Clara Carlier
M2 Mathématiques de l'Aléatoire
Finalité Statistiques et Machine Learning
Université Paris-Saclay, *Orsay*

Calibration de modèle pour le véhicule autonome
Rapport de stage

du 15 Avril 2020 au 15 Août 2020

Tuteur : Matthieu Lerasle
Laboratoire du Crest
Ensae, *Palaiseau*

Avec l'aide de : Loïc Giraldi
et Fabien Mangeant
Technocentre de Renault, *Guyancourt*

Remerciements

Je tiens tout d'abord à remercier Matthieu Lerasle, sans qui ce stage ne se serait littéralement pas déroulé. Bien que cette période soit un peu étrange et assez compliquée pour tout le monde, il a pris le temps de s'intéresser à un sujet qu'il ne connaissait pas afin de m'épauler et me conseiller à distance en apportant toute son expérience en recherche.

Je remercie également Loïc Giraldi et Fabien Mangeant que j'ai rencontrés à quelques reprises avant le confinement et qui malgré l'annulation de mon stage chez Renault, ont tout de même pris le temps de m'encadrer et d'alimenter ce sujet.

Pour finir, je tiens à remercier Christophe Giraud qui a été très présent tout au long de cette année, que ce soit par son oreille attentive ou par ses précieux conseils.

Table des matières

Introduction

Le monde de l'automobile est en pleine expansion et évolution avec l'apparition des voitures autonomes. Qu'elles soient électriques ou non, elles nécessitent une quantité énorme de capteurs en tout genre. De nombreuses entreprises ont à cœur de développer ce type de modèle et des prototypes roulent d'ores et déjà quotidiennement.

En plus de construire un véhicule fiable, un enjeu important réside dans leur validation. En effet, ces nouvelles technologies peuvent avoir un impact majeur sur la vie humaine, il est donc important de les réglementer.

Le Groupe Renault a développé des plateformes numériques qui permettent de modéliser et simuler des systèmes d'aide à la conduite et du véhicule autonome. C'est l'équipe Direction Alliance Ingénierie Produit qui en est responsable.

De par le nombre de capteurs embarqués sur la voiture, ils ont accès à un grand nombre d'informations. Par exemple, la position, la vitesse et l'accélération sont données selon six axes différents et il existe bien d'autres mesures. De plus, la réglementation stricte et leur multiplicité (France, Russie, Inde, Brésil,...) requiert de réaliser beaucoup de tests et donc de parcourir de nombreux kilomètres. S'ajoute à ceci la multiplicité des systèmes ce qui participe à la création de bases de données simulées gigantesques.

De mon côté, j'ai réalisé mon stage au sein de l'équipe de statistiques du Crest du 15 avril 2020 au 15 août 2020. J'ai été encadré par Matthieu Lerasle et le sujet a été proposé et alimenté par Loïc Giraldi et Fabien Mangeant ; deux chercheurs de l'équipe Daip du Groupe Renault. L'objectif global était de calibrer ces modèles puis de valider leurs véhicules à travers ces simulations.

La démarche générale est très similaire à celle utilisée dans l'article [1]. Il a alors été une référence importante tout au long du stage. Le déroulé du sujet s'est articulé autour de ces trois étapes principales :

1. Familiarisation avec le contexte et calibrage des données
2. Construction du modèle de substitution
3. Inférence des paramètres propres à chaque expérience

Chapitre 1

Présentation et traitement des données

Les données reposent sur un scénario spécifique : le véhicule est lancé à une vitesse donnée face à un obstacle et il doit freiner afin de ne pas le percuter. Il y a cinq vitesses initiales différentes : 10, 20, 30, 40 et 50 km/h.

Nous avons d'abord accès à 10 **expériences réelles**, deux par vitesse initiale. Puis, nous avons une base de données plus conséquente répertoriant 500 **simulations numériques** qui sont censées correspondre aux données réelles.

Pour chaque simulation numérique, il nous est renseigné deux paramètres que l'on nomme `EgoSpeed` et `Overlap`. Cependant, pour les données réelles, nous n'en connaissons pas la valeur.

1.1 Les paramètres

1.1.1 Paramètre EgoSpeed

Le paramètre EgoSpeed correspond à la vitesse initiale du véhicule. Dans la figure (1.1), nous représentons son histogramme et nous distinguons bien les cinq groupes correspondant aux cinq vitesses initiales différentes.

1.1.2 Paramètre Overlap

Le paramètre Overlap dépend du décalage de la voiture selon l'axe du milieu. Une représentation est donnée dans la figure (1.2). Nous remarquons là une petite incohérence : les paramètres -100 et 100 correspondent à la même position du véhicule. Nous allons alors procéder à un mapping afin de répartir les valeurs entre -1 et 1. Ainsi, lorsque le véhicule est bien centré, son paramètre vaut 0. Dans la figure (1.3), nous représentons d'abord l'histogramme des données non mappées. Le deuxième histogramme correspond aux données mappées.

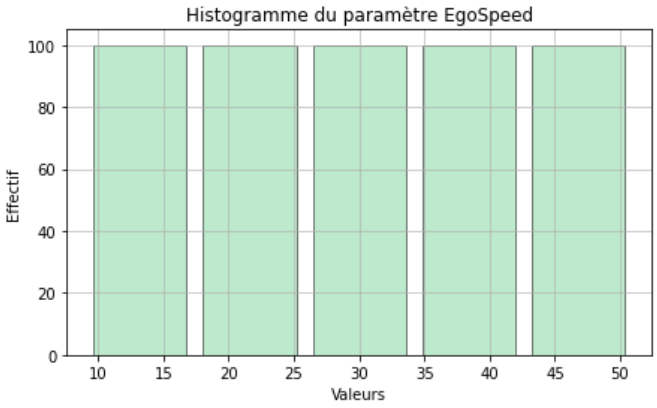


FIGURE 1.1 – Histogramme du paramètre EgoSpeed

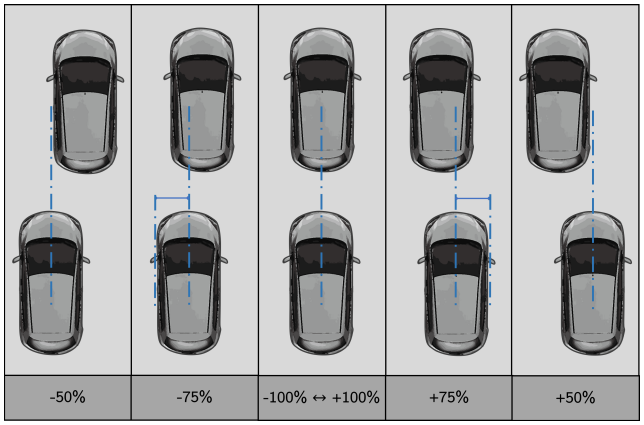
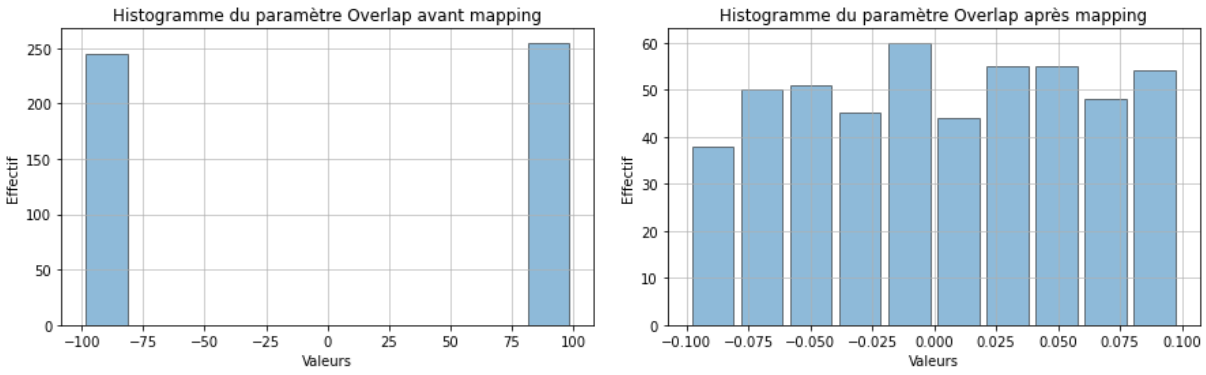


FIGURE 1.2 – Présentation du paramètre latéral Overlap



(a) Avant (b) Après

FIGURE 1.3 – Histogrammes représentant le paramètre Overlap avant et après mapping

1.2 Données réelles

Pour chaque expérience réelle, nous avons accès aux informations suivantes :

- le temps,
- la position selon les axes x, y et z,
- la vitesse selon six axes différents,
- l'accélération et l'angle des roues selon trois axes,
- la distance longitudinale relative (Bumper Distance) et le temps avant collision (TTC).

Dans la figure (1.4), nous représentons graphiquement la vitesse (en m/s), l'accélération, la Bumper Distance et le TTC des données réelles selon le premier axe. Les premières valeurs n'apportent pas d'information importante car le véhicule est simplement lancé à vitesse constante, par conséquent nous allons supprimer le début des courbes.

Nous notons trois problèmes principaux. Tout d'abord, nous remarquons que les courbes ne sont pas bien synchronisées, cela se voit clairement avec la vitesse : le freinage n'est pas initialisé au même moment. Ensuite, les courbes d'accélération sont très bruitées. Enfin, les courbes du TTC admettent des valeurs aberrantes.

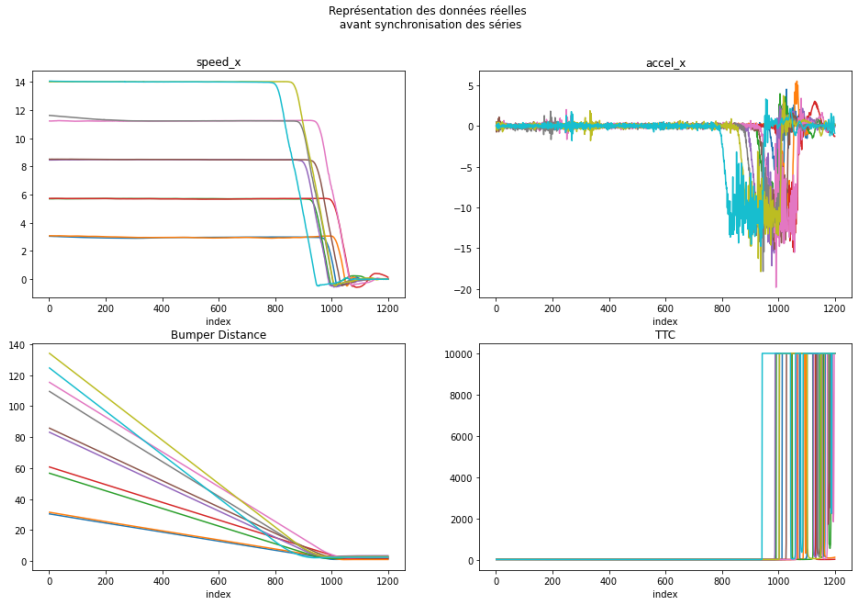


FIGURE 1.4 – Représentation graphique des variables d'intérêt des données réelles avant modification et synchronisation

1.2.1 Synchronisation des données

Nous commençons par synchroniser les données. Pour cela, nous décalons les séries afin que toutes les vitesses valent 1 m/s au même pas de temps. Bien évidemment, nous réalisons le décalage sur toutes les courbes.

Les données synchronisées sont représentées dans la figure (1.5). Désormais, le freinage débute environ au même moment, quelque soit la vitesse initiale. Cela va nous permettre de simplifier la construction de nos modèles par la suite et ainsi obtenir de meilleurs résultats.

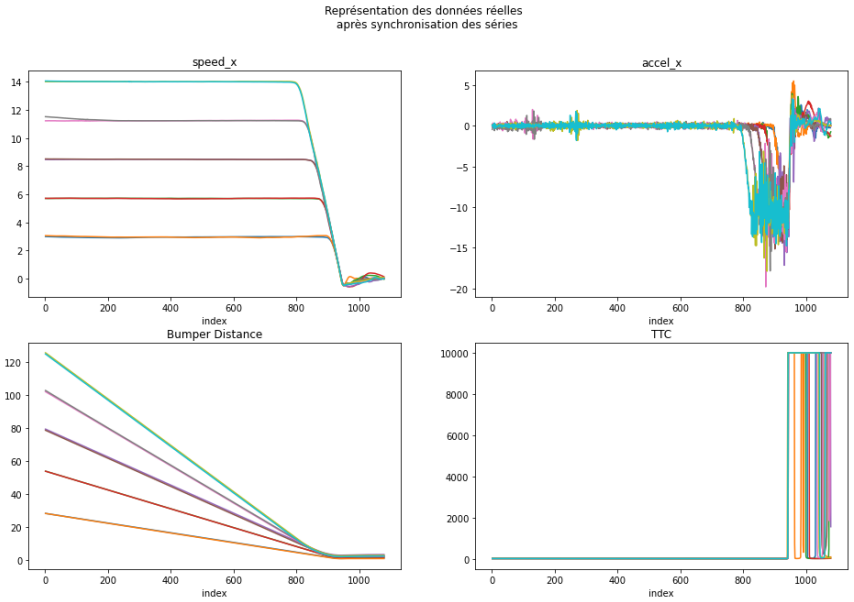


FIGURE 1.5 – Représentation graphique des variables d'intérêt des données réelles après synchronisation

1.2.2 Filtrage de l'accélération

Nous allons utiliser un *filtre passe-bas* [1] de **Butterworth**. Il nous a été proposé par Loïc car il s'agit du filtre utilisé par les organismes de certification [2].

Il s'agit d'un filtre linéaire qui a pour objectif de posséder un gain aussi constant que possible dans la bande passante puis de tendre vers 0 dans la bande de coupure. C'est ce qui va permettre de conserver les basses fréquences et de supprimer les hautes.

1. il s'agit d'un filtre qui laisse passer les basses fréquences et atténue les hautes fréquences (fréquence de coupure haute)
2. Section 4.4.1.2, page 7 : https ://cdn.euroncap.com/media/56143/euro-ncap-aeb-c2c-test-protocol-v302.pdf

Le gain d'un filtre de Butterworth passe-bas d'ordre n est donné par la formule (1.1).

$$G_n(w) = |H_n(iw)| = \frac{1}{\sqrt{1 + \left(w/w_c\right)^{2n}}} \tag{1.1}$$

où H_n correspond à la *fonction de transfert*[3], i l'unité imaginaire, w la fréquence angulaire du signal et w_c la fréquence de coupure du filtre.

Dans la figure (1.6), nous représentons les accélérations des données réelles selon le premier axe x avant et après filtrage. Le filtre permet d'atténuer une bonne partie du bruit. Le fait qu'il reste quelques oscillations ne pose pas de soucis car nous retrouverons le même type de courbes avec les données simulées.

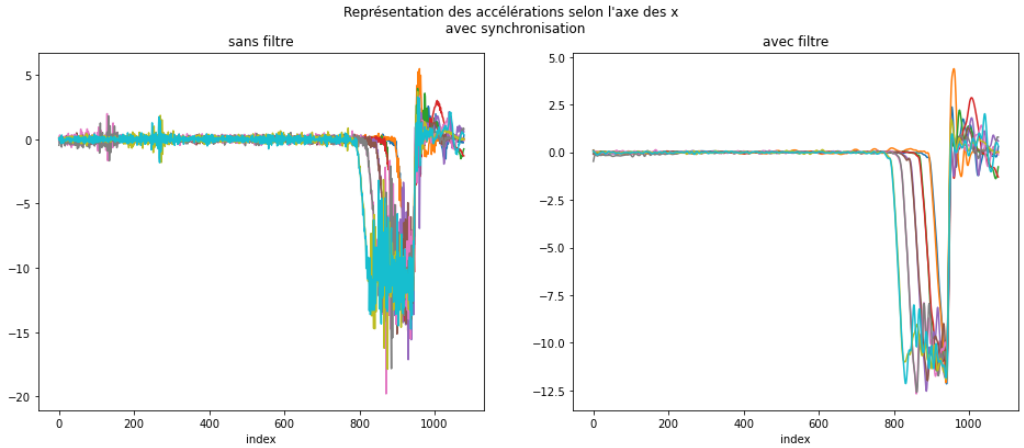


FIGURE 1.6 – Représentation graphique de l'accélération selon l'axe x des données réelles après synchronisation

1.3 Données simulées

Pour chaque simulation, nous avons accès à la position, la vitesse et l'accélération selon six axes différents ainsi qu'au temps, la Bumper Distance et le TTC.

Dans la figure (1.7), nous représentons la vitesse (en m/s), l'accélération, la Bumper Distance et le TTC des données simulées selon le premier axe. Les simulations ne sont pas toutes de la même longueur. Cette fois-ci, le découpage du début des données est encore plus important.

Cette fois, les courbes sont bien synchronisées et les accélérations sont peu bruitées. Par ailleurs, nous observons le même défaut pour les courbes du TTC. De plus, les courbes de la Bumper Distance ne semble pas correspondre au profil de celles des données réelles.

3. c'est un modèle mathématique de la relation entre l'entrée et la sortie d'un système linéaire, le plus souvent invariant

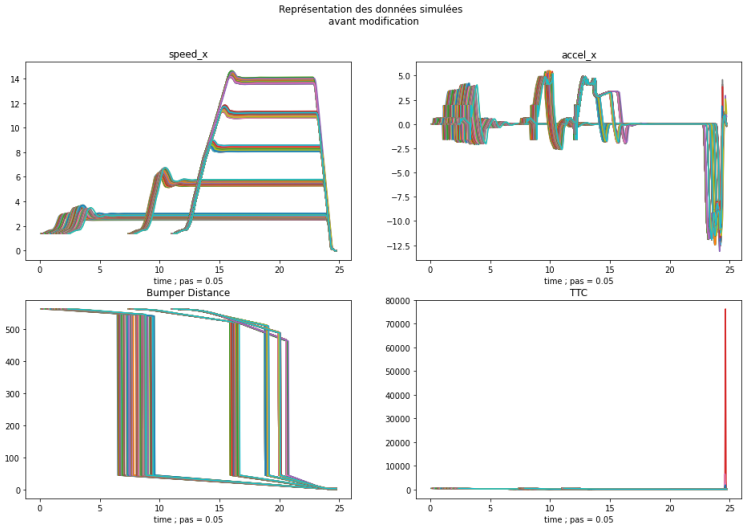


FIGURE 1.7 – Représentation graphique des variables d'intérêt des données simulées avant modification

1.3.1 Découpage des données

Nous représentons les nouvelles courbes dans la figure (1.8). Le découpage des données nous permet de confirmer notre premier avis : les courbes sont bien synchronisées et les accélérations ont des allures suffisamment débruitées. De plus, nous avons désormais des courbes de Bumper Distance satisfaisantes qui correspondent mieux à celles des données réelles.

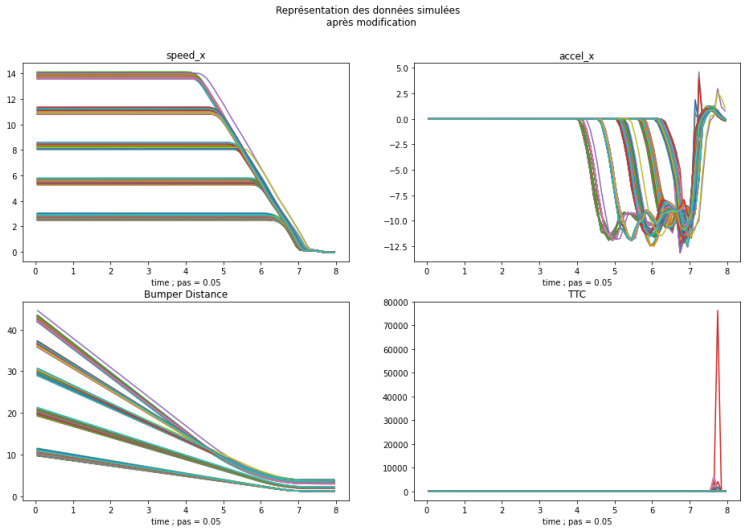


FIGURE 1.8 – Représentation graphique des variables d'intérêt des données simulées après modification

1.4 Seuillage du TTC des données réelles et simulées

Nous procédons au seuillage du TTC pour les données réelles et simulées. En réalité, nous allons uniquement conserver les trente valeurs précédents les grands pics. Le résultat obtenu est représenté en figure (1.9).

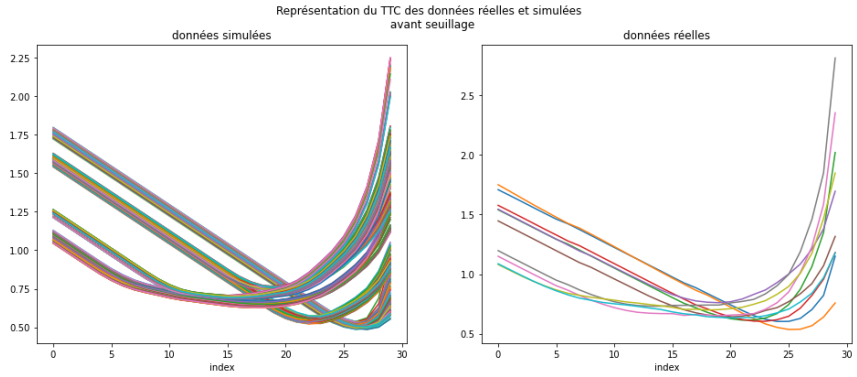


FIGURE 1.9 – Représentation graphique du TTC des données réelles et simulées après seuillage

1.5 Synchronisation des données réelles et simulées

Après toutes ces modifications réalisées sur les données réelles et simulées séparemment, il faut maintenant les synchroniser afin que les allures des courbes soient le plus similaire possible.

Tout d'abord, nous avons besoin de découper à nouveaux les données réelles afin que les allures des courbes soient semblables et que, par exemple, le freinage se réalise à peu près au même pas de temps.

Ensuite, la deuxième chose à changer est la fréquence des mesures. En effet, les données réelles ont une mesure tous les 0.01 pas de temps tandis que celles simulées, tous les 0.05. Nous allons donc supprimer quelques mesures afin d'obtenir des bases de données de même taille.

Les données obtenues après tous ces changements sont représentées graphiquement dans la figure (1.10). Pour cela, nous superposons les données réelles et simulées afin de pouvoir les comparer plus facilement. En orange, ce sont les données réelles et en bleu, les données simulées.

À ce stade, la synchronisation des données ne nous a pas paru pleinement satisfaisante. Tout d'abord, les dernières valeurs de la vitesse doivent être nulles car le véhicule, une fois arrêté, n'est pas censé redémarrer. Nous avons même l'impression que les courbes oranges sont négatives sur la fin. Ensuite, en regardant les courbes de l'accélération, nous trouvons que les données réelles ne correspondent pas correctement à celles simulées. Nous décidons donc de synchroniser à nouveau

nos courbes en se référant au pas de temps où l'accélération vaut 2, soit le début du freinage. Nous allons également découper la fin des données. Le résultat obtenu est représenté en figure (1.11).

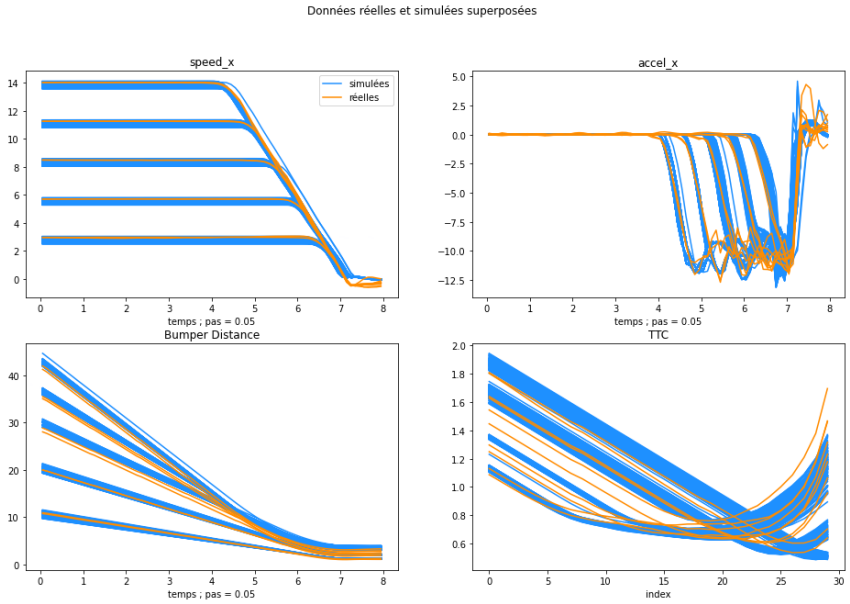


FIGURE 1.10 – Représentation graphique des données réelles et simulées avant synchronisation

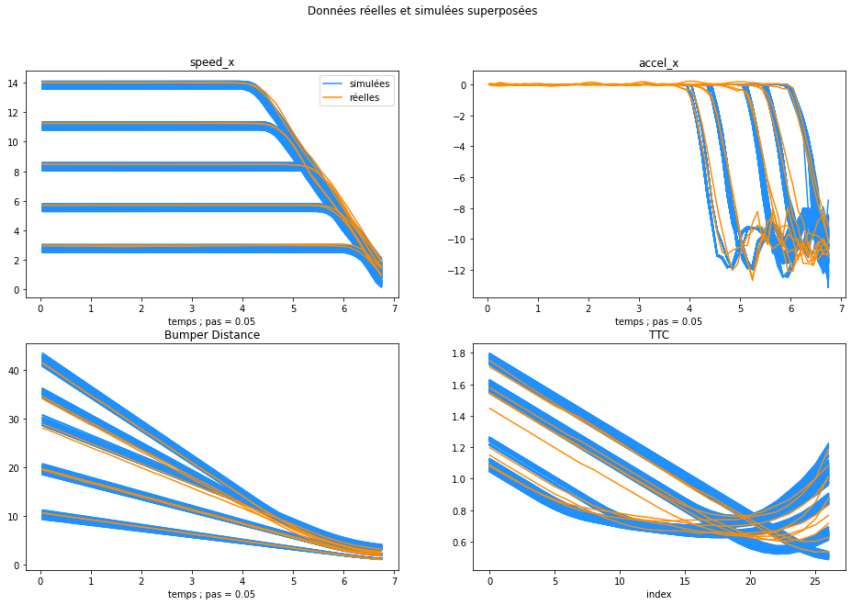


FIGURE 1.11 – Représentation graphique des données réelles et simulées après synchronisation

Chapitre 2

Construction du modèle de substitution

Dans cette section, nous cherchons à construire un modèle de substitution. L'idée est de pouvoir prédire le comportement du véhicule durant l'expérience (position, vitesse, accélération, bumper distance et ttc) à l'aide de deux valeurs : les paramètres EgoSpeed et Overlap. Ce modèle va nous permettre par la suite, lors de l'étape d'inférence, d'avoir une estimation de la vraisemblance (voir plus d'explications dans le chapitre 3).

Notre objectif final est d'utiliser des réseaux de neurones. Afin d'avoir une première idée des résultats que nous pourrions obtenir, nous appliquons d'abord une méthode des forêts aléatoires.

2.1 Quantités d'intérêts

Pour déterminer la qualité du modèle, nous allons nous intéresser à deux quantités. La première correspond à l'erreur quadratique moyenne de la position en x au dernier pas de temps. La deuxième quantité est donnée par la formule (2.1).

$$Q = 0.5 \times \|s_x\|_s + \|a_x\|_a \tag{2.1}$$

où s_x (respectivement a_x) correspond au vecteur de la vitesse (respectivement l'accélération) selon l'axe x à chaque pas de temps.
Les normes s et a sont données par les formules (2.2) et (2.3).

$$\|x\|_s = \|x\|_1 + \|x\|_2 + 0.5 \times \|x\|_{+\infty} \tag{2.2}$$
$$\|y\|_a = \|y\|_1 + \|y\|_2 + \|y\|_{+\infty} \tag{2.3}$$

Par ailleurs, nous calculerons l'erreur quadratique moyenne globale afin de se faire une idée de la qualité des modèles.

2.2 Forêts aléatoires

Nous allons commencer par faire quelques rappels sur la théorie des forêts aléatoires. Ce qui suit a été en partie tiré du cours de Marie-Anne POURSAT portant sur les méthodes de simulation statistiques [2].

2.2.1 Théorie

La méthode des forêts aléatoires reprend l'idée du *bagging*[1] en l'améliorant. L'objectif de ce type de méthode est de construire un estimateur meilleur en moyennant différents prédicteurs, ce qui va permettre de réduire la variance de la prédiction.

Les forêts aléatoires procède à l'aggrégation des prédicteurs et elles fonctionnent particulièrement bien avec des prédicteurs peu biaisés. En l'occurence, nous allons utiliser des *arbres*[2]. Le pseudo-code est donné ci-après.

1. Générer un échantillon *bootstrap*[3]
2. Ajuster un *arbre binaire* à l'échantillon bootstrap
   - à chaque nœud, sélection aléatoire de m variables explicatives parmi p ; le split est basé sur l'une des m variables sélectionnées ;
   - laisser l'algorithme de partition récursive se dérouler jusqu'à l'arbre maximal T_b (sans élagage).
3. Répéter B fois la première et la deuxième étape
4. À partir de la forêt aléatoire $\{T_b,\ b = 1, \ldots, B\}$, pour une régression, construire le prédicteur en x :
$$\widehat{f}_{\mathrm{RF}}(x) = \frac{1}{B} \sum_{b} T_b(x)$$

La forêt aléatoire est une méthode facile à mettre en œuvre et admet de très bonnes performances. Il n'y a que deux paramètres à calibrer : B et m. Cependant, l'interprétation n'est pas facile. En régression, il est recommandé de prendre $m = \sqrt{p}$ où p est le nombre de variables. Lorsque m est égal à p, cela revient à faire du bagging.

1. voir Annexe A.3
2. voir Annexe A.2
3. voir Annexe A.1

2.2.2 Résultats obtenus

Nous avons d'abord testé différents paramètres m et cela ne semblait pas influer fortement sur les résultats, nous avons donc décidé de laisser le choix par défaut.

Nous avons ensuite fait varier le paramètre B. L'évolution des quantités d'intérêt est donnée dans la figure (2.1). En bleu, nous retrouvons les quantités calculées sur les bases d'apprentissage et en orange, les bases de test.

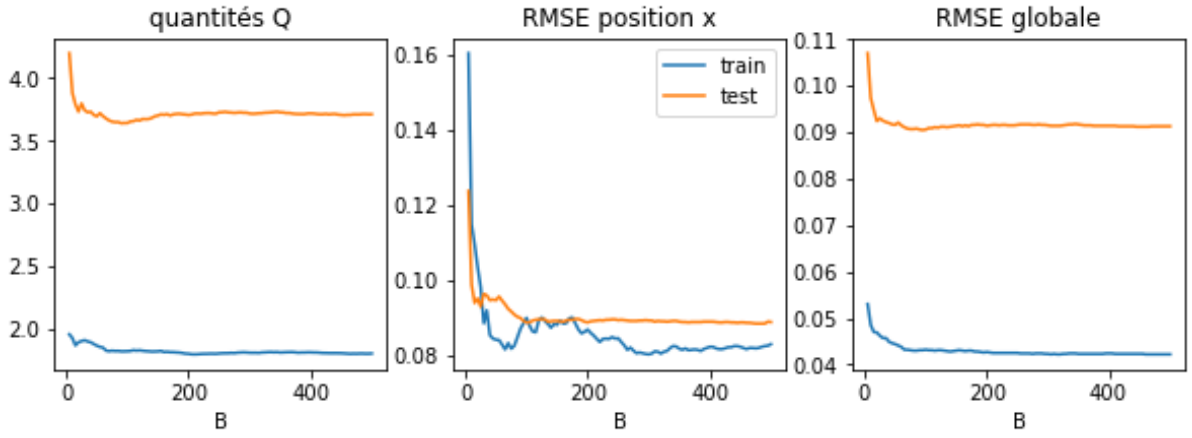


FIGURE 2.1 – Évolution des quantités d'intérêt pour différents paramètres B

Finalement, nous retenons $B = 91$ et nous obtenons les quantités d'intérêt données dans le tableau 2.1. Le paramètre m est choisi par défaut.

	RMSE position x	quantité Q	RMSE globale
sur données *train*	0.0874	1.8151	0.0431
sur données *test*	0.0893	3.6361	0.0903

TABLE 2.1 – Quantités d'intérêts obtenues à l'aide d'une forêt aléatoire

Les résultats obtenus sont étonnamment bons. Nous espérons alors obtenir des résultats au moins similaires avec un réseau de neurones.

Dans la figure (2.2), nous représentons la prédiction de la position, la vitesse, l'accélération, la bumper distance et le ttc d'une expérience faite à l'aide du modèle de forêt aléatoire à partir de paramètres EgoSpeed et Overlap. Elle est en orange et en bleu, il s'agit des vraies valeurs. Les courbes sont effectivement très similaires, ce qui est cohérent avec les bonnes valeurs que nous avons obtenues.

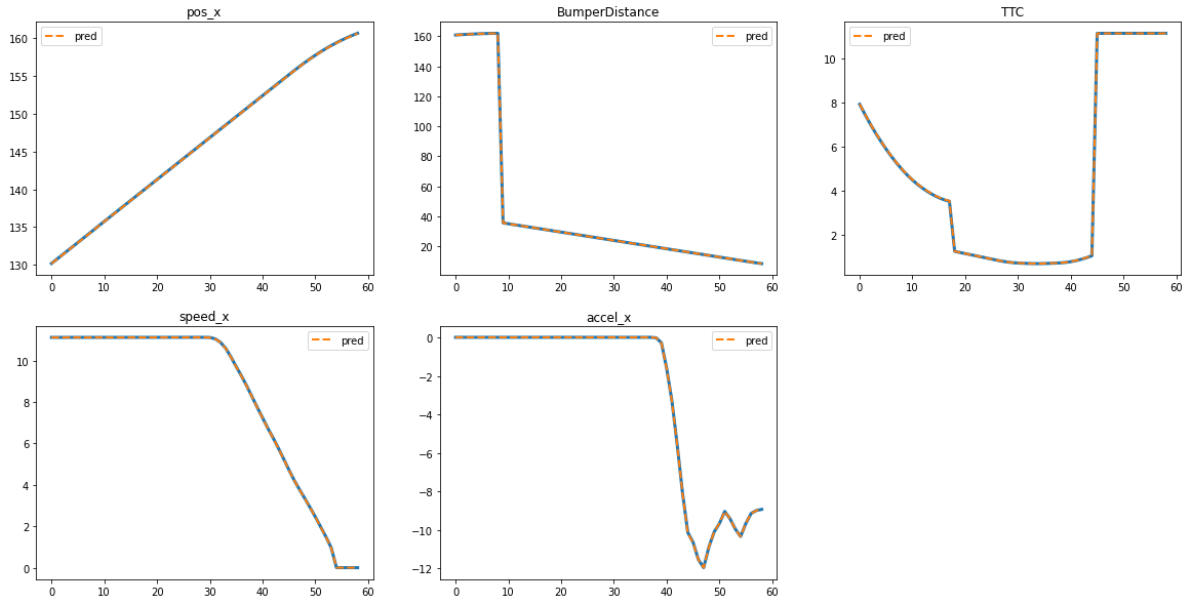


FIGURE 2.2 – Prédiction réalisée avec le modèle de forêt aléatoire pour la 30$^{\text{ième}}$ expérience

2.3 Réseaux de neurones

Commençons par quelques rappels sur les réseaux de neurones. Les informations ont été tirées du chapitre 4 du cours d'apprentissage statistiques de Vianney PERCHET [3], du cours sur les réseaux de neurones de Marc Parizeau [4] et d'un cours de réseaux de neurones de l'Université de Toulouse [5].

2.3.1 Théorie

Un réseau de neurone est l'association d'un graphe dirigé plus ou moins complexe et d'objets élémentaires : les neurones formels. Généralement, ils sont organisés en couches mais leur architecture peut varier. Le niveau de complexité du réseau dépend, entre autre, du nombre de neurones et de la présence ou non de boucles de rétroaction. Les neurones peuvent être de différents types qui sont alors définis par différentes fonction d'activation et de transition (voir ci-après). Les réseaux permettent de réaliser plusieurs tâches : apprentissage supervisé ou non, optimisation, systèmes dynamiques,...

Un neurone formel est défini par un modèle caractérisé par un état interne $s \in S$, des signaux d'entrée $x_1, \dots, x_r$ et une fonction d'activation :

$$s = h(x_1, \dots, x_p) = g\left(w_0 + \sum_{j=1}^{r} w_j x_j\right) = g(w_0 + w'x) \tag{2.4}$$

où nous avons :

- g est la fonction d'activation qui opére une transformation sur une combinaison affine des entrées,
- w_0 correspond au biais du neurone.

Le vecteur $w' = (w_0, \dots, w_r)$ correspond aux poids associé à chaque neurone et dont les valeurs sont estimées pendant la phase d'apprentissage.

Les principaux types de fonction d'activation g sont données ci-après.

type	fonction
linéaire	fonction identité
seuil	$g(x) = \mathbb{1}_{[0,+\infty[}(x)$
sigmoïde	$g(x) = 1/(1+\exp(x))$
ReLU	$g(x) = \max(0, x)$
softmax	$g(x)_j = \exp(x_j)/\sum_{k=1}^{K}\exp(x_k)$
radiale	$g(x) = \sqrt{1/2\pi}\exp(-x^2/2)$
stochastique	$g(x) = 1$ avec proba. $1/(1+\exp(-x/H)), 0$ sinon

TABLE 2.2 – Différentes fonction d'activation possible pour les neurones

Dans la figure (2.3), nous représentons un exemple de réseau de neurones contenant deux couches chacune ayant respectivement p et q neurones.

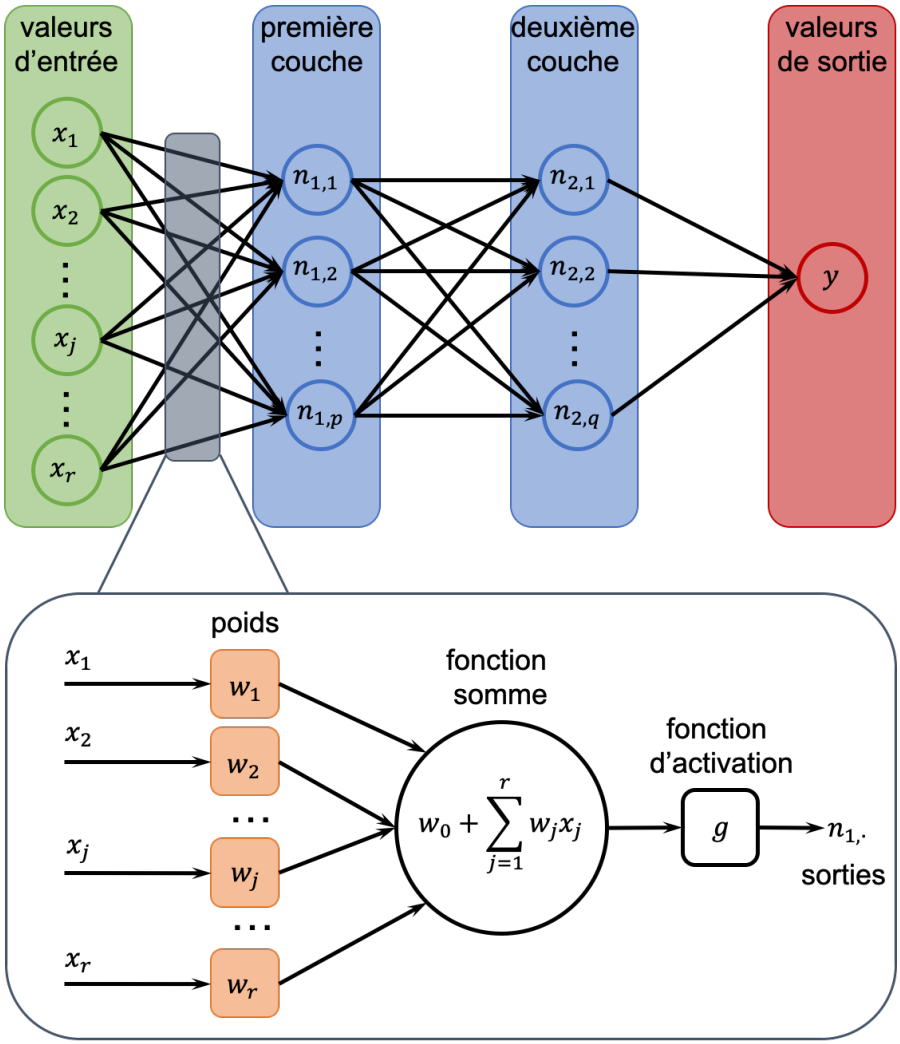


FIGURE 2.3 – Schéma représentant un réseau de neurones contenant deux couches : une de p neurones et l'autre de q neurones

2.3.2 Premier modèle naïf

Nous allons tout d'abord appliqué un réseau de neurones classique. Pour cela nous utilisons la fonction `Dense` de `keras`. Nous y avons couplé un *early stopping*[4].

Nous avons donc fait varier le nombre de neurones pour chaque couche du réseau et le nombre d'*epochs* dans le modèle. Les différentes quantités d'intérêt obtenues sont renseignées dans les

4. méthode de régularisation permettant d'éviter le sur-apprentissage

tableaux 2.3 et 2.4.

nb. neurones	nb. epochs	données	RMSE position x	quantité Q	RMSE globale
10 neurones	100 epochs	*train*	18.7997	93.5663	7.8494
		test	20.0016	93.5604	8.3945
100 neurones	100 epochs	*train*	2.9725	71.8331	1.5653
		test	3.2131	73.989	1.5954
300 neurones	100 epochs	*train*	0.9243	**43.5585**	0.6575
		test	1.0312	**42.8992**	0.6856
500 neurones	100 epochs	*train*	**0.7531**	45.5311	**0.6181**
		test	**0.7880**	45.3243	**0.6253**
600 neurones	100 epochs	*train*	0.7897	47.9229	0.6496
		test	0.8846	46.7708	0.6608

TABLE 2.3 – Quantités d'intérêts obtenues à l'aide d'un premier réseau de neurones contenant 100 epochs et pour différents nombre de neurones

nb. neurones	nb. epochs	données	RMSE position x	quantité Q	RMSE globale
300 neurones	100 epochs	*train*	0.9243	43.5585	0.6575
		test	1.0312	42.8992	0.6856
300 neurones	200 epochs	*train*	0.7848	35.2717	0.5532
		test	0.8092	37.6581	0.5915
300 neurones	300 epochs	*train*	1.2274	**31.5462**	0.6803
		test	1.2394	**34.3152**	0.7056
300 neurones	400 epochs	*train*	**0.7068**	35.0264	**0.5097**
		test	**0.7062**	37.1387	**0.5394**

TABLE 2.4 – Quantités d'intérêts obtenues à l'aide d'un premier réseau de neurones contenant 300 neurones et pour différents nombre d'epochs

Ces résultats sont plus mauvais que ceux obtenus avec la forêt aléatoire. Même en rajoutant des neurones et des epochs, ils ne sont pas aussi bons que précédemment. Nous espérons qu'en complexifiant le modèle, nous obtiendrons des améliorations.

À nouveau, dans la figure (2.4), nous représentons les prédictions faites pour la $30^{\text{ième}}$ expérience à l'aide de ce premier réseau de neurones. Nous retrouvons les irrégularités qui font que les quantités d'intérêt ne sont pas satisfaisantes.

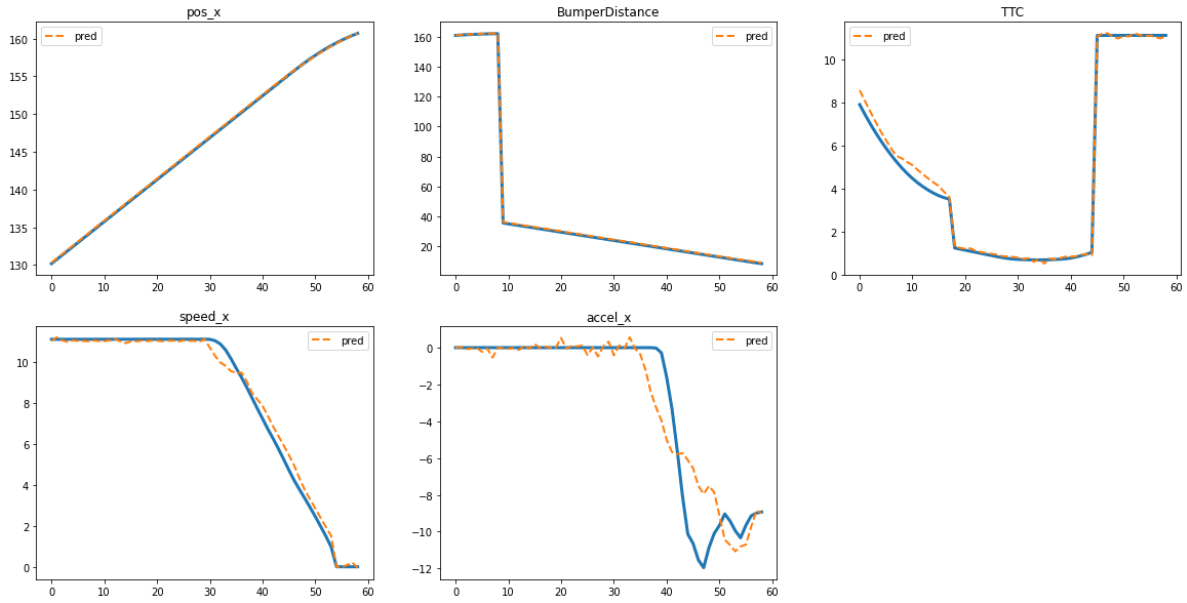


FIGURE 2.4 – Prédiction réalisée avec le premier modèle de réseau de neurones pour la 30$^{\text{ième}}$ expérience

2.3.3 Modèle convolutionnel

Nous avons ensuite testé un modèle de réseau de neurone convolutionnel un peu plus complexe. Nous y avons également couplé un early stopping.

Cette fois-ci, nous avons fait varier uniquement le nombre d'epochs. Les résultats obtenus sont donnés dans le tableau 2.5.

nb. epochs	données	RMSE position x	quantité Q	RMSE globale
100 epochs	*train*	0.9068	35.2669	0.6293
	test	0.9229	35.9702	0.6388
300 epochs	*train*	0.7374	28.9097	0.4521
	test	0.8867	29.7377	0.5036
500 epochs	*train*	**0.6958**	**14.5039**	**0.3796**
	test	**0.7196**	**14.7249**	**0.3981**
700 epochs	*train*	0.8729	24.5555	0.4776
	test	0.8928	24.5201	0.4928

TABLE 2.5 – Quantités d'intérêts obtenues à l'aide d'un deuxième réseau de neurones pour différents nombre d'epochs

Les quantités d'intérêt sont meilleures mais ce modèle n'est toujours pas aussi bon que celui de la forêt aléatoire.

Dans la figure (2.5), nous représentons les prédictions faites pour la 30$^{\text{ième}}$ expérience. Bien qu'elles semblent meilleures qu'avec le premier réseau de neurones, elles ne sont toujours pas au niveau de celles faites avec la forêt aléatoire.

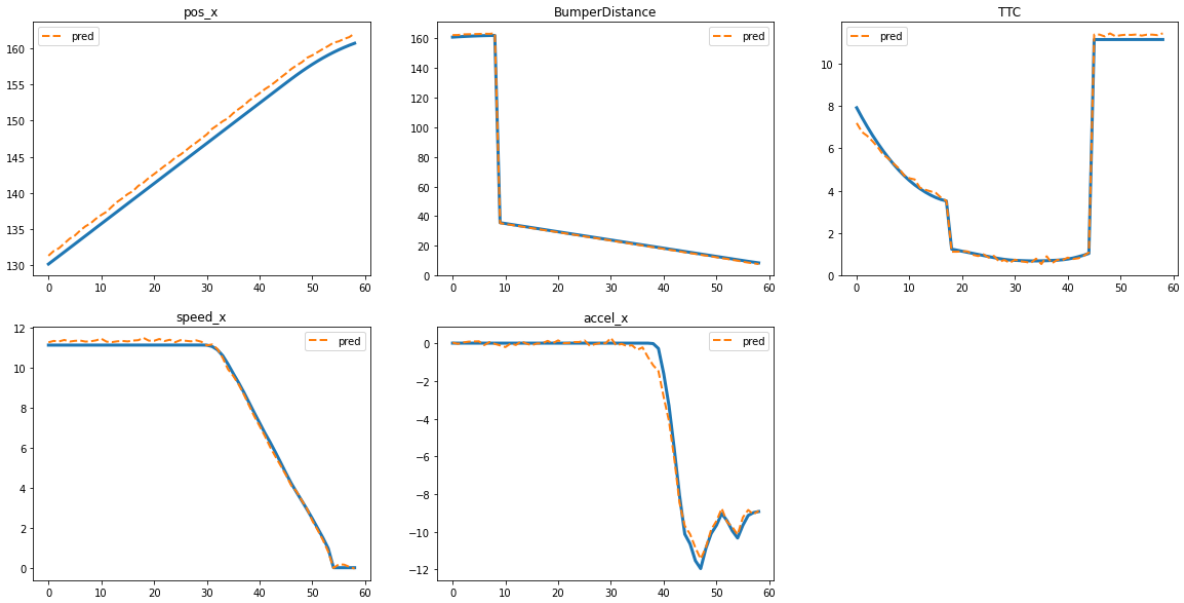


FIGURE 2.5 – Prédiction réalisée avec le deuxième modèle de réseau de neurones pour la 30$^{\text{ième}}$ expérience

Au vu de tous ces résultats, nous pensons qu'il est tout à fait raisonnable de conserver le modèle de forêt aléatoire comme modèle de substitution et de ne pas s'attarder plus sur la construction d'un réseau de neurones plus performant. C'est donc la forêt aléatoire qui sera utilisée dans la suite.

Chapitre 3

Inférence des paramètres

Nous allons maintenant construire une représentation de la position, la vitesse, l'accélération, la Bumper Distance et le TTC des données réelles pour l'exploiter afin d'en déduire les paramètres EgoSpeed et Overlap liés à chaque expérience.

Notons $\theta = (\texttt{EgoSpeed}, \texttt{Overlap})^{\mathrm{T}}$ un vecteur aléatoire de dimension 2 et Y les données expérimentales auxquelles nous avons accès. L'objectif est d'estimer la distribution postérieure $P(\theta|Y)$. Pour une réalisation y des données, le célèbre théorème de Bayes nous donne :

$$P(\theta|Y = y) = \frac{P(Y = y|\theta)P(\theta)}{P(Y = y)} \tag{3.1}$$

où $P(Y = y|\theta)$ correspond à la vraisemblance et $P(\theta)$ est le prior. Pour le prior, nous souhaitons prendre une loi uniforme. La vraisemblance nous est donnée par le simulateur S qui est le modèle donné par la forêt aléatoire.

Dans [1], ils utilisent un modèle de bruit additif $y = S(\theta) + \varepsilon$ où ε est un bruit gaussien puis ils échantillonnent la distribution postérieure par MCMC. Dans notre contexte, nous avons supposé que le signal est exactement reproductible par le simulateur $S(\theta)$. Pour échantillonner la distribution postérieure, notre choix s'est porté sur l'utilisation des méthodes ABC (*Approximate Bayesian Computation*) dont un état de l'art est donné dans [6]. Ce sont des méthodes qui permettent d'inférer la distribution postérieure bien que la vraisemblance soit difficile ou coûteuse à évaluer. Ce type de méthode suit le schéma suivant :

1. Échantillonner un paramètre θ^* selon le prior $P(\theta)$
2. Simuler une base de données y^* à l'aide d'une fonction qui associe à θ des données de la même dimension que les données observées y_0 (étape `simulator`)
3. Comparer les données simulées y^* avec les données expérimentales y_0 en utilisant une distance d et un seuil de tolérance ε

Le résultat obtenu est un échantillonnage des paramètres selon une distribution qui vérifie :

$$P\big(\theta|d(y_0, y^*) \leq \varepsilon\big) \tag{3.2}$$

Pour ε suffisamment petit, cette distribution est une bonne approximation de la distribution postérieure $P(\theta|Y = y_0)$.

Avant de nous lancer dans l'utilisation d'une méthode ABC, nous allons commencer par appliquer une méthode classique de MCMC : l'algorithme de Metropolis-Hastings. Cela nous permettra de nous donner une idée du type de résultats que nous pourrions obtenir.

Dans la première partie de ce chapitre, nous expliquons comment nous allons évaluer la qualité de l'inférence.

3.1 Évaluation de l'inférence

L'objectif est d'inférer les paramètres EgoSpeed et Overlap pour chaque expérience réelle. Cependant, nous ne connaissons pas les valeurs réelles qu'ils sont censés avoir. Pour avoir une valeur à comparer, nous allons procéder comme suit :

1. Sélectionner l'expérience simulée la plus proche de celle réelle
2. Comparer les paramètres inférés à ceux de l'expérience simulée la plus proche

Pour sélectionner l'expérience la plus proche, nous allons calculer l'erreur quadratique moyenne entre les vecteurs de l'accélération selon x et garder la plus petite. Autrement dit, pour tout $l = 1, \ldots, 10$, nous allons minimiser :

$$\min_{k=1,\ldots,500} \left(\frac{1}{n} \sum_{i=1}^{n} \left((a_{x,k}^{simu})_i - (a_{x,l}^{real})_i \right) \right)^{1/2} \tag{3.3}$$

où $a_{x,k}^{simu}$ est un vecteur de taille n correspondant à l'évolution à travers le temps de l'accélération selon l'axe x de la $k^{\text{ième}}$ expérience simulée, de même pour $a_{x,l}^{real}$ qui correspond à la $l^{\text{ième}}$ expérience réelle cette fois.

Nous obtenons finalement un vecteur de taille 10 identifiant l'expérience simulée correspondant à chaque expérience réelle :

$$\left(k_1^*, k_2^*, \ldots, k_{10}^* \right) \tag{3.4}$$

Ainsi nous pouvons comparer les paramètres inférés aux paramètres sélectionnés par ce vecteur.

Avant de commencer à appliquer des méthodes d'inférence, nous calculons l'erreur quadratique moyenne entre les paramètres de référence et des valeurs "naïves" :

- le vecteur $(10, 10, 20, 20, 30, 30, 40, 40, 50, 50)$ pour EgoSpeed,
- le vecteur $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ pour Overlap.

Les erreurs quadratiques moyennes valent respectivement **0.6116** et **0.0661**. Idéalement, nous aimerions améliorer ces valeurs par la suite.

3.2 MCMC : algorithme de Metropolis-Hastings

Pour ce faire, nous allons utiliser le package `PyMC3` [1]. Il s'agit d'un package de programmation probabiliste. Cela va nous permettre d'implémenter une sorte d'étape de Metropolis-Hastings. En effet, dans celui-ci, l'écart-type de la loi de proposition est optimisé pour être dans un certain taux d'acceptation.

3.2.1 Première approche : cas par cas

Afin de nous approprier le package et comprendre son fonctionnement, nous avons commencer par tester cette méthode sur une seule expérience réelle.

La construction du modèle se fait comme suit.

```
with pm.Model() as model:
    a = pm.Normal('a', mu, sd) # EgoSpeed
    b = pm.Normal('b', mu, sd) # Overlap
    simulator = pm.Simulator('simulator', sim_rf, observed=obs)
    step1 = pm.Metropolis([a, b])
    trace = pm.sample(step=[step1])
```

La variable `a` (respectivement `b`) correspond au paramètre EgoSpeed (respectivement Overlap). Les deux premières lignes correspondent au choix des priors. Nous n'avons pas pu prendre des lois uniformes comme initialement souhaité car elles ne fonctionnaient pas. Nous avons donc opté pour des lois normales et avons fait varier leurs paramètres.

La ligne `simulator` correspond à l'étape 2 décrite au début de ce chapitre. La fonction `sim_rf` permet de prédire la position, la vitesse, l'accélération, la bumper distance et le ttc pour chaque pas de temps à partir de deux valeurs (EgoSpeed et Overlap) à l'aide du modèle de forêt aléatoire construit dans le chapitre 2.

Nous allons alors inférer les paramètres des 10 expérience réelle que nous avons à disposition, puis calculer l'erreur quadratique moyenne avec les paramètres sélectionnés.

Dans le tableau 3.1, nous renseignons les différents écarts-types testés pour les priors, l'erreur quadratique moyenne et l'écart-type du posterior associés, ainsi que les temps d'exécution.

1. `https://docs.pymc.io`

	EgoSpeed			Overlap			tps. d'exéc.
	σ prior	RMSE	σ post.	σ prior	RMSE	σ post.	
Modèle 1	20	14.3674	19.7286	1	0.0897	0.9946	38 sec
Modèle 2	5	14.3237	5.0289	0.2	0.0700	0.2002	12 sec
Modèle 3	0.5	14.2503	0.4924	0.03	**0.0646**	0.0310	12 sec
Modèle 4	0.05	**14.2427**	0.0501	0.01	0.0657	0.0098	12 sec

TABLE 3.1 – Résultats obtenus en inférant expérience par expérience avec l'étape de Metropolis-Hastings

Pour le paramètre Overlap, les résultats obtenus sont plutôt positifs. Nous arrivons à améliorer légérement la valeur "naïve". Pour ce qui est du paramètre EgoSpeed, les résultats sont mauvais. De plus, les variances postérieures ne sont pas nécessairement plus petites.

3.2.2 Deuxième approche : modèle global

L'approche dite globale consiste à inférer tous les paramètres "en même temps". Ainsi, dans la construction du modèle, nous introduisons 20 paramètres (2 pour chaque expérience réelle). Cela nous permet de spécifier les priors des paramètres de type EgoSpeed :

- la moyenne sera choisie dans l'ensemble $\{10, 20, 30, 40, 50\}$ en fonction de la vitesse initiale de chaque expérience,
- nous ferons varier la variance.

Les résultats que nous avons obtenu sur les 10 expériences réelles sont donnés dans le tableau 3.2.

	EgoSpeed			Overlap			tps. d'exéc.
	σ prior	RMSE	σ post.	σ prior	RMSE	σ post.	
Modèle 1	5	0.6211	4.9446	1	0.0653	0.9941	9 sec
Modèle 2	1	0.6125	0.9968	0.5	**0.0634**	0.5003	10 sec
Modèle 3	0.5	0.6139	0.4992	0.5	0.0650	0.4922	12 sec
Modèle 4	0.05	0.6117	0.0502	0.05	0.0663	0.0505	12 sec
Modèle 5	0.01	**0.6116**	0.0099	0.01	0.0660	0.0099	13 sec

TABLE 3.2 – Résultats obtenus en inférant de manière globale avec l'étape de Metropolis-Hastings

Cette fois-ci, les résultats sont meilleurs. À deux exceptions près, les variances sont un petit peu améliorées. Pour le paramètre Overlap, les valeurs sont similaires à ce qu'on a eu précédemment mais légérement meilleures. Pour EgoSpeed, nous avons une nette amélioration. Considérer tous les paramètres en même temps semble permettre d'affiner l'inférence d'EgoSpeed. Nous notons que les temps d'exécution sont très courts.

3.3 Méthode ABC : Monte Carlo séquentiel

Nous avons choisi d'utiliser la méthode de Monte Carlo séquentiel [2] du package Python `PyMC3`. Les méthodes ABC implémentées dans ce package sont encore expérimentales. Il est alors probable que nous faisions face à quelques bugs et incohérences.

La méthode Monte Carlo séquentiel ABC transforme itérativement le prior en posterior en propageant les paramètres échantillonnés à travers une série de distributions $\phi\big(\theta^{(i)}\big)$, en pondérant les paramètres acceptés $\theta^{(i)}$ tel que :

$$w^{(i)} \propto \frac{\pi\big(\theta^{(i)}\big)}{\phi\big(\theta^{(i)}\big)} \tag{3.5}$$

Elle permet de combiner les avantages d'un SMC traditionnel, i.e. la possibilité d'échantillonner selon des distributions à pics multiples, mais sans qu'il soit nécessaire d'évaluer la vraisemblance.

3.3.1 Première approche : cas par cas

Nous procédons comme précédemment avec Metropolis-Hastings. Cette fois-ci, le modèle se construit comme suit.

```
with pm.Model() as model:
    a = pm.Normal('a', mu, sd)  # EgoSpeed
    b = pm.Normal('b', mu, sd)    # Overlap
    simulator = pm.Simulator('simulator', sim_rf, observed=obs)
    trace = pm.sample_smc(kernel="ABC")
```

À nouveau, nous donnons les écarts-types des prior et des posterior, l'erreur quadratique moyenne et les temps d'exécution dans le tableau 3.3 pour nos 10 expériences réelles.

	EgoSpeed			Overlap			tps. d'exéc.
	σ prior	RMSE	σ post.	σ prior	RMSE	σ post.	
Modèle 1	20	33.7286	9.1031	1	0.1466	0.9996	49 min
Modèle 2	5	23.1599	2.1529	0.2	0.0799	0.1946	1h et 33min
Modèle 3	0.5	**14.1176**	0.3446	0.03	0.0661	0.0302	37 min
Modèle 4	0.05	14.2417	0.0499	0.01	**0.0660**	0.0100	29 min

TABLE 3.3 – Résultats obtenus en inférant expérience par expérience avec le Monte Carlo séquentiel

2. `https://docs.pymc.io/notebooks/SMC-ABC_Lotka-Volterra_example.html`

Les temps d'exécution sont plutôt longs et les résultats sont globalement peu satisfaisants. Nous remarquons tout de même qu'en réduisant la variance des priors, les résultats s'améliorent. Les erreurs quadratiques moyennes finissent par stagner. Dans le tableau 3.3, nous indiquons les meilleurs résultats en gras. L'inférence d'EgoSpeed n'est pas bonne. Comme pour Metropolis-Hastings, nous pensons qu'en passant au modèle global, l'inférence sera meilleure.

3.3.2 Deuxième approche : modèle global

Nous passons maintenant au modèle global pour le Monte Carlo séquentiel. Les résultats obtenus pour les 10 expériences réelles sont donnés dans le tableau 3.4.

	EgoSpeed			Overlap			tps. d'exéc.
	σ prior	RMSE	σ post.	σ prior	RMSE	σ post.	
Modèle 1	5	5.4103	3.9257	1	0.1851	0.9383	9min et 43sec
Modèle 2	1	0.7729	0.9523	0.5	0.0807	0.4979	4min et 17sec
Modèle 3	0.5	0.6540	0.4870	0.5	0.0709	0.4965	4min et 11sec
Modèle 4	0.05	**0.6116**	0.0499	0.05	**0.0654**	0.0499	3min et 20sec
Modèle 5	0.01	0.6117	0.0099	0.01	0.0661	0.0099	3min et 22sec

TABLE 3.4 – Résultats obtenus en inférant de manière globale avec le Monte Carlo séquentiel (1/2)

Tout d'abord, nous remarquons que les temps d'exécution sont bien plus bas que lors du cas par cas. De plus, les résultats obtenus pour EgoSpeed sont bien meilleurs. Cependant, ceux pour Overlap sont quasiment inchangés. Les variances des posterior sont plus basses.

Nous sommes partis du principe que plus la variance était basse, meilleur était le résultat. D'après ce tableau, nous remarquons que ce n'est pas forcément vrai.

Par ailleurs, nous remarquons que l'approche ABC n'améliore pas nettement les résultats par rapport à l'utilisation d'une étape de Metropolis-Hastings. En plus, les temps d'exécution sont plus longs. Nous nous demandons alors si cela en vaut la peine.

Conclusion

Nous avons commencé et passé pas mal de temps sur le calibrage des données réelles et simulées. C'est une étape importante et primordiale pour la réussite de la construction des modèles. D'ailleurs, il serait intéressant de comparer les résultats que nous obtenons avec différentes synchronisations des séries. De plus, nous avions accès à 500 expériences simulées selon un même scénario et nous pensons qu'en augmentant ce nombre de simulations, les résultats pourraient s'améliorer.

Pour la construction du modèle de substitution, nous avons testé des forêts aléatoires et des réseaux de neurones. Il existe pléthore d'articles les comparant. Bien évidemment, le choix de l'un ou l'autre va dépendre de chaque projet. Nous avons été très satisfaits des résultats obtenus à l'aide des forêts aléatoires. De plus, il s'agit d'une méthode facile et rapide à mettre en œuvre. Il pourrait être intéressant d'essayer d'améliorer la précision du modèle de substitution à l'aide des réseaux de neurones. Il faudra alors y passer plus de temps et le calibrage du modèle va s'avérer plus délicat.

Pour l'inférence des paramètres des données réelles, nous sommes globalement satisfaits des résultats obtenus. Nous avions pour objectif d'utiliser les méthodes ABC afin d'obtenir de meilleurs résultats qu'avec des méthodes plus classiques. Cependant, le Monte Carlo séquentiel n'a pas grandement amélioré les performances par rapport à Metropolis-Hastings. Bien que ce soient des méthodes faciles à utiliser, les temps d'exécution sont plus longs. Par ailleurs, nous nous demandons si les résultats de l'inférence ne sont pas bridés par la qualité du modèle de substitution.

Nous avons supposé que le simulateur $y = S(\theta)$ reproduit exactement le signal. Cette assertion est fausse mais nous a permis de simplifier le modèle dans un premier temps. Il s'agit là d'une piste d'amélioration : le bruit permet de modéliser l'erreur du simulateur. Cela pourrait aider à la réduction de l'écart-type des posterior.

Bibliographie

[1] L. Giraldi, O.P. Le Maître, K.T. Mandli, C.N. Dawson, I. Hoteit, and O.M. Knio. Bayesian inference of earthquake parameters from buoy data using a polynomial chaos based surrogate. *Computational Geosciences*, 21(4) :683–699, 2017.

[2] M.-A. Poursat. Cours sur les méthodes de simulations statistiques, 2019.

[3] V. Perchet. Cours d'apprentissage statistiques, chapitre 4 : Optimisation & intro au neural nets, 2019-2020.

[4] M. Parizeau. Cours sur les réseaux de neurones, 2004.

[5] Cours sur les réseaux de neurones. `https://www.math.univ-toulouse.fr/ besse/Wikistat/pdf/st-m-app-rn.pdf`.

[6] G. Karabatsos and F. Leisen. An approximate likelihood perspective on abc methods. *Statistics Surveys*, 12 :66–104, 2018.

Annexe A

Méthodes d'apprentissage : *boostrap*, *bagging* et *arbre binaire*

Ce qui suit a également été en partie tiré du cours de Marie-Anne POURSAT portant sur les méthodes de simulation statistiques [2].

A.1 Qu'est-ce que le *bootstrap* ?

Le *boostrap* est une méthode statistique de ré-échantillonage qui permet d'évaluer la précision d'une méthode, par exemple estimer une variance, une erreur de prédiction, calculer un intervalle de confiance, etc,... Généralement, il faut observer plusieurs réalisations mais une seule expérience est fournie. L'idée est alors de générer des échantillons qui ressemblent à l'échantillon de départ en ré-utilisant les valeurs initialement fournies.

Considérons $\theta = T(P)$ un paramètre que l'on souhaite estimer. On estime $T(P)$ par $\widehat{\theta} = T(P_n)$. La donnée de l'échantillon $(Y_1, \ldots, Y_n)$ est équivalente à celle de P_n. L'échantillon boostrap de taille n est tiré selon P_n parmi $Y_1, \ldots, Y_n$. Il est noté $Y^* = (Y_1^*, \ldots, Y_n^*)$. En boostrap, au lieu d'estimer $\theta = T(P)$ par $\widehat{\theta} = T(P_n) = T(Y_1, \ldots, Y_n)$, on utilise l'estimateur $T(Y_1^*, \ldots, Y_n^*)$.

Le boostrap n'est valide que si on peut montrer que la loi de $T(Y^*) = \widehat{\theta}^*$ est une bonne approximation de la loi de $T(Y) = \widehat{\theta}$, c'est-à-dire si $\mathcal{L}_P(\widehat{\theta})$ et $\mathcal{L}_{P_n}(\widehat{\theta}^*|Y_1, \ldots, Y_n)$ admette la même loi limite.

A.2 Qu'est-ce qu'un *arbre binaire* ?

Un *arbre binaire* est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Ils permettent de répartir un ensemble de données en différents groupes homogènes selon des variables discriminantes et en fonction d'une variable d'intérêt

précise.

Prenons l'exemple décrit dans le figure (A.1). Il s'agit de la répartition des personnes décédées ou non (variable d'intérêt) à bord du Titanic en 1912 selon leur sexe, leur âge et le nombre de proches qui étaient à bord (variables discriminantes). L'arbre commence par diviser le groupe afin de séparer les hommes des femmes. Puis, du côté des femmes, il fait une distinction en fonction du nombre de proches qui étaient à bord :

- dans le cas où le nombre de proches est supérieur ou égal à 3, il y a 15 femmes qui sont mortes et 5 qui ont survécu,
- à l'inverse, lorsque le nombre de proche est inférieur à 3, il y a 86 femmes qui ont survécu contre 252 qui sont mortes.

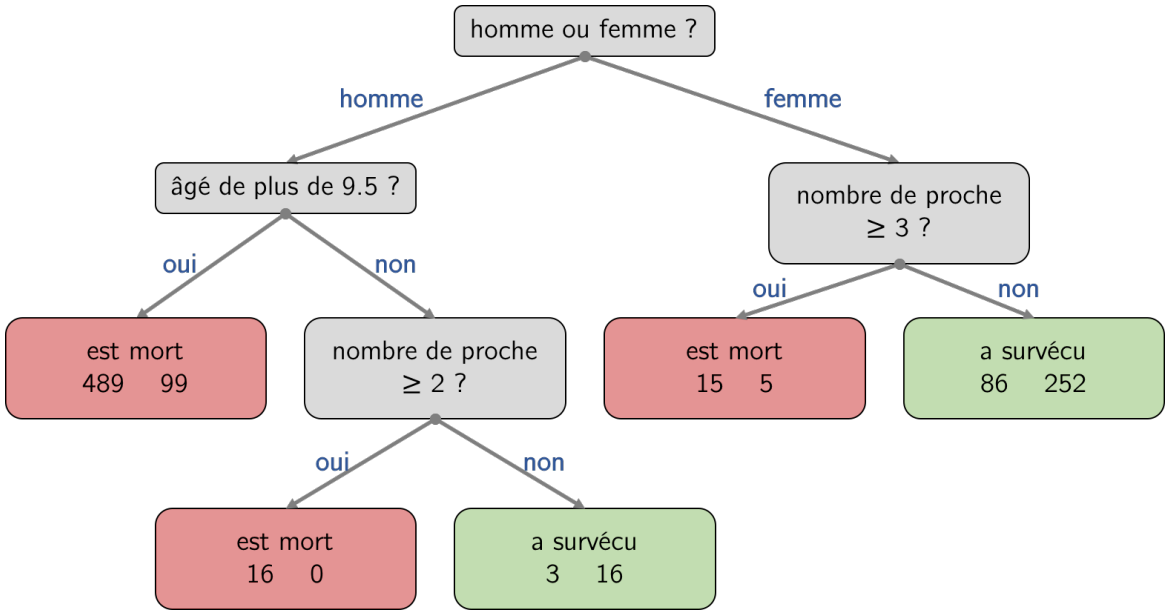


FIGURE A.1 – Exemple d'un arbre de décision binaire

A.3 Qu'est-ce que le *bagging* ?

Le mot *bagging* vient du mélange de boostrap et aggregating. L'objectif de cette méthode est d'améliorer la prédiction d'un estimateur. L'idée est de moyenner des prédicteurs afin de réduire la variance de la prédiction.

Lorsque l'on a accès à un unique échantillon d'apprentissage, nous allons construire B arbres à l'aide du boostrap qui va nous permettre d'accéder à B échantillons. En bagging, il faut utiliser des arbres de taille maximale car ils admettent un biais faible et une forte variance, en les moyennant cette dernière va diminuer sans augmenter le biais pour autant.