant donné un grand nombre de points de $(u_i = F(p_i), p_i)$, où $a = p_0 < p_1 < \cdots < p_N = b$, on tente de construire une approximation $H(u)$ de $F^{-1}(u)$. On obtient ainsi l'algorithme d'interpolation ci-dessous.

Algorithme 2.2 Méthode de la transformée inversée par interpolation

Entrée: Un ensemble de couples $(u_i = F(p_i), p_i)$;

Sortie: Une réalisation x de la variable aléatoire X de fdr F.

1: Construire une interpolation $H(u)$ de $F^{-1}(u)$ avec les couples $(u_i = F(p_i), p_i)$.
2: Générer une réalisation u de loi $U(0, 1)$.
3: Poser $x \leftarrow H(u)$.
4: **Retourner** x

Bien sûr, on souhaite le faire en contrôlant l'erreur d'interpolation maximale, soit la quantité:

$$\epsilon_x = \max_{u \in [0,1]} D(u), \text{ où } D(u) = |H(u) - F^{-1}(u)|.$$

Cependant, ce calcul nécessite d'évaluer F^{-1} par méthodes itératives ce qui est coûteux en général (section 2.3). C'est pourquoi on considère plutôt l'erreur-u maximale définie par:

$$\epsilon_u = \max_{u \in [0,1]} E(u), \text{ où } E(u) = |F(H(u)) - u|.$$

Les erreurs $D(u)$ et $E(u)$ sont représentées à la figure 2.1. On explique pourquoi on peut utiliser l'erreur-u au lieu de l'erreur d'interpolation à la section 3.1.

Plusieurs types d'interpolation sont disponibles pour construire une approximation de F^{-1} étant donné des points $(u_i = F(p_i), p_i)$. Considérons d'abord l'interpolation polynomiale. Cette méthode consiste à trouver un polynôme passant par tous les points (u_i, p_i). Ce polynôme sera ainsi de degré très élevé, de l'ordre du nombre de points. Ce type d'interpolation n'est cependant pas approprié pour notre problème, car des phénomènes d'oscillations peuvent faire exploser l'erreur (voir *phénomène de Runge* [1]).

Pour remédier à cette problématique, on peut utiliser l'interpolation par splines générales. Il s'agit d'une interpolation par morceaux à l'aide de polynômes de bas degré. On s'assure que les dérivées de la fonction obtenue sont, jusqu'à un certain ordre n dépendant du degré des polynômes interpolant, continues. Par exemple, si $n = 0$, il s'agit de l'interpolation linéaire et les splines seront des droites. Tandis que si $n = 2$, on parle d'interpolation par splines cubiques. La principale problématique de l'interpolation par splines est qu'il s'agit d'une approximation globale, au sens où si on souhaite améliorer l'approximation en ajoutant un point (u_k, p_k) tel que $u_i < u_k < u_{i+1}$, on devra recalculer toutes les splines. On remarque aussi que les dérivées de la fonction aux points d'interpolation ne correspondront pas nécessairement aux dérivées de F^{-1} en ces mêmes points.

Remarque 2.4.1. Pour tout $0 < i < N$, on a

$$(F^{-1})'(u_i) = \frac{1}{F'(F^{-1}(u_i))} = \frac{1}{F'(p_i)} = \frac{1}{f(p_i)}$$

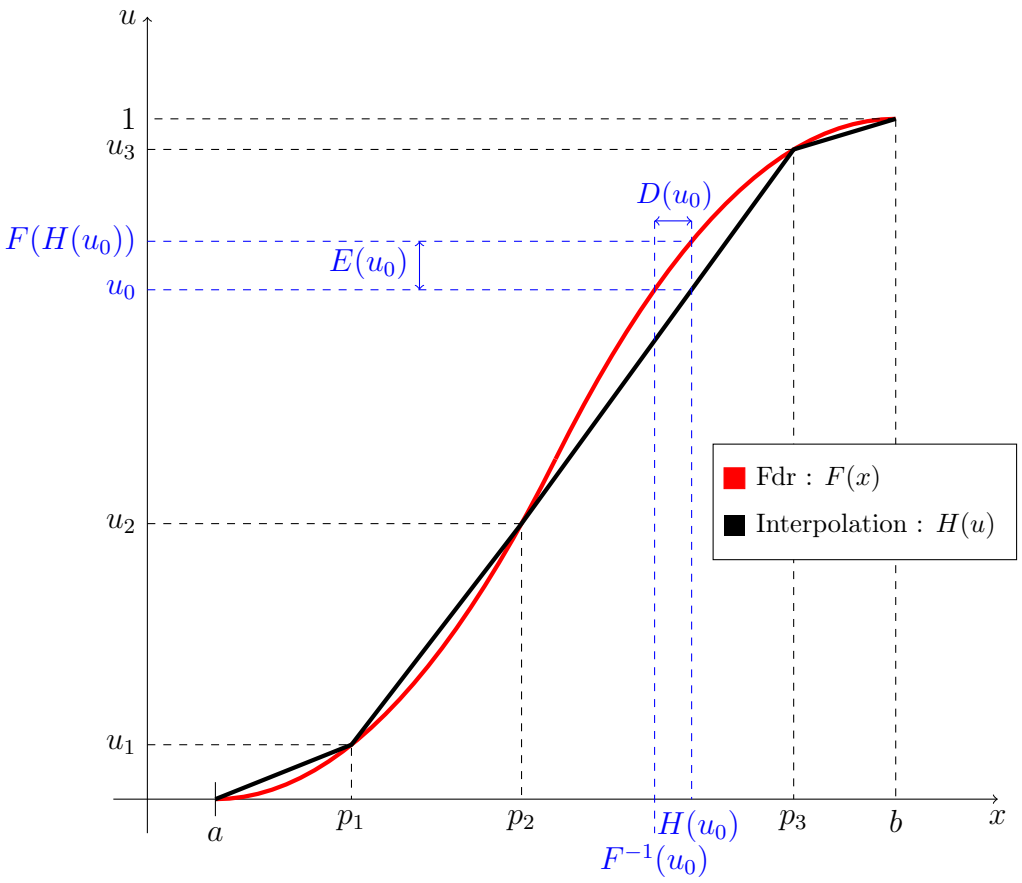

Figure 2.1: Représentation de l'erreur d'interpolation $D(u_0)$ et de l'erreur-u $E(u_0)$ pour un $u_0 \in [0, 1]$. L'interpolation linéaire est utilisée.

Or, on connaît f. Il est donc facile d'évaluer $(F^{-1})'(u_i)$. Il paraît donc sensé d'utiliser cette information pour améliorer la qualité de l'approximation. Cela nous amène à considérer l'interpolation d'Hermite.

2.5 Interpolation cubique d'Hermite de F^{-1}

L'interpolation d'Hermite, comme l'interpolation par splines générales, correspond à une interpolation de bas degré par morceaux. On considère ici les triplets $(u_i = F(p_i), p_i, \frac{1}{f(p_i)})$ qui, selon la remarque 2.4.1, correspondent aux triplets $(u_i, F^{-1}(u_i), (F^{-1})'(u_i))$. Pour chaque sous-intervalle $[u_i, u_{i+1}]$, on fera ainsi correspondre le polynôme cubique C_i tel que

$$\begin{aligned} C_i(u_i) &= F^{-1}(u_i) = p_i, & C_i'(u_i) &= (F^{-1})'(u_i) = \frac{1}{f(p_i)}, \\ C_i(u_{i+1}) &= F^{-1}(u_{i+1}) = p_{i+1}, & C_i'(u_{i+1}) &= (F^{-1})'(u_{i+1}) = \frac{1}{f(p_{i+1})}. \end{aligned}$$

En général, puisqu'on a un système à quatre équations et quatre inconnues (les coefficients de C_i), le polynôme C_i existe et est unique. En utilisant ces conditions, on trouve:

$$\begin{aligned} C_i(u) =& (1-\tilde{u})^2(1+2\tilde{u})p_i + \tilde{u}^2(3-2\tilde{u})p_{i+1} \\ &+ (u_{i+1}-u_i)\left(\frac{(1-\tilde{u})^2\tilde{u}}{f(p_i)} + \frac{\tilde{u}^2(\tilde{u}-1)}{f(p_{i+1})}\right) \end{aligned}$$

où

$$\tilde{u} = \frac{u-u_i}{u_{i+1}-u_i}$$

envoie $u \in [u_i, u_{i+1}]$ vers $[0,1]$. Cela peut être réécrit sous la forme:

$$C_i(u) = \sum_{j=0}^{3} a_{ij}\tilde{u}^j = a_{i0} + a_{i1}\tilde{u} + a_{i2}\tilde{u}^2 + a_{i3}\tilde{u}^3 \tag{2.1}$$

avec

$$
\begin{aligned}
a_{i0} &= p_i, \\
a_{i1} &= \frac{u_{i+1} - u_i}{f(p_i)}, \\
a_{i2} &= 3(p_{i+1} - p_i) - (u_{i+1} - u_i)\left(\frac{2}{f(p_i)} + \frac{1}{f(p_{i+1})}\right), \\
a_{i3} &= 2(p_i - p_{i+1}) + (u_{i+1} - u_i)\left(\frac{1}{f(p_i)} + \frac{1}{f(p_{i+1})}\right).
\end{aligned}
$$

Pour l'interpolation par splines générales, C_i dépendait de C_{i-1} et/ou de C_{i+1}, car on devait avoir la continuité des dérivées, d'où la qualification d'interpolation globale. Toutefois, pour l'interpolation cubique d'Hermite, C_i est complètement indépendant des autres C_j. C'est pourquoi l'interpolation d'Hermite est dite locale. Cela s'avère particulièrement utile lorsqu'on souhaite améliorer l'approximation en ajoutant un point (u_k, p_k) tel que $u_i < u_k < u_{i+1}$. En effet, on devra seulement recalculer deux polynômes.

Remarquons qu'on peut améliorer la qualité de l'approximation en utilisant $f'(p_i)$ et $f'(p_{i+1})$, en plus de $F(p_i)$, $F(p_{i+1})$, $f(p_i)$ et $f(p_{i+1})$. En effet, on peut montrer que $(F^{-1}(u_i))'' = -f'(p_i)/f(p_i)^3$. On obtient alors un système de 6 équations à 6 inconnues auquel un unique polynôme quintique satisfait. Il s'agit de l'interpolation quintique d'Hermite. En général, on ne peut pas évaluer exactement f'. Ainsi, si elle n'est pas donnée, il faudra qu'elle soit approximée à l'aide de méthodes numériques.

En raison de sa simplicité et de son approximation locale, on utilisera l'interpolation d'Hermite dans le reste du travail. On étudiera principalement sa version cubique, mais on se penchera également sur ses versions linéaire et quintique.

2.6 Choix des sous-intervalles

Jusqu'à maintenant, on supposait que des triplets $(u_i = F(p_i), p_i, \frac{1}{f(p_i)})$ nous étaient donnés. Toutefois, en pratique, on choisit quels (u_i, p_i) on utilise pour effectuer l'interpolation. La méthode la plus simple consiste à choisir des points p_i tels que $|u_{i+1} - u_i|$ est constant. Cependant, ce choix demande de calculer les $p_i = F^{-1}(u_i)$ de façon très précise. Cela doit être réalisé via des méthodes numériques itératives, processus généralement coûteux. On rappelle qu'on souhaite être en mesure de contrôler l'erreur-u maximale

$$\epsilon_u = \max_{u \in [0,1]} E(u), \text{ où } E(u) = |F(H(u)) - u|.$$

Supposons que $\overline{\epsilon}_u$ soit l'erreur maximale tolérée. Si on souhaite borner ϵ_u sur $[0, 1]$, on doit trouver un moyen de borner ϵ_u sur chacun des intervalles $[u_i, u_{i+1}]$. Malheureusement, il est en pratique très difficile de le faire, puisque cette tâche nécessite la connaissance de maximums de dérivées d'ordre élevé (voir chapitre 3). Toutefois, Hörmann et Leylold proposent une alternative. Il s'agit d'utiliser

$$\hat{\epsilon}_u = |F(H_i(\overline{u})) - \overline{u}|, \text{ où } \overline{u} = \frac{u_i + u_{i+1}}{2}.$$

comme approximation de l'erreur-u maximale sur l'intervalle $[u_i, u_{i+1}]$. Cette approximation est généralement assez bonne lorsque la fdr est lisse et n'a pas de point d'inflexions sur cet intervalle. Il correspond effectivement au sens commun que l'erreur maximale sera environ atteinte au point milieu de l'intervalle.

On propose de choisir des points (u_i, p_i) tels que l'interpolation $H(u)$ respecte $\hat{\epsilon}_u < \overline{\epsilon}_u$ pour tout intervalle $[u_i, u_{i+1}]$. L'approche proposée dans l'article de Hörmann et Leylold est présentée par l'algorithme 2.3.

Expliquons ce qu'on signifie par «diviser récursivement l'intervalle $[a, b]$». On commence par bissecter récursivement cet intervalle jusqu'à ce que

Algorithme 2.3 Choix des intervalles

Entrée: La fonction de répartition $F(x)$;

Entrée: Le support de la variable aléatoire $[a, b]$;

Entrée: L'erreur-u maximale tolérée $\overline{\epsilon}_u$.

Sortie: Un ensemble de points p_i;

1: **tant que** $\hat{\epsilon}_u > \overline{\epsilon}_u$ **faire**

2: Diviser l'intervalle $[a, b]$ récursivement.

3: **fin tant que**

4: **Retourner** les p_i obtenus lors de la division.

$|u_{i+1} - u_i| = |F(p_{i+1}) - F(p_i)|$ soit plus petit qu'une valeur de tolérance donnée comme 0,05. Puis, on continue la division récursive tant que $\hat{\epsilon}_u > \overline{\epsilon}_u$.

Précisons maintenant ce qu'on entend par bissecter ou diviser l'intervalle $[p_i, p_{i+1}]$. En fait, on a plusieurs possibilités. On pourrait choisir $H(\overline{u})$ ou encore $(p_i + p_{i+1})/2$ comme point de bissection. La première option permet d'éliminer une évaluation de la fdr, car le calcul de $F(H(\overline{u}))$ est déjà effectué lors de l'évaluation de $\hat{\epsilon}_u$. Toutefois, il a été observé par Hörmann et Leylold que la deuxième option s'avère plus stable, particulièrement lorsque les intervalles sont encore relativement longs. Elle est donc retenue. Remarquons que, dans le cas de l'interpolation linéaire, les deux options s'équivalent.

Au chapitre 3, on montre que, pour des fonctions avec des «bonnes» densités, l'erreur converge rapidement vers zéro lorsque le nombre d'intervalles augmente. Cependant, l'approximation de l'erreur-u maximale par l'erreur au milieu de l'intervalle peut se révéler complètement erroné. C'est le cas lorsque la densité n'est pas «lisse» ou admet des extremums locaux, i.e. la fdr admet des point d'inflexions. Pour remédier à la situation, Hörmann et Leylold proposent d'inclure ces points comme entrées de l'algorithme. On peut également effectuer une vérification empirique de l'erreur-u pour un grand échantillon de variables $U(0, 1)$ et la comparer avec l'erreur-u maximale tolérée.

Pour les domaines non-bornés (voir remarque 2.2.4), on pourra donner un domaine initial très grand en ajoutant l'étape finale donnée par l'algorithme 2.4.

Algorithme 2.4 Rejet des queues de la distribution

Entrée: La fonction de répartition $F(x)$

Entrée: Le support de la variable aléatoire $[a, b]$;

Entrée: L'erreur-u maximale tolérée $\overline{\epsilon}_u$;

Entrée: Les p_i obtenus à l'algorithme 2.3

1: $m \leftarrow \max\{\ 0 \leq i \leq N\ \mid F(u_i) < 0.1 \cdot \hat{\epsilon}_u\} \cup \{-1\}$.

2: $M \leftarrow \min\{\ 0 \leq i \leq N\ \mid F(u_i) > 1 - 0.1 \cdot \hat{\epsilon}_u\} \cup \{N + 1\}$.

3: Rejeter tous les p_i tels que $i \leq m$ ou $i \geq M$.

4: **Retourner** les p_i non-rejetés.

Remarquons que cette étape est également nécessaire pour s'assurer que les cas où $f(a)$ ou $f(b)$ sont nuls ne perturbent pas l'algorithme. En effet, si $f(x) = F'(x) = 0$, alors $(F^{-1})'(F(x))$ n'est pas défini.

2.7 Monotonicité

Selon la remarque 2.2.2, F est croissante. Ainsi, F^{-1} est également croissante. Il est souhaitable que l'interpolation de F^{-1} conserve cette propriété. Pour l'interpolation linéaire, la croissance de l'interpolation est tautologique. Pour l'interpolation cubique d'Hermite, il peut arriver que certaines parties de l'interpolation ne soient pas croissantes. Il existe des conditions exactes, qui ne dépendent que des points d'interpolation, pour montrer que les C_i sont croissants. Puisqu'elle sont assez compliquées, on préfère utiliser une condition suffisante pour que les C_i soient croissants. Elle est due à de Boor et Swartz [2] et elle sera donnée après la proposition suivante tirée du travail de Huynh [6].

Proposition 2.7.1. *Soit C l'interpolation cubique d'Hermite d'une fonction croissante $g(x)$ et soient $x_0 < x_1 < \cdots < x_N$ les points d'interpolation. Alors, la condition suivante est suffisante pour que C soit croissante sur l'intervalle $[x_i, x_{i+1}]$*

$$g'(x_i) \leq 3\left(\frac{g(x_{i+1}) - g(x_i)}{x_{i+1} - x_i}\right) \quad et \quad g'(x_{i+1}) \leq 3\left(\frac{g(x_{i+1}) - g(x_i)}{x_{i+1} - x_i}\right) \tag{2.2}$$

Démonstration. Notons les données $g(x_i)$ et $g'(x_i)$ par g_i et g'_i respectivement. Le cas $g_i = g_{i+1}$ est trivial, puisque C est alors constante. Pour la suite, on suppose donc que $g_i \neq g_{i+1}$. On considère le changement linéaire de coordonnées suivant

$$\tilde{x} = \frac{x - x_i}{x_{i+1} - x_i} \text{ et } \tilde{y} = \frac{y - g_i}{g_{i+1} - g_i},$$

où (x, y) sont les coordonnées du système original. On remarque que le polynôme cubique C est transformé en le polynôme cubique

$$\widetilde{C}(\tilde{x}) = \widetilde{C(x)} = \frac{C(x) - g_i}{g_{i+1} - g_i}.$$

Il est clair, par la règle de dérivation en chaîne, que la croissance de $C(x)$ sur $[x_i, x_{i+1}]$ est équivalente à la croissance de $\widetilde{C}(\tilde{x})$ sur $[0, 1]$. Ainsi, on peut supposer, sans perte de généralité, que $x_i = 0$, $x_{i+1} = 1$, $g_i = 0$ et $g_{i+1} = 1$. Considérons les quatre paires

$$(g'_i, g'_{i+1}) = (0, 0), (0, 3), (3, 0), (3, 3).$$

et dénotons les polynôme cubique d'Hermite pour ces quatre paires par $H_{(0,0)}$, $H_{(0,3)}$, $H_{(3,0)}$ et $H_{(3,3)}$. Avec la construction pour le polynôme cubique d'Hermite 2.1, on obtient

$$H_{(0,0)}(x) = -2x^3 + 3x^2 \qquad H_{(0,3)}(x) = x^3$$
$$H_{(3,0)}(x) = x^3 - 3x^2 + 3x \qquad H_{(3,3)}(x) = 4x^3 - 6x^2 + 3x$$

On peut vérifier que ces quatres polynômes sont croissants sur $[0,1]$ et vérifient $H(0) = 0$ et $H(1) = 1$. Ainsi, chaque combinaison linéaire avec des coefficients positifs de ces polynômes sera aussi croissante sur $[0,1]$. De plus, si la somme des coefficients est égale à 1, la combinaison linéaire vérifie $H(0) = 0$ et $H(1) = 1$. Posons $(g'_i, g'_{i+1}) = (\alpha, \beta)$. Le but devient alors de trouver une combinaison linéaire $H_{(\alpha,\beta)}$ telle que $H'_{(\alpha,\beta)}(0) = \alpha$ et $H'_{(\alpha,\beta)}(1) = \beta$. Or, la condition 2.2 donne $0 \leq \alpha/3 \leq 1$ et $0 \leq \beta/3 \leq 1$. Les combinaisons linéaires suivantes sont donc croissantes

$$\begin{aligned} H_{(\alpha,0)} &= \frac{\alpha}{3} H_{(3,0)} + \left(1 - \frac{\alpha}{3}\right) H_{(0,0)} \\ H_{(\alpha,3)} &= \frac{\alpha}{3} H_{(3,3)} + \left(1 - \frac{\alpha}{3}\right) H_{(0,3)} \\ H_{(\alpha,\beta)} &= \frac{\beta}{3} H_{(\alpha,3)} + \left(1 - \frac{\beta}{3}\right) H_{(\alpha,0)} \end{aligned}$$

où un calcul montre que

$$\begin{aligned} H'_{(\alpha,0)}(0) = \alpha \text{ et } H'_{(\alpha,0)}(1) = 0 \\ H'_{(\alpha,3)}(0) = \alpha \text{ et } H'_{(\alpha,3)}(1) = 3 \\ H'_{(\alpha,\beta)}(0) = \alpha \text{ et } H'_{(\alpha,\beta)}(1) = \beta. \end{aligned}$$

Cela montre que, pour toute paire (g'_i, g'_{i+1}) respectant la condition 2.2, il existe une combinaison linéaire avec coefficients positifs de fonctions croissantes donnant le polynôme cubique d'Hermite interpolant ces points. Cette combinaison linéaire est aussi croissante et respecte $H(0) = 0$ et $H(1) = 1$, d'où le résultat.

□

Cette dernière proposition nous indique une condition suffisante pour que C_i soit croissant sur $[u_i, u_{i+1}]$, soit

$$\frac{1}{f(u_i)} \leq 3\left(\frac{p_{i+1} - p_i}{u_{i+1} - u_i}\right) \quad \text{et} \quad \frac{1}{f(u_{i+1})} \leq 3\left(\frac{p_{i+1} - p_i}{u_{i+1} - u_i}\right).$$

On peut alors diviser les intervalles où cette condition n'est pas respectée, jusqu'à ce qu'elle le soit. D'ailleurs, le raisonnement suivant montre que ce processus termine toujours si la densité f est continue. Effectuons le développement de Taylor de F^{-1} en u_i. Puisque f est supposée continue, F^{-1} est continûment dérivable et on obtient

$$\begin{aligned} & F^{-1}(u_{i+1}) = F^{-1}(u_i) + (F^{-1})'(u_i) \cdot (u_{i+1} - u_i) + \mathcal{O}((u_{i+1} - u_i)^2) \\ \Longrightarrow \quad & (F^{-1})'(u_i) = \frac{F^{-1}(u_{i+1}) - F^{-1}(u_i)}{u_{i+1} - u_i} + \mathcal{O}(u_{i+1} - u_i) \\ \Longrightarrow \quad & (F^{-1})'(u_i) = \frac{p_{i+1} - p_i}{u_{i+1} - u_i} + \mathcal{O}(u_{i+1} - u_i) \end{aligned}$$

Cela montre que, pour un intervalle $[u_i, u_{i+1}]$ assez petit, on aura $\frac{1}{f(u_i)} \leq 3\left(\frac{p_{i+1}-p_i}{u_{i+1}-u_i}\right)$. Similairement pour $\frac{1}{f(u_{i+1})}$. En conclusion, on a montré qu'on pourra trouver une approximation cubique d'Hermite croissante C de F^{-1}.

Pour l'interpolation quintique d'Hermite, il existe un résultat similaire. Il est brièvement présenté dans l'article de Hörmann et Leylold. [5]

2.8 Densité s'annulant sur le domaine

Les interpolations cubiques et quintiques d'Hermite ne fonctionnent pas si $f(x) = 0$ en certains points du domaine, car si $f(x) = F'(x) = 0$, alors $(F^{-1})'(F(x))$ n'est pas défini. Une façon de contourner ce problème est d'utiliser l'interpolation linéaire. Par exemple, si $f(p_i) = 0$ ou $f(p_{i+1}) = 0$ pour un intervalle $[p_i, p_{i+1}]$, on utilise l'interpolation linéaire, i.e. on pose $a_{i2} = a_{i3} = 0$ dans la forme 2.1.

3 Analyse de l'erreur d'interpolation théorique

3.1 Retour sur l'erreur-u

Les algorithmes développés jusqu'à maintenant pour l'inversion numérique par interpolation prenait en compte l'erreur-u maximale. Dans cette section, il est expliqué pourquoi il s'agit d'une mesure d'erreur valide. On propose de quantifier l'erreur d'une interpolation par une mesure de la disparité de $F(H(U))$ qu'on appelle écart, telle que définie ci-dessous.

Définition 3.1.1. L'écart (*discrepancy*) D_n d'une suite finie $x_0, x_1, x_2, \ldots, x_n \in [0,1]$ est défini par

$$D_n(x_1, \ldots, x_n) = \sup_{I \subset [0,1]} \left| \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_I(x_i) - l(I) \right|,$$

où I est un intervalle, $l(I)$ est la longueur de l'intervalle I et $\mathbb{1}_I$ est la fonction indicatrice sur I.

Une suite est donc équidistribuée sur $[0,1]$ si $D_n \to 0$ lorsque $n \to \infty$.

La lemme suivant permet de lier l'écart et l'erreur-u. Il correspond au lemme 2.5 de Niederreiter [9] et est accepté sans démonstration.

Lemme 3.1.2. *Si* $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n \in [0,1]$ *respectent* $|x_i - y_i| \leq \epsilon$

pour tout $i = 1, 2, \ldots, n$, *alors*

$$|D_n(x_1, \ldots, x_n) - D_n(y_1, \ldots, y_n)| \leq 2\epsilon$$

On peut en déduire le corollaire suivant.

Corollaire 3.1.3. *Soient* $u_1, u_2, \ldots, u_n$, *des nombres obtenus avec un générateur uniforme sur* $[0, 1]$. *Pour tout* $i = 1, 2, \ldots, n$, *posons* $w_i = F(H(u_i))$. *Supposons que* $|F(H(u_i)) - u_i| = |w_i - u_i| \leq \overline{\epsilon}_u$. *Alors,*

$$|D_n(w_1, \ldots, w_n) - D_n(u_1, \ldots, u_n)| \leq 2\overline{\epsilon}_u$$

De plus,

$$D_n(w_1, \ldots, w_n) \leq 2\overline{\epsilon}_u + D_n(u_1, \ldots, u_n)$$

Démonstration. La première inégalité suit directement du lemme, alors que la deuxième utilise aussi le fait que l'écart est toujours positif.

□

Dans la section précédente, on utilise l'approximation de l'erreur-u suivante

$$\hat{\epsilon}_u = \max_i |F(H_i(\overline{u})) - \overline{u}|, \text{ où } \overline{u} = \frac{u_i + u_{i+1}}{2}.$$

Ainsi, lorsque

$$\hat{\epsilon}_u \approx \epsilon_u = \max_{u \in [0,1]} |F(H(u)) - u| \leq \overline{\epsilon}_u$$

le corollaire donne l'inégalité

$$D_n(w_1, \ldots, w_n) \leq 2\overline{\epsilon}_u + D_n(u_1, \ldots, u_n)$$

Si l'écart du générateur uniforme, $D_n(u_1, \ldots, u_n)$, est connu, on peut facilement borner l'écart des $w_i = F(H(u_i))$. Autrement dit, lorsqu'on contrôle l'erreur-u maximale, on contrôle également l'écart des $F(H(u_i))$. Cela explique pourquoi on peut considérer l'erreur-u au lieu de l'écart des $F(H(u_i))$.

Finalement, remarquons que, si la tolérance $\bar{\epsilon}_u$ est plus petite que l'écart du générateur, alors l'écart des $F(H(u_i))$ sera borné par $3D_n(u_1, \ldots, u_n)$. Dans ce cas, on voit que la qualité de l'approximation obtenue dépend seulement de la qualité du générateur.

Il existe également un lien entre l'erreur d'interpolation ϵ_x et l'erreur-u ϵ_u. En effet, considérons un $u_0 \in [0, 1]$ et posons $x_0 = H(u_0)$. L'erreur dans la direction-x et l'erreur dans la direction-u en u_0 sont respectivement

$$\epsilon_x(u_0) = |x - F^{-1}(u_0)| \qquad \text{et} \qquad \epsilon_u(u_0) = |F(x) - u_0|.$$

Si on effectue le développement de Taylor de F^{-1} en u_0 en supposant que f est continue, c'est-à-dire que F^{-1} est continûment dérivable, on obtient

$$\begin{aligned} F^{-1}(u) - F^{-1}(u_0) &= (F^{-1})'(u_0)(u - u_0) + \mathcal{O}((u - u_0)^2) \\ &= \frac{1}{f(u_0)}(u - u_0) + \mathcal{O}((u - u_0)^2) \end{aligned}$$

En remplaçant u par $F(x)$ dans la dernière expression, on a alors

$$\begin{aligned} & x - F^{-1}(u_0) = \frac{1}{f(x_0)}(F(x) - u_0) + \mathcal{O}((F(x) - u_0)^2) \\ \implies \quad & \epsilon_x(u_0) = \frac{\epsilon_u(u_0)}{f(x_0)} + \mathcal{O}(\epsilon_u(u_0)^2) \end{aligned}$$

Cela montre que, tant que la densité est bornée, on observera les mêmes comportements asymptotiques pour ϵ_x et ϵ_u.

3.2 Borne d'erreur théorique

Ciarlet et al. [10] ont montré des bornes sur l'erreur d'interpolation lorsque l'interpolation d'Hermite est utilisée.

Pour l'interpolation linéaire et une fdr deux fois dérivable, on a la borne d'erreur suivante sur l'intervalle $[p_i, p_{i+1}]$

$$\epsilon_x \leq \frac{1}{32} \left| (u_{i+1} - u_i)^2 \max_{u \in [u_i, u_{i+1}]} (F^{-1})''(u) \right|$$

Ainsi, pour des points $u_1, u_2, \ldots, u_n$ équidistribuées, on a

$$\epsilon_x = \mathcal{O}(1/n^2)$$

Pour l'interpolation cubique d'Hermite et une fdr lisse sur l'intervalle $[p_i, p_{i+1}]$, on a la borne d'erreur similaire suivante

$$\epsilon_x \leq \frac{1}{384} \left| (u_{i+1} - u_i)^4 \max_{u \in [u_i, u_{i+1}]} (F^{-1})''''(u) \right|$$

Ainsi, pour des points $u_1, u_2, \ldots, u_n$ bien choisis, on a

$$\epsilon_x = \mathcal{O}(1/n^4)$$

Pour l'interpolation quintique d'Hermite, un résultat similaire montre que l'erreur d'approximation est d'ordre 6. On a montré, à la section précédente, que les comportements asymptotiques de ϵ_x et ϵ_u sont les mêmes si la densité est bornée. Cela permet de conclure que

$$\epsilon_u = \mathcal{O}(1/n^{d+1}),$$

où d représente l'ordre de l'interpolation d'Hermite utilisée. Cela montre aussi que l'écart des $F(H(u_i))$ converge à l'ordre $d + 1$.

4 Algorithme

4.1 Développement

En combinant les parties d'algorithme dégagées de la méthode de la transformée inverse par interpolation, on obtient l'algorithme complet ci-dessous.[5] Il est important de remarquer que, une fois l'interpolation complétée, on peut générer autant de nombres aléatoires que désirés, sans avoir à recalculer l'interpolation. Ainsi, cette méthode est particulièrement avantageuse lorsqu'on a beaucoup de nombres aléatoires à générer. En contraste avec les méthode itératives, l'interpolation requiert une étape lente d'initialisation, mais permet ensuite de générer des nombres aléatoires rapidement.

4.2 Implémentation

L'algorithme a été implémenté avec le langage C++. Il s'agit d'un langage de programmation orienté-objet de haut niveau. Les fichiers du programme sont disponibles à l'adresse suivante: `https://github.com/julapril/PFE_Inv_num_fdr.git` Le fichier principal est celui nommé *generation_ inversion.cpp*. Sa structure correspond à la charpente de l'algorithme 4.1. Deux parties de l'algorithme doivent être expliquées un peu plus en détails. Il s'agit de la structure de donnée utilisée et de la recherche lors de la génération.

D'abord, expliquons comment sont stockés les intervalles et les interpolations. La structure utilisée est une liste chaînée, car cette dernière doit

Algorithme 4.1 Méthode de la transformée inverse par interpolation

Entrée: La fdr $F(x)$, la densité $f(x)$ (cubique et quintique) et sa dérivée $f'(x)$ (quintique); l'ordre d'interpolation (1, 3 ou 5), la tolérance $\overline{\epsilon}_u$, le support $[a, b]$, les extremums locaux et discontinuités de la densité et de ses dérivées (optionnel).

Sortie: Une réalisation x de la variable aléatoire X de fdr F.

1: Initialisation d'une liste chaînée pour les points p_i, avec $p_0 = a$ et $p_1 = b$.
2: Diviser les intervalles en ajoutant les extremums locaux et discontinuités.
3: **tant que** $F(p_{i+1}) - F(p_i) > 0.05$ pour un certain i **faire**
4: Diviser l'intervalle $[p_i, p_{i+1}]$ récursivement.
5: **fin tant que**
6: **tant que** La fin de la liste n'est pas atteinte **faire**
7: Calculer l'interpolation $H_i(u)$ selon l'ordre. (section 2.5)
8: Calculer $\overline{u}$ et $\hat{\epsilon}_u$.
9: Vérifier si l'interpolation obtenue est monotone. (section 2.7)
10: **si** $\hat{\epsilon}_u < \overline{\epsilon}_u$ et H_i est monotone **alors**
11: Passer au prochain intervalle.
12: **sinon**
13: Diviser l'intervalle $[p_i, p_{i+1}]$ au point $\tilde{p} = (p_i + p_{i+1})/2$.
14: **fin si**
15: **fin tant que**
16: Rejeter les queues de la distribution. (algorithme 2.4)
17: Générer une réalisation u de loi $U(0, 1)$.
18: $j \leftarrow \max\{j : F_j(p_j) < u\}$. (Utiliser un index)
19: Poser $x \leftarrow H_j(u)$.
20: **Retourner** x

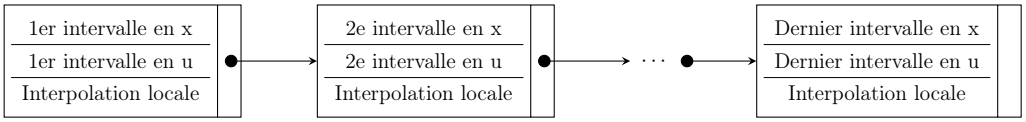

Figure 4.1: Représentation de la liste chaînée utilisée pour stocker l'interpolation d'Hermite de l'inverse de la fonction de répartition.

supporter une insertion rapide. En effet, le processus de division récursive nécessite beaucoup d'insertions de nouveaux éléments. Les noeuds de la liste contiendront toutes les informations de l'interpolation, soient l'intervalle en x, l'intervalle en u correspondant et les coefficidents de l'interpolation d'Hermite associée. Cette structure est représentée à la figure 4.1.

Lorsque l'interpolation est complétée, et que l'on souhaite générer beaucoup de nombres aléatoires, on doit chercher chaque polynôme d'interpolation dans la liste chaînée. Si on parcourt naïvement la liste pour chaque u généré, cela peut s'avérer très coûteux. En effet, le temps de recherche dans une liste est linéaire, c'est-à-dire qu'il dépend linéairement du nombre d'éléments de la liste. Il est plus intéressant d'utiliser un vecteur, car le temps de recherche devient alors constant. Après l'interpolation, on construit ainsi un vecteur nommé `index` et contenant C éléments, tel que la i-ème entrée indique dans quel noeud de la liste chaînée se trouve $u = i/C$. Puis, pour chaque u généré, il suffit de trouver le noeud correspondant à l'élément en position $\lfloor C \cdot u \rfloor$ de l'`index`. S'il ne s'agit pas du noeud contenant u, il suffira de rechercher dans les noeuds subséquents de la liste. Cette procédure sera, en général, beaucoup plus courte que la recherche linéaire. Des explications légèrement plus détaillées sont données à le section 3.1.2 de Hörmann et al. [11] Pour illustrer l'avantage, on a généré 1 million de nombres aléatoires de loi normale en utilisant l'interpolation linéaire et une tolérance sur l'erreur-u de 10^{-10} avec les deux types de recherche. On a obtenu une liste de 109678 noeuds et la génération avec une recherche linéaire a pris 2156 secondes, alors que

la recherche avec l'index, incluant l'initialisation de l'index, a nécessité 0.35 secondes.

Pour terminer, notons qu'une implémentation similaire est disponible au sein de *SciPy*, une librairie «source ouverte» scientifique utilisant le langage Python. Elle est disponible à l'adresse suivante:
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.sampling.NumericalInverseHermite.html`
Toutefois, elle n'a pas été étudiée ni comparée avec l'implémentation de ce travail.

5 Résultats empiriques

Il est maintenant intéressant de vérifier si les ordres de convergence théorique dégagés dans la partie 3 sont observés en pratique avec l'algorithme implémenté dans la partie 4. Donnons d'abord une façon de vérifier l'ordre de convergence d'une méthode. On a:

$$\begin{aligned} \epsilon_u = \mathcal{O}(1/n^d) \quad &\iff \quad \epsilon_u = \frac{C}{n^d}, \quad \text{pour une certaine constante } C \\ &\iff \quad \ln \epsilon_u = \ln C - d \ln n \end{aligned} \tag{5.1}$$

Autrement dit, l'erreur-u convergera à l'ordre d si, et seulement si, il existe une relation linéaire entre les $\ln n$ (vus comme les valeurs d'abscisse) et les $\ln \epsilon_u$ (vus comme les valeurs d'ordonnée) et que la pente de la droite correspondante est $-d$. Évidemment, on ne s'attend pas à obtenir des droites parfaites. Ainsi, on utilisera la méthode des moindres carrés pour estimer d.

Le tableau 5.1 montre le nombre d'intervalles obtenus lors de l'interpolation. Cela permet de calculer les $\ln \epsilon_u$ et les $\ln n$ pour chacune des distributions. Les résultats de la régression linéaire sont présentées au tableau 5.2. On constate que les valeurs de d pour les interpolations linéaire, cubique et quintique sont respectivement de 2, 4 et 6. Cela valide le résultat théorique selon lequel

$$\epsilon_u = \mathcal{O}(1/n^{d+1}),$$

où d représente l'ordre de l'interpolation d'Hermite utilisée.

Tableau 5.1: Nombres d'intervalles nécessaires pour obtenir l'erreur-u maximale demandée selon différentes distributions statistiques et le type d'interpolation.

Lois	$\overline{\epsilon}_u = 10^{-6}$	$\overline{\epsilon}_u = 10^{-8}$	$\overline{\epsilon}_u = 10^{-10}$	$\overline{\epsilon}_u = 10^{-12}$
Interpolation linéaire				
Loi normale	1088	11902	109680	967088
Loi de Cauchy	1896	18348	175706	1848206
Loi exponentielle	1043	10268	98513	1037659
Loi Gamma(5)	1098	11187	103125	1101641
Loi Gamma(1/2)	1547	15431	154251	1542484
Loi Beta(2,2)	824	8010	88180	828969
Loi Beta(0.3,3)	1885	18786	187786	1877790
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	714	7231	75246	715915
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	1064	11699	108918	1056287
Interpolation cubique				
Loi normale	100	320	1056	2770
Loi de Cauchy	188	504	1530	4630
Loi exponentielle	77	206	653	2080
Loi Gamma(5)	112	324	970	2896
Loi Gamma(1/2)	77	227	707	2227
Loi Beta(2,2)	96	254	782	2456
Loi Beta(0.3,3)	89	259	780	2395
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	90	255	717	2445
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	120	311	903	3166
Interpolation quintique				
Loi normale	64	110	214	532
Loi de Cauchy	124	192	372	762
Loi exponentielle	50	79	146	319
Loi Gamma(5)	66	115	236	495
Loi Gamma(1/2)	51	79	154	330
Loi Beta(2,2)	64	104	208	440
Loi Beta(0.3,3)	65	89	149	307
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	46	98	208	444
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	67	120	242	508

Tableau 5.2: Résultats de la régression linéaire des $\ln \epsilon_u$ en fonction des $\ln n$ *(voir l'équation 5.1)* selon différentes distributions statistiques et le type d'interpolation.

Lois	**Pente** $(-d)$	**Coefficient de corrélation**
Interpolation linéaire		
Loi normale	-2.037531218	-0.999756397
Loi de Cauchy	-2.010292784	-0.999956497
Loi exponentielle	-2.004807683	-0.99996489
Loi Gamma(5)	-2.005950728	-0.999931241
Loi Gamma(1/2)	-2.00079519	-0.999999967
Loi Beta(2,2)	-1.989817772	-0.999920362
Loi Beta(0.3,3)	-2.001033114	-0.99999994
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	-1.995729482	-0.999968892
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	-2.007673844	-0.999884271
Interpolation cubique		
Loi normale	-4.118377281	-0.998936028
Loi de Cauchy	-4.291697024	-0.999610357
Loi exponentielle	-4.164222012	-0.999265359
Loi Gamma(5)	-4.242481231	-0.999973054
Loi Gamma(1/2)	-4.100037646	-0.999905494
Loi Beta(2,2)	-4.238347231	-0.99930296
Loi Beta(0.3,3)	-4.193646275	-0.999939392
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	-4.202123738	-0.999116105
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	-4.214642843	-0.998054712
Interpolation quintique		
Loi normale	-6.469837662	-0.993012162
Loi de Cauchy	-7.455177757	-0.994419566
Loi exponentielle	-7.35793047	-0.993176556
Loi Gamma(5)	-6.780280799	-0.997907384
Loi Gamma(1/2)	-7.245147095	-0.993140034
Loi Beta(2,2)	-7.047943935	-0.995610258
Loi Beta(0.3,3)	-8.63341884	-0.984751943
Loi $\mathcal{P}(1/2)$ composée $\Gamma(5)$	-6.096224756	-0.999999046
Loi $\mathcal{P}(10)$ composée $\Gamma(5)$	-6.773992767	-0.998564982

Dans les tableaux, les lois *Poisson* et *Gamma* sont respectivement abrégées par $\mathcal{P}$ et Γ. Notons que pour les lois de Poisson composée de sévérité Gamma (voir exemple 1.2.3), on évalue la fdr et la densité en tronquant la série. On est capable de borner l'erreur commise par cette troncation avec l'inégalité de Bennett pour la distribution de Poisson.[3]

Remarquons finalement que les résultats obtenus sont très similaires aux résultats présentés dans l'article étudié.[5] Leurs résultats sont placés à l'annexe A.

6 Conclusion

On a montré que l'inversion numérique avec l'interpolation d'Hermite est une méthode simple qui permet de générer des nombres aléatoires d'une distribution continue donnée. De plus, l'interpolation d'Hermite s'avère généralement moins coûteuse par rapport aux méthodes itératives, particulièrement lorsqu'on souhaite générer beaucoup de nombres aléatoires. En utilisant l'interpolation cubique ou quintique, on est en mesure d'obtenir une erreur inférieure à 10^{-12} avec une liste de petite taille. Les ordres de convergence théorique des différentes versions de cette méthode ont d'ailleurs été observées empiriquement à l'aide d'une implémentation de l'algorithme.

Plusieurs pistes sont ouvertes pour poursuivre le projet. On pourrait par exemple étudier le temps d'exécution de l'implémentation effectuée dans ce projet et le comparer avec d'autres implémentations existantes, ou même avec d'autres méthodes permettant de générer des nombres aléatoires. Aussi, il serait certainement intéressant de développer l'analogue multidimensionnel de cette méthode qui utiliserait l'interpolation multidimensionnelle.

Remerciements

L'auteur tient à remercier Philippe-André Luneau et Jean Deteix pour leurs précieux conseils et les discussions fructueuses dans l'élaboration de ce projet.

Références

[1] Carl de Boor. *A Practical Guide to Splines, Revised Edition*. Springer, 2001.

[2] Carl de Boor and Blair Swartz. *Piecewise monotone interpolation*. Journal of approximation theory 21, 411-416, 1977.

[3] John D. Cook. *Poisson distribution tail bounds*. `https://www.johndcook.com/blog/2022/12/14/poisson-bounds/`.

[4] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

[5] Wolfgang Hörmann and Josef Leydold. *Continuous Random Variate Generation by Fast Numerical Inversion*. English. WorkingPaper 45. Department of Statistics, Mathematics, Abt. f. Angewandte Statistik u. Datenverarbeitung, WU Vienna University of Economics, and Business, 2002.

[6] Hung T. Huynh. *Accurate monotone cubic interpolation*. SIAM J. Numer. Anal. 30, 1, 57-100, 1991.

[7] Thierry Duchesne et Khader Khadraoui. *Statistique mathématique*. Département de mathématiques et statistique, Université Laval, 2022.

[8] Etienne Marceau. *Modélisation et évaluation quantitative des risques en actuariat*. Springer, 2013.

[9] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. Society for Industrial and Applied Mathematics, 1992.

[10] R. S. Varga P. G. Ciarlet M. H. Schultz. *Numerical methods of high-order accuracy for nonlinear boundary value problems*. Numer. Math. 9, 394-430, 1967.

[11] Josef Leydold Wolfgang Hörmann and Gerhard Derflinger. *Automatic nonuniform random variate generation*. Springer, 2004.

A Résultats de l'article étudiée

Tableau A.1: Nombres d'intervalles nécessaires pour obtenir l'erreur-u maximale demandée selon différentes distributions statistiques et le type d'interpolation. Résultats de l'article étudié. [5]

Lois	$\overline{\epsilon}_u = 10^{-6}$	$\overline{\epsilon}_u = 10^{-8}$	$\overline{\epsilon}_u = 10^{-10}$	$\overline{\epsilon}_u = 10^{-12}$
Interpolation linéaire				
Loi normale	1063	11533	117875	-
Loi de Cauchy	1849	17491	185335	-
Loi exponentielle	1012	10406	101959	-
Loi Gamma(5)	1072	11225	109336	
Loi Gamma(1/2)	1546	15432	154291	-
Loi Beta(2,2)	823	8009	88179	
Loi Beta(0.3,3)	1884	18783	187786	-
Interpolation cubique				
Loi normale	109	335	941	3091
Loi de Cauchy	179	481	1491	4741
Loi exponentielle	71	207	661	2016
Loi Gamma(5)	105	308	954	3060
Loi Gamma(1/2)	131	277	760	2306
Loi Beta(2,2)	91	251	787	2477
Loi Beta(0.3,3)	167	328	944	2740
Interpolation quintique				
Loi normale	73	127	245	513
Loi de Cauchy	107	175	345	743
Loi exponentielle	49	78	148	316
Loi Gamma(5)	69	119	251	538
Loi Gamma(1/2)	108	137	218	409
Loi Beta(2,2)	65	103	207	451
Loi Beta(0.3,3)	146	169	255	484