Introduction à la statistique bayésienne

avec le logiciel R

Olivier Gimenez

éditions Quæ

Introduction à la statistique bayésienne

avec le logiciel R

Olivier Gimenez

éditions Quæ

Pour citer cet ouvrage :
Gimenez O., 2026. *Introduction à la statistique bayésienne avec le logiciel R*.
Versailles, éditions Quæ, 78 p.
https://doi.org/10.35690/978-2-7592-4258-0

Photo de couverture : © Yann Raulet.

Les éditions Quae réalisent une évaluation scientifique des manuscrits avant publication
dont la procédure est décrite ici :
https://www.quae.com/store/page/199/processus-d-evaluation

Le processus éditorial s'appuie également sur un logiciel de détection des similitudes
et des textes potentiellement générés par intelligence artificielle.

Cet ouvrage a bénéficié d'un financement du CNRS et de l'université de Montpellier.

Éditions Quæ
RD 10
78026 Versailles Cedex, France
www.quae.com
www.quae-open.com

ISBN papier : 978-2-7592-4257-3
ISBN PDF : 978-2-7592-4258-0
ISBN ePub : 978-2-7592-4259-7

SOMMAIRE

AVANT-PROPOS

On retrouve la statistique bayésienne un peu partout en sciences. Par exemple, en épidémiologie, pour prédire la circulation des virus, en écologie, pour expliquer l'extinction des espèces végétales et animales, ou encore en informatique, pour filtrer les courriels nuisibles. Si l'utilisation de la statistique bayésienne a explosé au cours des dernières années, c'est grâce au progrès de nos ordinateurs. C'est aussi grâce à la nature même de l'approche qui permet de coller à notre façon d'apprendre, de raisonner et d'accumuler des connaissances.

Dans ce livre, je vous propose une introduction à la statistique bayésienne. Ce livre est en français parce que c'est la langue dans laquelle je me sens le plus à l'aise pour écrire, et parce que j'aurais aimé avoir plus d'ouvrages dans ma langue maternelle lorsque j'étais étudiant.

Je me suis fixé comme objectifs de 1) synthétiser les aspects méthodologiques à bien comprendre, et 2) fournir les moyens pratiques pour utiliser vous-mêmes la statistique bayésienne. Parce que l'on comprend mieux en faisant, nous utiliserons un logiciel pour pratiquer la statistique. Ce logiciel, c'est R, c'est un logiciel libre pour faire des statistiques et de la science des données. En français, je recommande l'excellent manuel de Julien Barnier, *Introduction à R et au tidyverse*, disponible en ligne[1] et le site du projet collaboratif Analyse-R[2]. Pour la statistique bayésienne en particulier, je présente un package qui propose une syntaxe simple et familière, proche de celle utilisée pour les régressions dans R : brms. Dans la version enrichie de ce livre consultable en ligne[3], je présente aussi un package qui nécessite de programmer (écrire des boucles par exemple), mais qui offre en contrepartie une grande flexibilité : NIMBLE.

Plutôt que dans un style académique, j'ai choisi d'écrire un peu comme si nous étions ensemble dans la même pièce ou en visioconférence, et que je devais vous expliquer de vive voix la statistique bayésienne. Ainsi, je ferai parfois (souvent, en fait) des abus de langage et des approximations mathématiques pour faciliter la compréhension. Vous ne m'en voudrez pas, j'espère.

POURQUOI S'INTÉRESSER À LA STATISTIQUE BAYÉSIENNE ?

La statistique bayésienne est une approche pour analyser les données et prendre des décisions en présence d'incertitude, comme lorsqu'on lance un dé ou que l'on prévoit la météo : on ne peut pas savoir exactement ce qui va se passer, mais on peut estimer les chances des différents résultats. Pourquoi adopter cette approche ? Plusieurs raisons peuvent motiver son utilisation :

– une interprétation naturelle des probabilités : en statistique bayésienne, une probabilité représente un degré de confiance dans une hypothèse ou dans un paramètre, ce qui correspond bien à notre manière intuitive de raisonner face à l'incertitude ;

– une grande flexibilité : le cadre bayésien s'adapte bien à des données incomplètes, hétérogènes ou rares, ainsi qu'à des modèles complexes (hiérarchiques, non linéaires, dynamiques, etc.) ;

– la possibilité d'intégrer des connaissances préalables : on peut capitaliser sur des résultats d'études précédentes ou des avis d'expertises de manière transparente et formalisée ;

– une gestion rigoureuse de l'incertitude : la statistique bayésienne fournit non seulement une estimation des paramètres, mais aussi une mesure directe de l'incertitude associée.

1. https://juba.github.io/tidyverse
2. https://larmarange.github.io/analyse-R/
3. https://oliviergimenez.github.io/statistique-bayes/

CE QUE NOUS ALLONS VOIR DANS CE LIVRE

J'aimerais vous guider dans l'apprentissage de la statistique bayésienne. J'ai rassemblé le matériel qui m'a paru essentiel pour la comprendre et l'appliquer. L'objectif est que vous soyez à l'aise avec l'approche bayésienne et que vous puissiez l'appliquer à vos propres données. Les objectifs sont de :

– démythifier la statistique bayésienne et les méthodes de Monte Carlo par chaînes de Markov (MCMC) ;

– comprendre les différences entre approche bayésienne et approche fréquentiste ;

– lire et comprendre les sections « méthodes » des articles scientifiques utilisant l'approche bayésienne ;

– savoir mettre en œuvre vos analyses avec la statistique bayésienne dans R.

Le chapitre 1 pose les bases en revenant sur quelques rappels de probabilité utiles pour la suite. Ce sera aussi l'occasion d'introduire les notions clés de la statistique bayésienne, à travers un exemple simple qui permettra de fixer les idées.

Dans le chapitre 2, nous passerons dans les coulisses de la statistique bayésienne, avec les méthodes MCMC, qui rendent l'inférence possible en pratique. Nous mettrons un peu la main à la pâte en codant nous-mêmes une analyse bayésienne.

Le chapitre 3 présentera brms, un outil très utile pour faire de la statistique bayésienne sans trop d'efforts. Dans la version enrichie du livre[3], je présenterai aussi NIMBLE. Grâce à ces outils, nul besoin de tout faire soi-même.

Le chapitre 4 sera consacré aux distributions a priori. Nous verrons comment les choisir avec pertinence, comment traduire de l'information existante sous forme de prior, et les pièges à éviter.

Dans le chapitre 5, nous verrons comment faire une régression linéaire en statistique bayésienne. Nous en profiterons pour illustrer la comparaison et la validation des modèles. Nous utiliserons brms (et NIMBLE) et le comparerons à l'approche fréquentiste.

Le chapitre 6 nous emmènera vers les modèles linéaires généralisés, avec ou sans effets aléatoires – des modèles très utilisés en pratique. Nous nous appuierons sur la simulation de données, un outil précieux pour bien comprendre ce que fait un modèle. Je vous montrerai comment faire ces analyses avec brms (et avec NIMBLE) et nous les comparerons aux analyses fréquentistes.

Enfin, un dernier chapitre viendra résumer les messages clés du livre et proposer quelques conseils pour appliquer la statistique bayésienne.

COMMENT LIRE CE LIVRE ?

Je n'ai pas vraiment de conseil à vous donner sur la meilleure manière de lire ce livre. Personnellement, je trouve toujours difficile d'absorber toute l'information contenue dans un ouvrage. Vous pouvez lire en continu ou bien grappiller des éléments de-ci de-là.

Dans chacun des chapitres, le code R est fourni, je l'ai aussi hébergé sur mon site[4] et le mettrai à jour. S'exercer permet de mieux comprendre et de vérifier que l'on a bien assimilé. Si vous lisez la version électronique de cet ouvrage, vous pouvez copier les lignes de code puis les coller dans R pour les exécuter. Pour gagner un peu de place, et éviter de perturber trop la lecture, certains codes ne sont pas donnés, en particulier ceux qui permettent de produire les figures, mais ils sont disponibles en ligne[4].

Si vous voulez aller plus loin, vous trouverez une liste bibliographique enrichie à la fin de ce livre. Les ouvrages Biobayes collectif (2015), McCarthy (2007), Kéry et Kellner (2010), Johnson *et al.* (2022), Kruschke (2010), Gelman *et al.* (2013) et McElreath (2020) ont été une source d'inspiration dans la rédaction de ce livre. J'ai hésité à donner plus de références et à citer (beaucoup) d'articles scientifiques, mais je ne le ferai pas. Les ouvrages cités sont largement suffisants.

4. https://github.com/oliviergimenez/statistique-bayes-quae

1. L'approche bayésienne

Dans ce chapitre, nous posons les bases en faisant quelques rappels de probabilité utiles pour la suite. J'introduis les notions clés de la statistique bayésienne à travers un exemple simple qui permet de fixer les idées, et que nous utiliserons souvent dans le livre. Nous ferons aussi des parallèles entre la statistique classique ou fréquentiste et la statistique bayésienne.

1.1. LE THÉORÈME DE BAYES

Ne tardons plus et entrons dans le vif du sujet. La statistique bayésienne repose sur le théorème de Bayes (ou formule de Bayes), dont on doit la première formulation au mathématicien et révérend Thomas Bayes. Ce théorème a été publié en 1763, deux ans après la mort de Bayes, grâce aux efforts de son ami Richard Price. Il a par ailleurs été découvert indépendamment par Pierre-Simon Laplace.

Le théorème de Bayes porte sur les probabilités conditionnelles, qui sont parfois un peu délicates à comprendre. La probabilité conditionnelle d'un événement A sachant un événement B, que l'on note Pr($A \mid B$), est la probabilité que A se produise, révisée en tenant compte de l'information supplémentaire que l'événement B s'est produit. Par exemple, imaginez qu'un de vos amis lance un dé (équilibré) et vous demande la probabilité que le résultat soit un six (A). Votre réponse est 1/6, car chaque face du dé a la même chance d'apparaître. Maintenant, imaginez que l'on vous dise que le nombre obtenu est pair (B) avant de répondre. Comme il n'y a que trois nombres pairs, dont un seul est six, vous pouvez réviser votre réponse : Pr($A \mid B$) = 1/3.

Vous voyez comment l'information supplémentaire (ici, savoir que le nombre est pair) change l'estimation ? C'est exactement ce type de raisonnement que le théorème de Bayes formalise et généralise : il permet de calculer la probabilité d'un événement A sachant qu'un autre événement B s'est produit. Plus précisément, le théorème de Bayes vous donne Pr($A \mid B$) en utilisant les probabilités marginales Pr(A) et Pr(B) et la probabilité Pr($B \mid A$) :

$$\Pr\left(A \mid B\right) = \frac{\Pr\left(B \mid A\right)\Pr\left(A\right)}{\Pr\left(B\right)}$$

On parle de probabilité marginale quand on s'intéresse à la probabilité d'un événement « tout seul », sans condition particulière. Par exemple, Pr(A) ou Pr(B), ce sont les chances globales de A ou de B, sans tenir compte d'autre chose. On dit « marginale » parce que, si vous faisiez un tableau avec toutes les combinaisons possibles (par exemple les résultats d'un dé classés en pair/impair et en six/pas six), Pr(A) ou Pr(B) s'obtiendraient alors en additionnant les cases d'une ligne ou d'une colonne, autrement dit ce qu'on lit en marge du tableau.

Le théorème de Bayes est souvent considéré comme un moyen de remonter d'un effet B à une cause A inconnue, en connaissant la probabilité de l'effet B sachant la cause A. Pensez, par exemple, à une situation où un diagnostic médical est requis, avec A, une maladie inconnue et B, des symptômes ; la médecin connaît les risques d'avoir certains symptômes en fonction de plusieurs maladies ou Pr(symptômes | maladie), et souhaite déduire la probabilité d'avoir une maladie connaissant les symptômes ou Pr(maladie | symptômes). Cette manière de renverser Pr($B \mid A$) en Pr($A \mid B$) explique pourquoi le raisonnement bayésien est parfois appelé « probabilité inverse ».

Plutôt que d'utiliser des lettres au risque de s'embrouiller, je trouve plus facile de retenir le théorème de Bayes écrit ainsi :

$$\Pr\left(\text{hypothèse} \mid \text{données}\right) = \frac{\Pr\left(\text{données} \mid \text{hypothèse}\right)\Pr\left(\text{hypothèse}\right)}{\Pr\left(\text{données}\right)}$$

L'hypothèse peut être un paramètre, comme la probabilité d'apparition d'une maladie, ou des coefficients de régression liant cette probabilité à des facteurs de risque (par exemple le lieu de vie, le tabagisme). Le théorème de Bayes nous dit comment obtenir la probabilité d'une hypothèse à partir des données disponibles.

C'est pertinent car, réfléchissez-y, c'est exactement ce que fait la méthode scientifique. Nous voulons savoir à quel point une hypothèse est plausible à partir de données que nous avons collectées, et peut-être comparer plusieurs hypothèses entre elles. De ce point de vue, le raisonnement bayésien s'accorde avec le raisonnement scientifique, ce qui explique probablement pourquoi le cadre bayésien est si naturel pour faire et comprendre la statistique.

Vous pourriez alors demander pourquoi la statistique bayésienne n'est-elle pas la norme ? Pendant longtemps, la mise en œuvre du théorème de Bayes a été limitée par des difficultés de calcul, comme nous le verrons au chapitre suivant. Fort heureusement, l'amélioration de la puissance de calcul de nos ordinateurs et le développement de nouveaux algorithmes ont entraîné un essor marqué de l'approche bayésienne au cours des trente dernières années.

1.2. QU'EST-CE QUE LA STATISTIQUE BAYÉSIENNE ?

Les problèmes statistiques typiques consistent à estimer un paramètre (ou plusieurs) à partir de données disponibles. On va noter ce ou ces paramètres de manière générique, disons θ. Pour estimer θ, vous êtes sûrement plus familier de l'approche fréquentiste que de l'approche bayésienne. L'approche fréquentiste, en particulier l'estimation par maximum de vraisemblance, suppose que les paramètres sont fixes et de valeurs inconnues. Les estimateurs classiques sont donc en général des valeurs ponctuelles ; par exemple, un estimateur de la probabilité d'obtenir une face d'un dé est le nombre de fois qu'on a obtenu cette face, divisé par le nombre de fois qu'on a lancé ce dé. L'approche bayésienne suppose que les paramètres ne sont pas fixes et suivent une distribution inconnue. Une distribution de probabilité est une expression mathématique qui donne la probabilité qu'une variable aléatoire prenne certaines valeurs. Elle peut être discrète (par exemple la loi de Bernoulli, la loi binomiale ou la loi de Poisson) ou continue (comme la loi normale ou gaussienne).

L'approche bayésienne repose sur l'idée que vous commencez avec certaines connaissances sur le système avant même de l'étudier vous-même. Ensuite, vous collectez des données et mettez à jour ces connaissances a priori en fonction des observations. Ce processus de mise à jour repose sur le théorème de Bayes. De façon simplifiée, si l'on prend $A = \theta$ et $B =$ données, alors le théorème de Bayes permet d'estimer le paramètre θ à partir des données comme suit :

$$\Pr\left(\theta \mid \text{données}\right) = \frac{\Pr\left(\text{données} \mid \theta\right) \times \Pr\left(\theta\right)}{\Pr\left(\text{données}\right)}$$

Prenons un peu de temps pour passer en revue chaque terme de cette formule.

À gauche, nous avons Pr(θ | données), la distribution a posteriori. La probabilité de θ sachant les données. Elle représente ce que vous savez de θ après avoir vu les données. C'est la base de l'inférence et c'est précisément ce que vous cherchez : une distribution, possiblement multivariée si vous avez plusieurs paramètres.

À droite, on trouve Pr(données | θ), la vraisemblance (ou *likelihood* en anglais). La probabilité des données sachant θ. Cette quantité est la même que dans l'approche classique ou fréquentiste. Car oui, les approches bayésienne et fréquentiste partagent la même composante, la vraisemblance, ce qui explique pourquoi leurs résultats sont souvent proches. La vraisemblance exprime l'information que contiennent vos données, étant donné un modèle paramétré par θ. On en reparle à la section 1.4.

Ensuite, nous avons Pr(θ) la distribution a priori. Cette quantité représente ce que vous savez sur θ avant de voir les données. Cette distribution a priori ne doit pas dépendre des données, autrement dit, on ne devrait pas utiliser les données pour la construire. Elle peut être vague ou non informative si vous ne savez rien sur θ. Souvent, vous ne partez jamais de zéro, et dans l'idéal vous souhaiteriez que votre a priori reflète les connaissances existantes. Je vous parlerai plus en détail des priors au chapitre 4.

Enfin, il y a le dénominateur Pr(données), parfois appelé vraisemblance moyenne (*average likelihood*), moyenne par rapport au prior, car il s'obtient en intégrant la vraisemblance selon la loi a priori Pr(données) = $\int$ Pr(données | θ) × Pr(θ) $d\theta$. Cette quantité permet de normaliser la distribution a posteriori pour qu'elle intègre à 1. Autrement dit, comme $\int$ Pr(θ | données) $d\theta$ = 1 puisque l'intégrale d'une densité de probabilité vaut 1, on a $\int \frac{\Pr(\text{données} \mid \theta) \times \Pr(\theta)}{\Pr(\text{données})} d\theta = 1$.

Et puisque Pr(données) ne dépend pas de θ, on a Pr(données) = $\int$ Pr(données | θ) × Pr(θ) $d\theta$. C'est une intégrale de dimension égale au nombre de paramètres θ à estimer : pour deux paramètres, une intégrale double, pour trois paramètres, une intégrale triple, etc. Or, au-delà de trois dimensions, il devient difficile, voire impossible, de calculer cette intégrale. C'est l'une des raisons qui expliquent que l'approche bayésienne n'a pas été utilisée plus tôt et que l'on a besoin d'algorithmes pour estimer les distributions a posteriori, comme je l'explique dans le chapitre 2. En attendant, on va dérouler un exemple relativement simple dans lequel la distribution a posteriori a une forme explicite.

1.3. UN EXEMPLE FIL ROUGE

Prenons un exemple concret pour fixer les idées. Je travaille sur le ragondin (*Myocastor coypus*) (**figure 1.1**), un rongeur semi-aquatique originaire d'Amérique du Sud, introduit en Europe pour l'élevage de fourrure. Il est aujourd'hui considéré comme une espèce exotique envahissante, en raison des dégâts qu'il occasionne dans les milieux humides (érosion des berges, destruction de la végétation) et de son rôle possible dans la transmission à l'humain de la leptospirose, une infection bactérienne potentiellement sévère, transmise *via* l'eau. Grâce à sa forte fécondité et à sa bonne adaptation aux climats tempérés, le ragondin a proliféré rapidement.

Une des questions qui m'intéressent est d'estimer la probabilité qu'ils survivent à l'hiver, les ragondins étant particulièrement sensibles au froid. Pour cela, on équipe plusieurs individus d'une balise GPS au début de l'hiver, disons ici n = 57. À la fin de l'hiver, on observe que y = 19 ragondins sont encore vivants. L'objectif est d'estimer la probabilité de survie hivernale que l'on notera θ. Voici les données :

```
y <- 19 # nombre d'individus ayant survécu à l'hiver
n <- 57 # nombre d'individus suivis au début de l'hiver
```

Vous vous dites sûrement qu'avec ces éléments on peut déjà estimer une probabilité de survie. Intuitivement, on pense à la proportion d'individus ayant survécu, soit 19/57. Et vous n'avez pas tort. C'est une estimation raisonnable de θ, la probabilité de survie hivernale. Essayons maintenant de formaliser cette intuition, afin de mieux comprendre ce qu'elle représente, et ce qu'elle suppose.

Comme mentionné plus haut, la vraisemblance est une idée centrale que l'on retrouve dans les approches fréquentiste et bayésienne. Commençons donc par construire cette vraisemblance. Pour cela, il nous faut poser quelques hypothèses.

Premièrement, nous supposons que les individus sont indépendants, c'est-à-dire que la survie d'un ragondin n'influence pas la survie des autres ragondins. C'est une hypothèse forte, surtout quand on sait qu'une femelle peut se reproduire deux à trois fois par an et donner naissance jusqu'à dix petits dépendants d'elle au début de leur vie. Mais, en modélisation, il vaut souvent mieux commencer simplement.

Figure 1.1. Cliché de ragondins (*Myocastor coypus*) pris sur le bassin-versant du Lez dans les environs de Montpellier. Crédits : Yann Raulet.

Deuxièmement, nous supposons que tous les individus ont la même probabilité de survie. Là encore, c'est une simplification : on sait, par exemple, que la mortalité des jeunes est plus élevée que celle des adultes.

Sous ces deux hypothèses, le nombre y d'animaux encore vivants à la fin de l'hiver suit une loi binomiale, avec θ comme probabilité de succès (survie), et n comme nombre d'essais (individus suivis). On notera $y \sim \text{Bin}(n,\theta)$. La loi binomiale est en fait la somme de plusieurs épreuves de Bernoulli indépendantes, comme dans l'exemple classique du « pile ou face ». À chaque lancé, ici le relâché d'un ragondin équipé d'un GPS en début d'hiver, on suppose une probabilité θ de succès, c'est-à-dire de survivre à l'hiver, et d'échec, c'est-à-dire de mourir de froid. Si toutes ces épreuves sont indépendantes et ont la même probabilité de succès (nos hypothèses), alors le nombre de succès ou de ragondins vivants en sortie d'hiver suit une loi binomiale (voir aussi le chapitre 6). Je donne des exemples de tirages Bernoulli et binomiaux dans la **figure 1.2**[5].

Au passage, il est facile de s'embrouiller entre tous les termes utilisés pour décrire une Bernoulli et une binomiale (et la normale) : vous pouvez retenir qu'une probabilité est un nombre, une distribution est une loi, une densité est la fonction qui la représente.

1.4. LE MAXIMUM DE VRAISEMBLANCE

Dans l'approche classique (ou fréquentiste), on estime la probabilité de survie θ en utilisant la méthode du maximum de vraisemblance. Mais qu'est-ce que cela signifie concrètement ? Il s'agit de trouver la valeur de θ qui rend les données observées les plus probables. Autrement dit, puisque les données sont ce qu'elles sont – elles ont été observées – on cherche la valeur de θ qui maximise la probabilité que cet ensemble de données ait été produit.

Comment justifie-t-on mathématiquement cette idée plutôt intuitive ? Relisez bien la fin du paragraphe précédent. L'idée de chercher la valeur qui donne la plus grande probabilité revient à maximiser quelque chose. Mais quoi exactement ? La probabilité des données, étant donné un certain modèle paramétré par θ, autrement dit la vraisemblance ou $\Pr(\text{données} \mid \theta)$, vue à la

5. Dans tout l'ouvrage, pour une question de cohérence entre les lignes de code, les figures (dans R) et le texte, les parties entières et décimales des nombres seront séparées par un point et non une virgule.

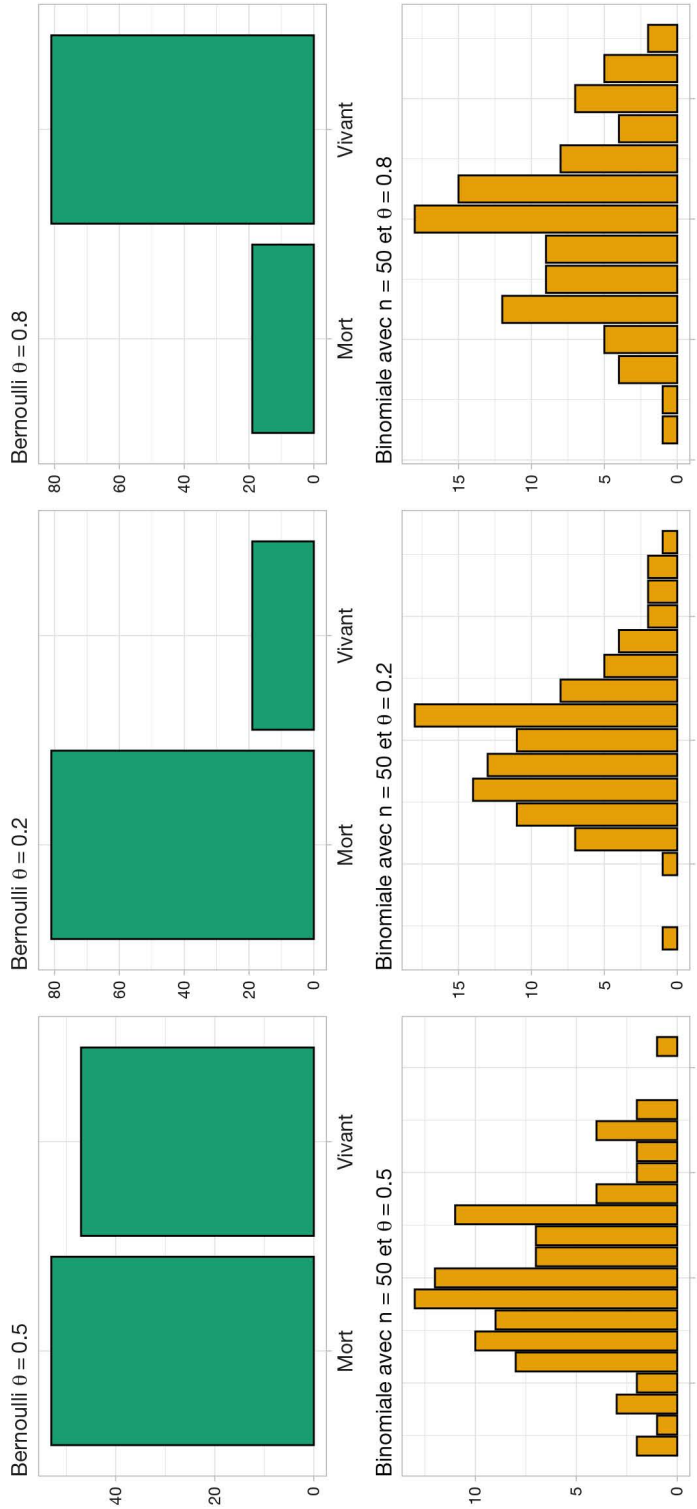

Figure 1.2. Distributions de probabilité discrètes, Bernoulli et binomiale, illustrées avec 100 simulations (des tirages aléatoires générés par ordinateur). On représente, sur la ligne du haut, la fréquence observée d'un tirage Bernoulli (mort ou vivant) pour différentes valeurs de probabilité de survie θ (0.5, 0.2 et 0.8) ; et sur la ligne du bas, les histogrammes pour un tirage binomial (nombre de survivants) avec 50 tentatives et différentes valeurs de probabilité de survie θ (0.5, 0.2 et 0.8).
Les codes pour reproduire les figures sont disponibles en ligne : https://github.com/oliviergimenez/statistique-bayes-quae

section 1.2. L'estimation classique repose donc sur la maximisation de la vraisemblance, ou plutôt de la fonction de vraisemblance, c'est-à-dire la vraisemblance considérée comme une fonction de θ.

Dans notre cas, on est face à une expérience binomiale : on suit n ragondins à travers l'hiver, chacun ayant une probabilité θ de survivre. On connaît la probabilité de chaque issue possible (ou fonction de masse). Par exemple, la probabilité qu'aucun ragondin ne survive est $(1 - \theta)^n$, car chacun des n individus meurt avec une probabilité $1 - \theta$. Si l'on prend par exemple une probabilité de survie de 0.5, on a $(1 - 0.5)^{57} \approx 0$. On peut calculer cette probabilité dans R avec la fonction dbinom() :

```
dbinom(x = 0, size = 57, prob = 1 - 0.5)
#> [1] 6.938894e-18
```

où le premier argument `x = 0` est pour aucun ragondin vivant. À l'inverse, la probabilité que tous survivent est θ^n qui vaut la même chose. Vous pouvez vérifier avec R et la ligne de commande dbinom(x = 57, size = 57, prob = 0.5). Si un seul ragondin survit, il faut que l'un des n survive avec une probabilité θ, et que les $n - 1$ autres meurent avec une probabilité $(1 - \theta)^{n-1}$. Comme n'importe lequel des n ragondins peut être celui qui survit, on obtient une probabilité totale de $n\theta(1 - \theta)^{n-1}$. On peut calculer cette probabilité avec dbinom(x = 1, size = 57, prob = 0.5). Plus généralement, la probabilité que y individus survivent est donnée par $\binom{n}{y}\theta^y\left(1-\theta\right)^{n-y}$. Si l'on considère cette expression comme une fonction de θ (et non de y), on obtient la fonction de vraisemblance $\mathcal{L}\left(\theta\right) = \binom{n}{y}\theta^y\left(1-\theta\right)^{n-y}$. Le terme $\binom{n}{y}$ est appelé coefficient binomial et se lit « y parmi n ». Il correspond au nombre de façons différentes de choisir y survivants parmi les n ragondins, sans tenir compte de leur ordre.

On peut représenter cette vraisemblance dans R comme dans la **figure 1.3**.

Notre objectif est de trouver la valeur de θ qui maximise cette fonction. Autrement dit, on cherche la valeur de survie (sur l'axe des abscisses dans la **figure 1.3**) qui maximise la vraisemblance

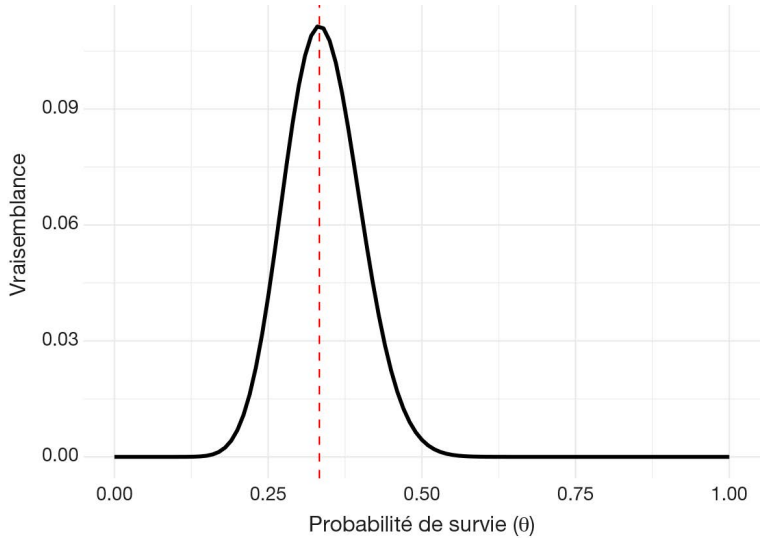

Figure 1.3. Fonction de vraisemblance pour la probabilité de survie hivernale du ragondin, calculée à partir de y = 19 survivants sur n = 57 individus suivis par GPS. Le maximum de vraisemblance est indiqué par le pointillé rouge.

(sur l'axe des ordonnées). Cette valeur correspond à l'estimateur du maximum de vraisemblance, souvent noté $\hat{\theta}$. Pour ce faire, il est souvent plus pratique de travailler avec le logarithme de la vraisemblance (la log-vraisemblance), car les sommes sont plus stables numériquement et plus faciles à dériver que les produits :

$$\ell\left(\theta\right) = \log \mathcal{L}\left(\theta\right) = \log \binom{n}{y} + y \log \theta + \left(n - y\right) \log\left(1 - \theta\right)$$

Le premier terme, $\log \binom{n}{y}$, ne dépend pas de θ, on peut donc l'ignorer pour la suite. On dérive alors la log-vraisemblance par rapport à θ :

$$\frac{d\ell\left(\theta\right)}{d\theta} = \frac{y}{\theta} - \frac{n - y}{1 - \theta}$$

On cherche la valeur de θ qui annule cette dérivée :

$$\frac{y}{\theta} - \frac{n - y}{1 - \theta} = 0$$

Après quelques simplifications, on obtient que l'estimateur du maximum de vraisemblance $\hat{\theta}$ est :

$$\hat{\theta} = \frac{y}{n}$$

Ce résultat rejoint notre intuition de départ : l'estimateur du maximum de vraisemblance est la proportion d'individus qui ont survécu, soit 19/57 ≈ 0.333. On peut visualiser ce résultat sur la **figure 1.3**, où le maximum de vraisemblance est indiqué par le pointillé rouge.

En pratique, les modèles contiennent plusieurs paramètres, des dizaines voire des centaines, et on ne peut pas appliquer la même méthode pour maximiser la vraisemblance et trouver les estimateurs du maximum de vraisemblance. On utilise plutôt des algorithmes itératifs d'optimisation, qui vont résoudre le problème pour nous, en ajustant pas-à-pas une valeur initiale jusqu'à trouver celle qui maximise la vraisemblance. Par exemple, dans R, on peut obtenir exactement ce même résultat en utilisant une régression logistique sans covariable (voir chapitre 6) :

```
mod <- glm(cbind(y, n - y) ~ 1, family = binomial)
theta_hat <- plogis(coef(mod))
theta_hat
#> (Intercept)
#>   0.3333333
```

Le calcul direct $\hat{\theta} = y/n$ et le résultat de l'appel à la fonction glm() sont cohérents, ils donnent la même valeur.

1.5. ET EN BAYÉSIEN ?

Dans l'approche bayésienne, on commence par exprimer nos connaissances a priori sur la quantité que l'on souhaite estimer. Ici, la probabilité de survie hivernale θ. On sait que θ est une variable continue comprise entre 0 et 1. Une distribution a priori naturelle dans ce cas est la distribution bêta. La distribution bêta est définie par deux paramètres a et b qui en contrôlent la forme :

$$q\left(\theta \mid a, b\right) = \frac{1}{\text{Beta}\left(a, b\right)} \theta^{a-1} \left(1 - \theta\right)^{b-1}$$

avec :

$$\text{Beta}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \Gamma(n) = (n-1)\times(n-2)\times\ldots\times 2\times 1$$

Vous pouvez oublier ces équations si vous n'êtes pas à l'aise avec. Essayons plutôt de visualiser cette distribution comme dans la **figure 1.4**.

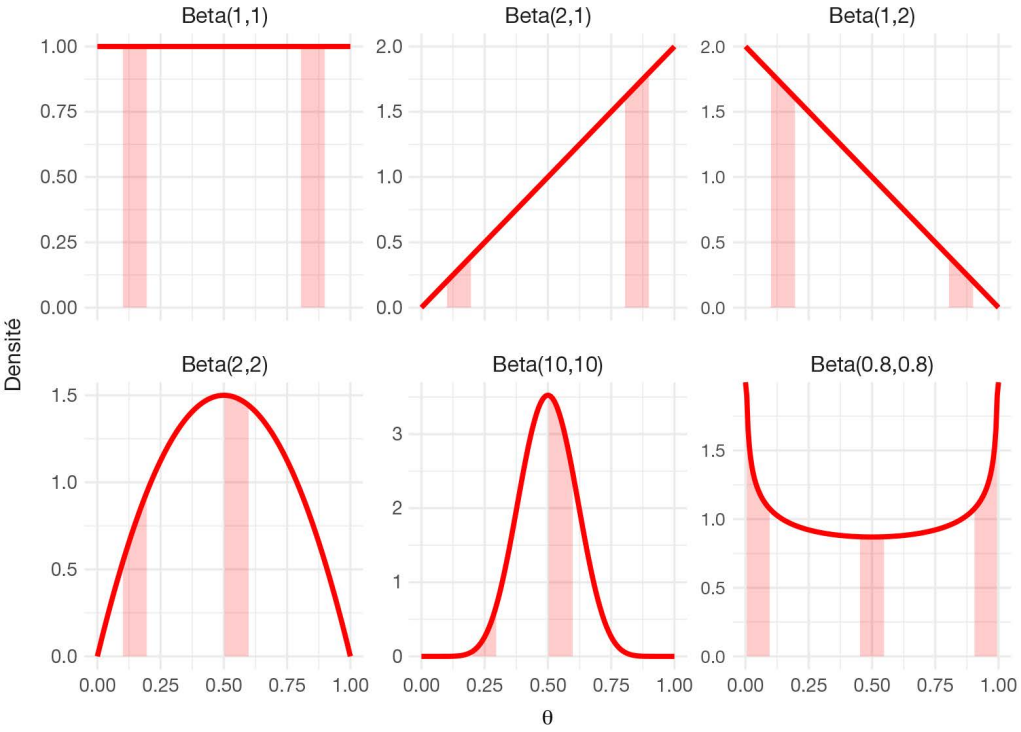

Figure 1.4. Exemples de lois bêta pour différentes valeurs des paramètres *a* et *b*. Dans chaque panneau, les zones ombrées illustrent la probabilité d'observer une valeur dans un intervalle donné.

Chaque panneau de la figure montre la forme d'une loi bêta pour un couple de paramètres (a,b) donné. On peut y observer plusieurs comportements caractéristiques.

Beta(1,1) (en haut à gauche) correspond à la loi uniforme entre 0 et 1 : toutes les valeurs de θ entre 0 et 1 sont considérées comme également probables. La densité est constante, ce qui signifie que la probabilité d'observer une valeur entre 0.1 et 0.2 est la même que celle d'en observer une entre 0.8 et 0.9. Cette probabilité est l'aire du rectangle délimité par le trait rouge et les lignes verticales calées à 0.1 et 0.2 ou 0.8 et 0.9 (les zones ombrées en rouge). On a une situation d'absence de connaissance a priori.

Beta(2,1) et Beta(1,2) représentent des connaissances asymétriques : la première est biaisée vers des valeurs proches de 1, la seconde vers des valeurs proches de 0. La probabilité d'observer une valeur entre 0.1 et 0.2 est plus petite que celle d'en observer une entre 0.8 et 0.9, et vice versa.

Beta(2,2) est symétrique, mais donne plus de poids aux valeurs centrales qu'une loi uniforme. La probabilité d'observer une valeur entre 0.1 et 0.2 est plus petite que celle d'en observer une entre 0.5 et 0.6.

Beta(10,10) représente une connaissance très concentrée autour de 0.5 : c'est un prior très informatif. La probabilité d'observer une valeur entre 0.2 et 0.3 est beaucoup plus petite que celle d'en observer une entre 0.5 et 0.6.

Beta(0.8,0.8) illustre une distribution en forme de U ou de baignoire, qui favorise les valeurs extrêmes (proches de 0 ou de 1). Les probabilités d'observer une valeur entre 0 et 0.1 et entre 0.9 et 1 sont plus grandes que celle d'en observer une entre 0.45 et 0.55.

Ces exemples permettent de visualiser comment les paramètres a et b influencent la forme du prior. Comment passe-t-on de ce prior à la distribution a posteriori ?

On suppose que $\theta \sim \text{Beta}(a,b)$ et on a observé $y = 19$ survivants parmi $n = 57$ individus. La vraisemblance est $\binom{n}{y}\theta^y\left(1-\theta\right)^{n-y}$. Pour l'instant, on va ignorer le dénominateur $\Pr(y)$ dans le théorème de Bayes, on verra dans le chapitre suivant pourquoi. On a donc l'a posteriori proportionnel au produit de la vraisemblance et de l'a priori : $\Pr\left(\theta \mid y\right) \propto \Pr\left(y \mid \theta\right) \times \Pr\left(\theta\right)$. Dans notre cas, on multiplie terme à terme la vraisemblance et le prior et, en réarrangeant les termes en θ et $1 - \theta$, on a :

$$\Pr\left(\theta \mid y\right) \propto \underbrace{\theta^y\left(1-\theta\right)^{n-y}}_{\text{vraisemblance binominale}} \times \underbrace{\theta^{a-1}\left(1-\theta\right)^{b-1}}_{\text{a priori bêta}}$$

$$\propto \underbrace{\theta^{a+y-1}\left(1-\theta\right)^{b+n-y-1}}_{\text{forme d'une loi bêta}}$$

Autrement dit, on retrouve une loi bêta, avec des paramètres mis à jour $a + y$ et $b + n - y$. On dit que la loi binomiale et la loi bêta sont conjuguées : lorsque l'on utilise une loi bêta comme distribution a priori pour un paramètre de probabilité dans un modèle binomial, la loi a posteriori obtenue est également une loi bêta. Si l'on utilise une loi uniforme a priori en 0 et 1 (autrement

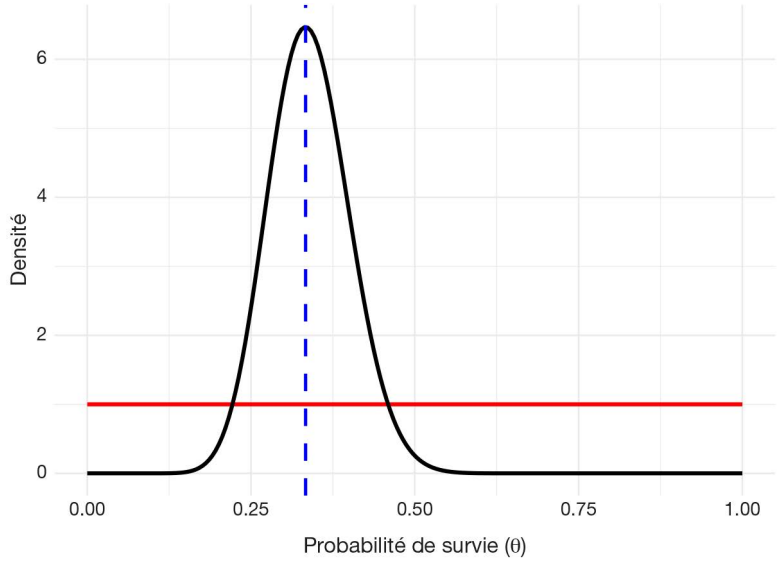

Figure 1.5. Distribution a priori uniforme (rouge) et distribution a posteriori (noire) de la probabilité de survie hivernale du ragondin. Le pointillé bleu correspond à l'estimateur du maximum de vraisemblance.

dit, une Beta(1,1)), on obtient que la distribution a posteriori de la survie hivernale est une Beta(1 + 19.1 + 57 – 19) = Beta(20,39). Au passage, la distribution a posteriori est connue, ce qui facilite grandement les calculs et leur interprétation. Par exemple, on sait que la moyenne de la Beta(a,b) est $\frac{a}{a+b}$, soit $\frac{20}{59} \approx 0.339$. On peut comparer cette valeur à l'estimateur du maximum de vraisemblance 19/57 ≈ 0.333. On peut également visualiser la distribution a posteriori comme dans la **figure 1.5**, puisque l'on connaît l'équation de la densité d'une loi bêta.

Plus généralement, lorsque l'on dispose de suffisamment de données, les estimateurs bayésien et fréquentiste ont tendance à être très proches. Intuitivement, les données finissent par « dominer » l'information a priori. Grossièrement, le mode de la distribution a posteriori (la valeur pour laquelle la densité est maximale) correspond exactement à l'estimateur du maximum de vraisemblance.

C'est une illustration du lien entre les deux approches et du rôle central que joue la vraisemblance en statistique : elle constitue le point commun fondamental entre les approches bayésienne et fréquentiste.

À RETENIR

- Le théorème de Bayes est un outil de mise à jour des connaissances.
- La statistique bayésienne repose sur la vraisemblance et sur une loi a priori pour les paramètres du modèle.
- La statistique fréquentiste fournit un estimateur ponctuel quand la statistique bayésienne estime une distribution pour chaque paramètre.
- Souvent, les approches classique et bayésienne donnent des estimations proches.
- Dans certains cas, la loi a posteriori est explicite (par exemple dans le cas de la conjugaison bêta/binomiale).
- Dans la plupart des cas, il nous faudra utiliser des simulations pour obtenir la distribution a posteriori (voir chapitre 2).

2. Les méthodes de Monte Carlo par chaînes de Markov

J'espère que je ne vous ai pas (trop) perdu dans le chapitre précédent avec toutes ces équations. Dans ce nouveau chapitre, nous passons dans les coulisses de la statistique bayésienne, en découvrant les méthodes de Monte Carlo par chaînes de Markov (MCMC). Vous verrez comment et pourquoi ces techniques de simulation sont devenues essentielles pour mettre en œuvre l'inférence bayésienne en pratique. Et comme rien ne vaut la pratique, nous mettrons un peu la main à la pâte en codant nous-mêmes, à travers notre exemple fil rouge sur l'estimation d'une probabilité de survie.

2.1. APPLICATION DU THÉORÈME DE BAYES

Revenons à notre exemple fil rouge sur les ragondins. Je redonne les données :

```
y <- 19 # nombre d'individus ayant survécu à l'hiver
n <- 57 # nombre d'individus suivis au début de l'hiver
```

Appliquons le théorème de Bayes de manière plus directe que dans le chapitre 1, dans lequel on a mis de côté le dénominateur Pr(données). Voyons s'il est possible de faire avec. Comme on l'a vu, ce dénominateur est donné par $\Pr(y) = \int \Pr(\text{données} \mid \theta)\ \Pr(\theta)\ d\theta$. On va donc devoir calculer cette intégrale. Commençons par écrire une fonction R qui calcule le produit de la vraisemblance (binomiale) par le prior Beta(1,1), c'est-à-dire le numérateur dans le théorème de Bayes $\Pr(\text{données} \mid \theta) \times \Pr(\theta)$:

```
num <- function(theta) dbinom(y, n, theta) * dbeta(theta, 1, 1)
```

Nous pouvons maintenant écrire la fonction qui calcule le dénominateur et, pour ce faire, nous allons utiliser la fonction de R qui permet de calculer l'intégrale d'une fonction d'une variable : integrate. La fonction integrate() met en œuvre des techniques de quadrature pour diviser en petits carrés l'aire sous la courbe délimitée par la fonction à intégrer, et pour les compter.

```
den <- integrate(num, 0, 1)$value
```

Nous obtenons alors une approximation numérique de la distribution a posteriori de la survie hivernale comme dans la **figure 2.1**.

```
# Crée une grille de valeurs pour la probabilité de survie (entre 0 et 1)
grid <- seq(0, 1, 0.01)

# Calcule les valeurs de la densité a posteriori sur la grille
# num(grid) est la vraisemblance*prior, den est la constante de normalisation
posterior <- data.frame(
  survival = grid,
  ratio = num(grid) / den  # densité a posteriori normalisée
)

# Trace la courbe de la densité a posteriori
posterior %>%
  ggplot(aes(x = survival, y = ratio)) +
  geom_line(size = 1.5) +
  labs(x = "Probabilité de survie", y = "Densité") +
  theme_minimal()
```

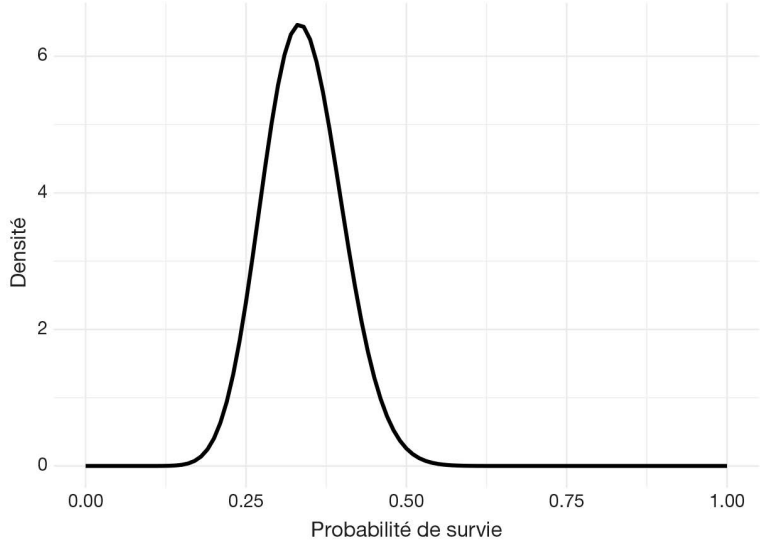

Figure 2.1. Approximation numérique de la distribution a posteriori de la survie hivernale.

Quelle est la qualité de cette approximation numérique? Idéalement, on voudrait comparer l'approximation à la véritable distribution postérieure. Ça tombe bien, on l'a obtenue dans le chapitre 1, il s'agit d'une distribution bêta de paramètres 20 et 39. On peut se rendre compte dans la **figure 2.2** que les deux courbes se superposent parfaitement.

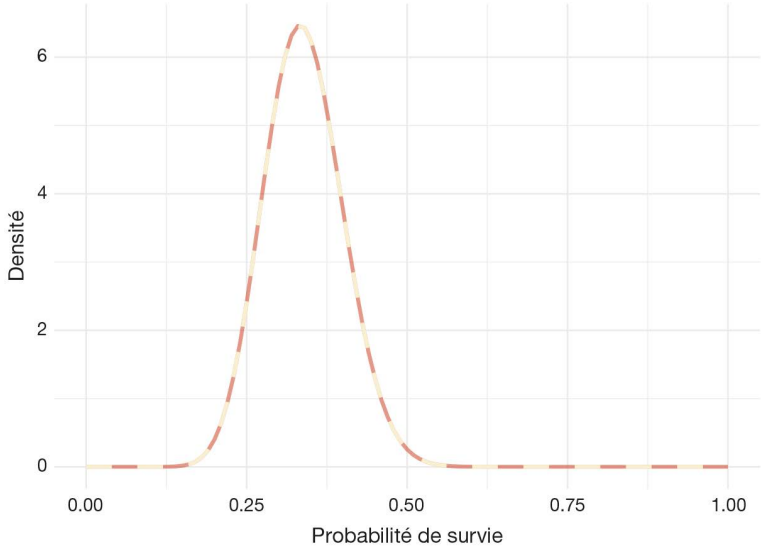

Figure 2.2. Comparaison entre la postérieure exacte (couleur rouge brique) et l'approximation numérique (couleur crème).

La distribution postérieure exacte (couleur rouge brique) et l'approximation numérique (couleur crème) de la survie hivernale ne peuvent être distinguées, ce qui suggère que l'approximation numérique est plus que satisfaisante.

Dans notre exemple, nous avons un seul paramètre à estimer : la survie hivernale. Cela signifie qu'il s'agit d'une intégrale unidimensionnelle au dénominateur, ce qui est assez facile avec les techniques de quadrature et la fonction R integrate().

Mais que se passe-t-il si nous avons plusieurs paramètres ? Par exemple, imaginez que vous vouliez ajuster un modèle de régression avec une probabilité de survie qui dépend d'une variable explicative, par exemple la masse des ragondins. L'effet de cette variable est capturé par le paramètre de régression β_0 (l'ordonnée à l'origine), β_1 la pente associée, et on a aussi l'erreur résiduelle avec l'écart-type σ (voir chapitre 5). Le théorème de Bayes donne alors la distribution postérieure jointe de ces paramètres (c'est-à-dire des trois paramètres ensemble) :

$$\Pr\left(\beta_0, \beta_1, \sigma \mid y\right) = \frac{\Pr\left(y \mid \beta_0, \beta_1, \sigma\right) \times \Pr\left(\beta_0, \beta_1, \sigma\right)}{\iiint \Pr\left(y \mid \beta_0, \beta_1, \sigma\right) \Pr\left(\beta_0, \beta_1, \sigma\right) d\beta_0 d\beta_1 d\sigma}$$

Il y a deux défis numériques majeurs :
– souhaite-t-on vraiment calculer une intégrale triple ? Non, car les méthodes classiques ne vont pas beaucoup plus loin que deux dimensions ;
– on s'intéresse souvent aux distributions marginales (voir section 1.1) des paramètres (par exemple celle de β_1 correspondant à l'effet de la masse sur la survie), obtenues en intégrant la distribution a posteriori jointe sur les autres paramètres (ici une intégrale double par rapport à β_0 et σ) – ce qui devient vite incalculable quand leur nombre augmente.

Dans la section suivante, nous introduisons des méthodes de simulation puissantes pour surmonter ces limitations.

2.2. LES ALGORITHMES MCMC

En bref, l'idée des méthodes MCMC est d'utiliser des simulations pour approximer les distributions a posteriori avec une certaine précision, en tirant un grand nombre d'échantillons. Cela évite le calcul explicite des intégrales multidimensionnelles auxquelles on a à faire lorsque l'on applique le théorème de Bayes.

Ces algorithmes de simulation se composent de deux parties : chaînes de Markov et Monte Carlo. Essayons de comprendre ces deux termes.

Que signifie Monte Carlo ? L'intégration Monte Carlo est une technique de simulation utilisée pour calculer des intégrales de fonctions quelconques f d'une variable aléatoire X suivant une distribution $\Pr(X)$, comme $\int f(X) \Pr(X) \, dX$. On tire des valeurs $X_1, \ldots, X_k$ dans $\Pr(X)$, on applique la fonction f à ces valeurs, puis on calcule la moyenne de ces nouvelles valeurs : $\frac{1}{k}\sum_{i=1}^{k} f\left(X_i\right)$ pour approximer l'intégrale.

Comment utilise-t-on l'intégration Monte Carlo dans un contexte bayésien ? La distribution a posteriori contient toute l'information nécessaire sur le paramètre à estimer. Mais lorsqu'il y a plusieurs paramètres, on souhaite souvent résumer cette information en calculant des résumés numériques. Le résumé le plus simple est la moyenne de la distribution a posteriori, soit $E(\theta) = \int \theta \Pr(\theta \mid \text{données}) \, d\theta$, où X est ici θ et f est l'identité. Cette moyenne a posteriori peut être estimée par intégration Monte Carlo, par exemple pour la survie des ragondins :

```
# tirage de 1 000 valeurs depuis la postérieure bêta(20,39)
sample_from_posterior <- rbeta(1000, 20, 39)
# calcul de la moyenne par intégration Monte Carlo
mean(sample_from_posterior)
#> [1] 0.3381964
```

On peut vérifier que la moyenne obtenue est proche de l'espérance théorique d'une distribution bêta :

```
20/(20+39) # espérance de la loi bêta(20,39)
#> [1] 0.3389831
```

Un autre résumé numérique utile est l'intervalle de crédibilité à l'intérieur duquel se trouve le paramètre avec une certaine probabilité, généralement 0.95, soit un intervalle de crédibilité à 95 %. Déterminer les bornes d'un tel intervalle nécessite le calcul de quantiles, ce qui implique aussi des intégrales, donc un recours à l'intégration Monte Carlo. Un intervalle de crédibilité à 95 % pour la survie hivernale s'obtient avec :

```
quantile(sample_from_posterior, probs = c(2.5/100, 97.5/100))
#>      2.5%     97.5%
#> 0.2241288 0.4496781
```

Notons qu'il y a une différence entre l'intervalle de crédibilité en statistique bayésienne et l'intervalle de confiance de la statistique fréquentiste. Un intervalle de confiance à 95 % signifie que si l'on répétait l'expérience un très grand nombre de fois (équiper des ragondins de GPS et constater le nombre de survivants à l'hiver), environ 95 % des intervalles construits de cette manière contiendraient la vraie valeur θ du paramètre. Mais on ne peut pas dire que la probabilité que le paramètre soit dans un intervalle donné est de 95 %. Un intervalle de crédibilité à 95 %, en revanche, signifie qu'il y a 95 % de probabilité que le paramètre se trouve dans cet intervalle. L'interprétation de l'intervalle de crédibilité est un peu plus intuitive que celle de l'intervalle de confiance.

Maintenant, qu'est-ce qu'une chaîne de Markov ? Une chaîne de Markov est une séquence aléatoire de nombres, dans laquelle chaque nombre dépend uniquement du nombre précédent. Un exemple est la météo dans ma ville, Montpellier, dans le sud de la France, où une journée ensoleillée est très probablement suivie d'une autre journée ensoleillée, disons avec une probabilité de 0.8, et une journée pluvieuse est rarement suivie d'une autre journée pluvieuse, disons avec une probabilité de 0.1. La dynamique de cette chaîne de Markov est capturée par la matrice de transition :

	Ensoleillé demain	**Pluvieux demain**
Ensoleillé aujourd'hui	0.8	0.2
Pluvieux aujourd'hui	0.9	0.1

Les lignes indiquent la météo aujourd'hui, et les colonnes, celle de demain. Les cellules donnent la probabilité d'avoir une journée ensoleillée ou pluvieuse demain, selon le temps qu'il fait aujourd'hui (des probabilités conditionnelles, voir chapitre 1).

Sous certaines conditions, une chaîne de Markov converge vers une distribution stationnaire unique. Dans notre exemple météorologique, lançons la chaîne pour 20 étapes :

```
# matrice de transition
temps <- matrix(c(0.8, 0.2, 0.9, 0.1), nrow = 2, byrow = T)
etapes <- 20
for (i in 1:etapes){
  temps <- temps %*% temps # multiplication matricielle
}
round(temps, 2) # produit matriciel après 20 étapes
#>      [,1] [,2]
#> [1,] 0.82 0.18
#> [2,] 0.82 0.18
```

Chaque ligne de la matrice converge vers la même distribution (0.82,0.18) au fur et à mesure que le nombre d'étapes augmente. La convergence se produit quel que soit l'état de départ : on a alors une probabilité de 0.82 d'avoir du soleil et de 0.18 d'avoir de la pluie.

Revenons aux méthodes MCMC. L'idée centrale est que l'on peut construire une chaîne de Markov dont la distribution stationnaire est justement la distribution a posteriori de nos paramètres. Retenez bien cette idée, elle est centrale.

En combinant Monte Carlo et chaînes de Markov, les méthodes MCMC nous permettent de générer un échantillon de valeurs dont la distribution converge vers la distribution a posteriori (chaîne de Markov), et d'utiliser cet échantillon pour calculer des résumés numériques a posteriori (Monte Carlo), comme la moyenne ou les intervalles de crédibilité.

Il existe plusieurs manières de construire des chaînes de Markov pour l'inférence bayésienne. Vous avez peut-être entendu parler de l'algorithme de Metropolis-Hastings ou de l'échantillonneur de Gibbs. Vous pouvez consulter en ligne[6] une galerie interactive d'algorithmes MCMC. Ici, j'illustre l'algorithme de Metropolis et sa mise en œuvre pratique. Pour cela je m'inspire de l'excellent livre de Jim Albert (2009). Il n'est pas question d'être capable d'écrire un tel algorithme par soi-même, seulement d'en saisir les grandes lignes et, surtout, la notion de simulations.

Revenons à notre exemple d'estimation de la survie. Nous allons illustrer l'échantillonnage depuis la distribution a posteriori de la survie. Commençons par écrire les fonctions pour la vraisemblance, le prior et la postérieure. On se place sur l'échelle log pour manipuler des sommes et des soustractions plutôt que des produits et des ratios qui rendent les calculs numériques instables :

```
# 19 animaux retrouvés vivants sur 57 capturés, marqués et relâchés
y <- 19
n <- 57

# log-vraisemblance binomiale Bin(n = 57,p)
loglikelihood <- function(x, p){
  dbinom(x = x, size = n, prob = p, log = TRUE)
}

# densité du prior uniforme
logprior <- function(p){
  dunif(x = p, min = 0, max = 1, log = TRUE)
  # ou bien dbeta(x = p, shape1 = 0, shape2 = 1, log = TRUE)
}

# densité a posteriori (échelle logarithmique)
posterior <- function(x, p){
  loglikelihood(x, p) + logprior(p)
}
```

L'algorithme de Metropolis fonctionne comme suit.

1. On choisit une valeur initiale pour le paramètre à estimer. C'est notre valeur de départ ou point initial de la chaîne de Markov.

2. Pour décider de l'étape suivante, on propose de s'éloigner de la valeur courante du paramètre – c'est la valeur candidate. On ajoute à la valeur courante une valeur tirée d'une loi normale avec une certaine variance – c'est la loi de proposition. L'algorithme de Metropolis est un cas particulier de celui de Metropolis-Hastings avec des propositions symétriques.

3. On calcule le rapport des vraisemblances entre la position candidate et la position courante : $R = \frac{\Pr(\text{valeur candidate} \mid \text{données})}{\Pr(\text{valeur courante} \mid \text{données})}$. Pour calculer le numérateur et le dénominateur, il suffit

6. https://chi-feng.github.io/mcmc-demo/

d'appliquer le théorème de Bayes, et c'est là que la magie des méthodes MCMC opère car, comme la quantité Pr(données) apparaît au numérateur et au dénominateur, elle s'annule et plus besoin de la calculer ! On a remplacé le calcul d'une intégrale par des simulations.

4. Si la densité postérieure en la position candidate est plus grande qu'en la position courante, autrement dit si la valeur candidate est plus plausible, on l'accepte sans hésiter. Sinon, on l'accepte avec une probabilité R, et on la rejette avec une probabilité $1 - R$. Par exemple, si la valeur candidate est dix fois moins plausible, on l'accepte avec une probabilité 0.1. En pratique, on utilise un générateur uniforme entre 0 et 1 (appelons-le X), et si $X < R$, on accepte la valeur candidate, sinon on reste à la valeur courante. En pratique, on vise un taux d'acceptation entre 0.2 et 0.4, qu'on peut ajuster en calibrant la variance de la proposition, cela permet d'explorer tout le champ des possibles.

5. On répète les étapes 2 à 4 un certain nombre de fois – ce sont les itérations.

Assez de théorie, passons à l'implémentation. On commence par initialiser :

```
steps <- 100 # nombre d'étapes ou itérations de la chaîne
theta.post <- rep(NA, steps) # vecteur pour stocker les valeurs simulées
accept <- rep(NA, steps) # vecteur pour enregistrer les acceptations/rejets
set.seed(666) # pour la reproductibilité
```

Pourquoi faut-il initialiser ? Avant de lancer la chaîne de Markov, on prépare les objets dans lesquels seront stockées les valeurs simulées de notre paramètre (ici, la probabilité de survie), ainsi que l'information sur l'acceptation ou non de chaque proposition. Et set.seed(666), à quoi ça sert ? Cette commande fixe la graine du générateur de nombres aléatoires. Elle permet de garantir que les simulations soient reproductibles : en relançant le code, vous obtiendrez exactement les mêmes valeurs simulées que les miennes.

On choisit une valeur de départ :

```
inits <- 0.5 # valeur de départ choisie pour theta
theta.post[1] <- inits # enregistre valeur comme première position de la chaîne
accept[1] <- 1 # la valeur initiale est acceptée par défaut
```

Pourquoi une valeur de départ ? Une chaîne de Markov doit bien commencer quelque part : ici, on choisit arbitrairement 0.5 comme valeur initiale de la probabilité de survie. La seule contrainte est que cette valeur soit compatible avec le prior, on ne va pas prendre une survie négative ou de 15. On place cette valeur dans le premier élément du vecteur theta.post, et on indique dans accept[1] <- 1 que cette première valeur est acceptée par construction, puisque c'est notre point de départ.

Puis, on écrit une fonction pour proposer une valeur candidate à partir de la valeur courante :

```
move <- function(x, away = 1){
  logitx <- log(x / (1 - x)) # logit : transforme x de (0,1) vers (-∞,+∞)
  logit_candidate <- logitx + rnorm(1, 0, away) # ajoute bruit normal centré,
de variance contrôlée par away
  candidate <- plogis(logit_candidate) # logit réciproque (logit^-1) :
retourne une valeur entre 0 et 1
  return(candidate) # retourne la valeur proposée
}
```

Cette fonction introduit une proposition aléatoire autour de la valeur courante. On travaille sur l'échelle logit pour s'assurer que la proposition finale (candidate) reste toujours dans l'intervalle (0,1) (voir aussi le chapitre 6). Le paramètre away contrôle la dispersion des propositions : plus il est grand, plus la chaîne pourra faire de grands sauts ; plus il est petit, plus les propositions seront proches de la valeur actuelle.

Ensuite, on applique les étapes 2 à 4 de l'algorithme dans une boucle (c'est l'étape 5, la répétition des itérations) :

```
for (t in 2:steps){ # pour chaque itération, à partir de la 2e
  # Étape 2 : proposer une nouvelle valeur pour theta
  theta_star <- move(theta.post[t-1])  # valeur candidate tirée à partir de la
valeur précédente

  # Étape 3 : calculer le rapport des densités postérieures
(échelle log)
  # densité a posteriori à la valeur candidate
  pstar <- posterior(y, p = theta_star)
  # densité a posteriori à la valeur courante
  pprev <- posterior(y, p = theta.post[t-1])
  logR <- pstar - pprev # différence sur l'échelle log
  R <- exp(logR) # on revient à l'échelle naturelle
(rapport des densités)

  # Étape 4 : décider si on accepte ou rejette la proposition
  X <- runif(1, 0, 1) # tirage aléatoire entre 0 et 1
  if (X < R){ # si la proposition est plus plausible
    theta.post[t] <- theta_star # on accepte et on stocke la valeur candidate
    accept[t] <- 1 # on note que la proposition a été acceptée
  } else {
    theta.post[t] <- theta.post[t-1] # sinon on reste sur la valeur précédente
    accept[t] <- 0 # on note le refus
  }
}
```

Cette boucle construit itérativement la chaîne de Markov. La probabilité d'accepter une valeur moins plausible est proportionnelle à son rapport de vraisemblance. Le vecteur accept permet ensuite de diagnostiquer la fréquence d'acceptation, utile pour calibrer la chaîne.

Jetons un coup d'œil aux premières et dernières valeurs simulées :

```
head(theta.post)
#> [1] 0.5000000 0.5000000 0.3021903 0.3021903 0.1853669 0.1853669
tail(theta.post)
#> [1] 0.4076667 0.4076667 0.4076667 0.4076667 0.2914464 0.2914464
```

On peut maintenant visualiser l'évolution des valeurs de la chaîne grâce à un *trace plot* ou « graphique de la trace » (on va garder l'expression anglaise), c'est-à-dire une courbe qui montre les valeurs simulées de θ au fil des itérations, c'est la **figure 2.3**.

Que nous apprend ce *trace plot* ? L'axe horizontal représente les itérations (ou temps dans la chaîne de Markov). L'axe vertical montre les valeurs simulées de la probabilité de survie à chaque étape. Dans la figure, on observe que la chaîne reste parfois pendant plusieurs itérations consécutives à la même valeur. Cela se produit lorsque la valeur candidate proposée par l'algorithme est rejetée – la chaîne conserve alors la valeur précédente (ou courante plus précisément). À d'autres moments, on voit des sauts vers de nouvelles valeurs, qui correspondent aux propositions acceptées.

On peut ensuite encapsuler l'algorithme dans une fonction réutilisable, ce qui permet de lancer facilement plusieurs chaînes :

```
metropolis <- function(steps = 100, inits = 0.5, away = 1){

  theta.post <- rep(NA, steps) # crée un vecteur pour stocker les échantillons
  theta.post[1] <- inits # initialise avec la valeur de départ
  for (t in 2:steps){ # boucle sur les étapes (à partir de la 2e)
```

```
    theta_star <- move(theta.post[t-1], away) # proposition nouvelle valeur

    # calcule le log-ratio densité a posteriori entre candidat et position
courante
    logR <- posterior(y, theta_star) -
            posterior(y, theta.post[t-1])
    R <- exp(logR) # passage à l'échelle normale (non log)
    X <- runif(1, 0, 1) # tirage d'un nombre aléatoire uniforme
    theta.post[t] <- ifelse(X < R, # si le tirage < probabilité d'acceptation
                      theta_star, # on accepte la valeur proposée
                      theta.post[t-1]) # sinon on conserve la précédente
  }

  return(theta.post) # on retourne l'échantillon simulé
}
```

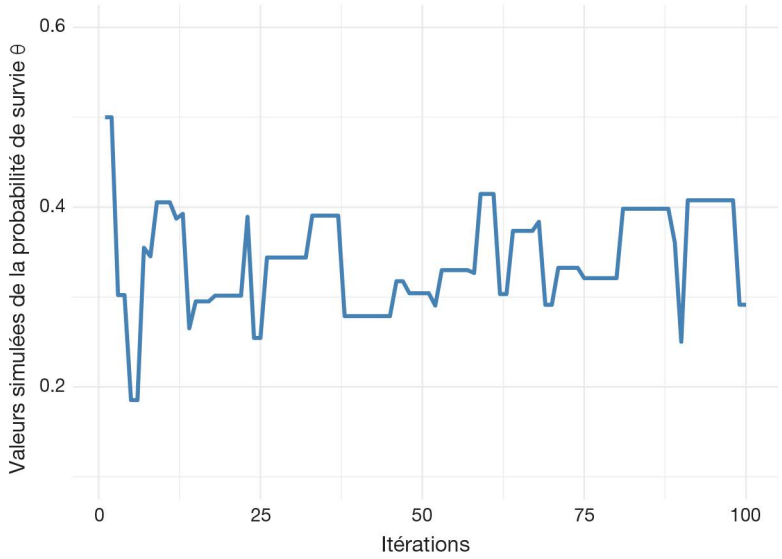

Figure 2.3. *Trace plot* des valeurs simulées de la probabilité de survie au fil des itérations.

On peut maintenant utiliser la fonction metropolis() pour lancer une autre chaîne, cette fois-ci en partant de 0.2 :

```
theta.post2 <- metropolis(steps = 100, inits = 0.2) # départ à 0.2
```

Notez que l'on parle souvent de « lancer plusieurs chaînes » MCMC afin de diagnostiquer la convergence. Il s'agit en réalité de réalisations indépendantes de la même chaîne de Markov, comme si on lançait plusieurs fois une pièce avec une distribution un peu plus compliquée qu'une Bernoulli.

On trace ensuite les deux chaînes ensemble comme dans la **figure 2.4**.

Notez que nous n'obtenons pas exactement les mêmes résultats, car l'algorithme est stochastique. On observe l'évolution parallèle de deux chaînes lancées avec des valeurs initiales différentes. Si les deux chaînes se rejoignent rapidement et oscillent autour des mêmes valeurs, cela indique une bonne convergence vers la distribution stationnaire souhaitée. C'est une étape clé des diagnostics de convergence MCMC que l'on verra dans la suite de ce chapitre. Pour observer la convergence sur une plus longue période, on lance une chaîne avec 1 000 itérations. Cela permet d'obtenir un *trace plot* plus « lisse » qui montre la stabilité de la chaîne, comme dans la **figure 2.5**.

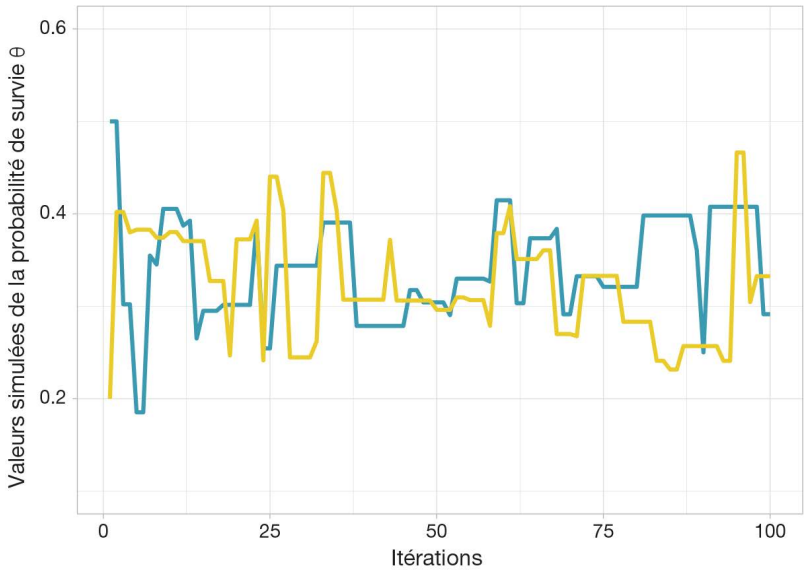

Figure 2.4. *Trace plot* des valeurs simulées de la probabilité de survie au fil des itérations. Deux chaînes ont été lancées avec des valeurs initiales différentes, 0.5 en bleu et 0.2 en jaune.

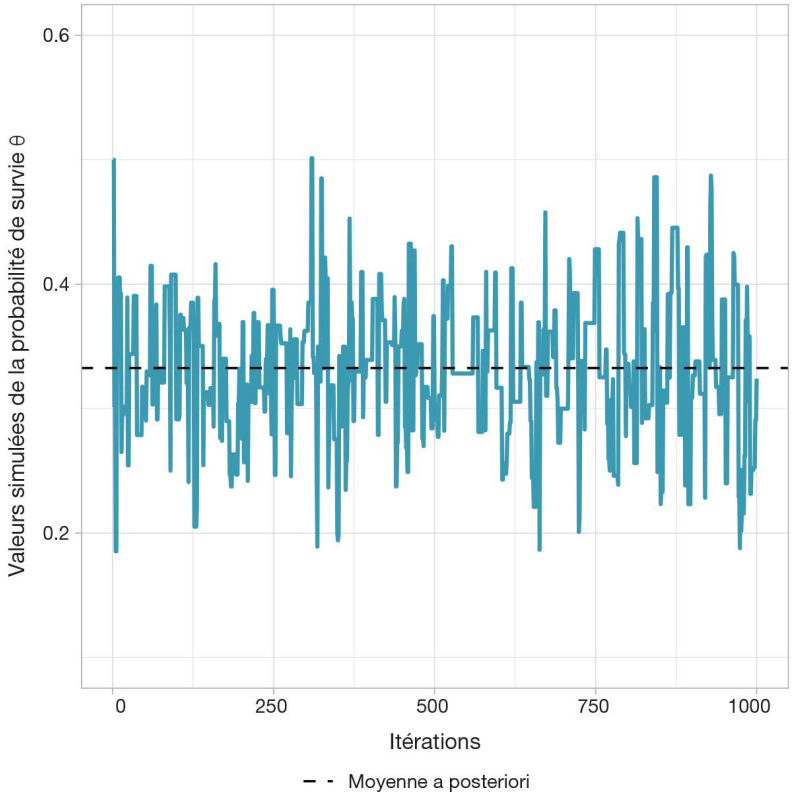

Figure 2.5. *Trace plot* des valeurs simulées de la probabilité de survie au fil des 1 000 itérations.

Avec un grand nombre d'itérations, chaque chaîne devrait se stabiliser autour de sa distribution stationnaire. Visuellement, on recherche une zone dense, homogène et bien explorée, ressemblant à une pelouse bien tondue (c'est une image).

Une fois la distribution stationnaire atteinte, vous pouvez considérer les valeurs simulées de la chaîne de Markov comme un échantillon de la distribution a posteriori et obtenir des résumés numériques des paramètres (moyenne a posteriori, intervalle de crédibilité).

Quand peut-on dire qu'on a atteint cette distribution stationnaire ? Une fois qu'on a la convergence, combien de simulations faut-il encore faire pour obtenir une bonne approximation de la distribution a posteriori de nos paramètres ? Je réponds à ces questions dans la section suivante.

2.3. ÉVALUER LA CONVERGENCE

Lorsqu'on applique une méthode MCMC, il faut déterminer combien de temps il faut à la chaîne de Markov pour converger vers la distribution cible, et combien d'itérations supplémentaires sont nécessaires après la convergence pour obtenir des estimations Monte Carlo fiables des résumés numériques (moyennes a posteriori, intervalles de crédibilité).

2.3.1. Burn-in

En pratique, on ignore les premières valeurs de la chaîne de Markov et on n'utilise que les valeurs simulées après convergence. Les observations initiales que l'on écarte sont généralement appelées la période de burn-in (ou préchauffage).

La méthode la plus simple pour déterminer la durée de la période de burn-in est d'inspecter les *trace plots*. Reprenons notre exemple et observons dans la **figure 2.6** un *trace plot* pour une chaîne démarrant à la valeur 0.99.

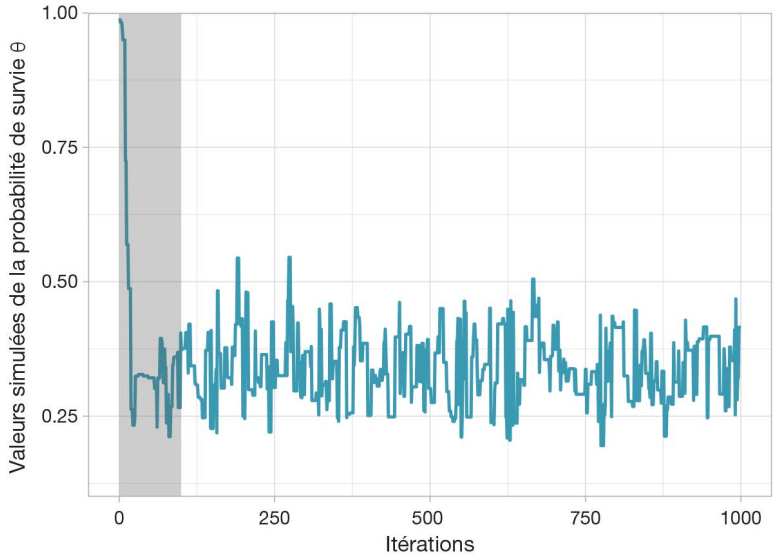

Figure 2.6. *Trace plot* pour une chaîne démarrant à 0.99. La zone ombrée illustre une période de burn-in possible.

La chaîne démarre à 0.99 et se stabilise rapidement, les valeurs oscillant autour de 0.3 à partir de la 100^{e} itération. On peut choisir la zone ombrée comme période de burn-in et éliminer les 100 premières valeurs. Par prudence, on pourrait utiliser 250, voire 500, itérations comme burn-in, si ça ne coûte pas trop cher en temps de calcul évidemment.

Examiner un *trace plot* d'une seule chaîne est utile, mais on lance généralement plusieurs chaînes avec des valeurs initiales différentes pour vérifier que toutes atteignent la même distribution stationnaire. Cette approche est formalisée par la statistique de Brooks-Gelman-Rubin (BGR), notée $\widehat{R}$, qui mesure le ratio entre la variabilité totale (entre les chaînes, plus à l'intérieur de chaque chaîne) et la variabilité intrachaîne. Elle est proche du test F dans une analyse de variance (ici à un facteur dont les modalités seraient les chaînes). Une valeur inférieure à 1.1 indique une convergence probable.

Revenons à notre exemple : nous exécutons deux chaînes de Markov avec des valeurs initiales de 0.2 et 0.8, en faisant varier le nombre d'itérations de 100 à 1 000, toutes les 50 itérations, et nous calculons la statistique BGR en utilisant la moitié des itérations comme période de burn-in (**figure 2.7**).

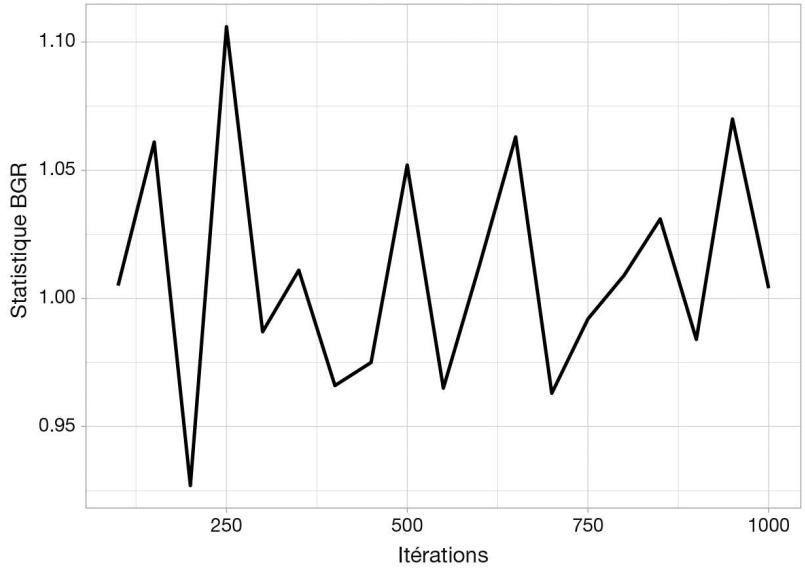

Figure 2.7. Valeur de la statistique de Brooks-Gelman-Rubin (BGR) en fonction du nombre d'itérations. Une valeur proche de 1 suggère la convergence.

Nous obtenons une valeur de la statistique BGR proche de 1 dès 300 itérations, ce qui suggère qu'avec un burn-in de 300 itérations, rien n'indique un problème de convergence.

Il est important de garder à l'esprit qu'une valeur proche de 1 pour la statistique BGR constitue une condition nécessaire, mais non suffisante, à la convergence. En d'autres termes, ce diagnostic ne permet pas d'affirmer avec certitude que la chaîne a convergé, mais simplement qu'on ne détecte pas de signe évident qu'elle ne l'a pas fait. Mon conseil : prenez toujours le temps de jeter un coup d'œil aux *trace plots*.

2.3.2. Longueur de chaîne

Quelle longueur de chaîne est nécessaire pour obtenir des estimations fiables des paramètres ? Il faut garder à l'esprit que les étapes successives d'une chaîne de Markov ne sont pas indépendantes. C'est ce que l'on appelle l'autocorrélation. Idéalement, on cherche à minimiser cette autocorrélation.

Ici encore, les *trace plots* permettent de diagnostiquer des problèmes d'autocorrélation. Revenons à l'exemple de survie. La **figure 2.8** montre les *trace plots* (3 000 itérations) pour différentes valeurs de l'écart-type (paramètre away) de la loi normale de proposition utilisée pour générer les valeurs candidates.

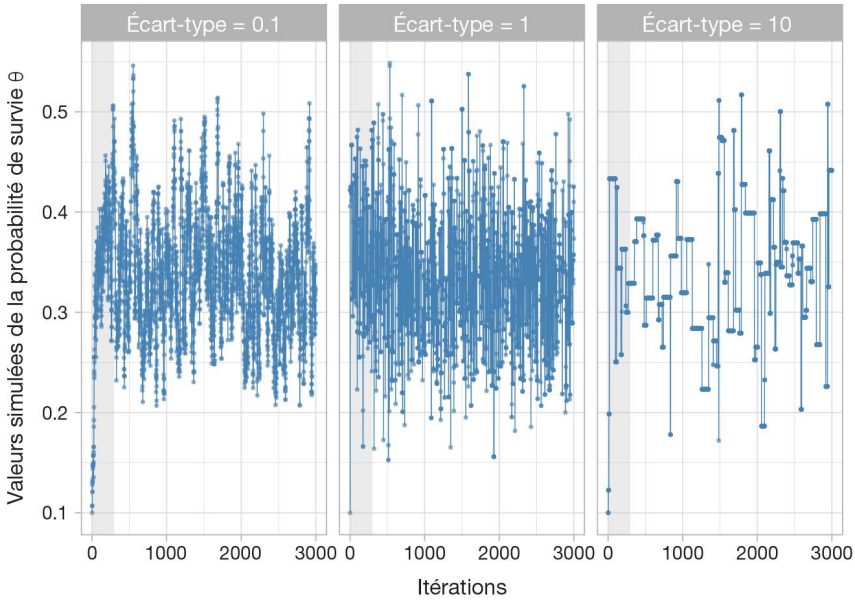

Figure 2.8. *Trace plots* pour différentes valeurs de l'écart-type de la proposition (away). Un bon mixing est observé avec away = 1. La zone grise ombrée correspond à un burn-in de 300 itérations.

Les petits et grands déplacements visibles dans les panneaux de gauche et de droite entraînent une forte corrélation entre les observations successives de la chaîne de Markov, tandis qu'un écart-type égal à 1 (au centre) permet une exploration efficace de l'espace des paramètres. Ce mouvement dans l'espace des paramètres est appelé « mixing ». Le mixing est considéré comme mauvais lorsque la chaîne fait de trop petits ou de trop grands sauts, et bon dans le cas contraire.

En complément des *trace plots*, les graphiques de la fonction d'autocorrélation (ACF) offrent un moyen pratique de visualiser la force de l'autocorrélation dans un échantillon donné. Les ACF montrent la corrélation entre les valeurs échantillonnées successivement, séparées par un nombre croissant d'itérations, appelé lag (décalage). On obtient dans la **figure 2.9** ces graphiques de fonction d'autocorrélation pour différentes valeurs de l'écart-type de la distribution de proposition grâce à la fonction forecast::ggAcf().

Dans les panneaux de gauche et de droite, l'autocorrélation est forte, diminue lentement avec le lag, et le mixing est mauvais. Dans le panneau central, l'autocorrélation est faible, diminue rapidement avec le lag, et le mixing est bon.

L'autocorrélation n'est pas forcément un gros problème. Des observations fortement corrélées nécessitent simplement un plus grand nombre d'échantillons, et donc des simulations plus longues. Mais combien d'itérations faut-il exactement ? La taille effective de l'échantillon (n.eff) mesure la longueur utile de la chaîne en tenant compte de l'autocorrélation. Il est recommandé de vérifier la n.eff pour chaque paramètre d'intérêt, ainsi que pour toute combinaison pertinente de paramètres. En général, on considère qu'il faut au moins n.eff ≥ 400 observations indépendantes pour obtenir des estimations Monte Carlo fiables des paramètres du modèle. Dans l'exemple de la survie animale, n.eff peut être calculée avec la fonction effectiveSize() du package coda :

```
# Générer les chaînes pour trois valeurs d'écart-type
d <- tibble(away = c(0.1, 1, 10)) %>%
     mutate(accepted_traj = map(away,
                                metropolis,
```

```
                                steps = n_steps,
                                inits = 0.1)) %>%
    unnest(accepted_traj) %>%
    mutate(proposal_sd = str_c("Écart-type = ", away),
           iter = rep(1:n_steps, times = 3))

# Calculer la taille effective de l'échantillon
neff1 <- effectiveSize(d$accepted_traj[d$proposal_sd=="Écart-type = 0.1"]
[-c(1:300)])
neff2 <- effectiveSize(d$accepted_traj[d$proposal_sd=="Écart-type = 1"]
[-c(1:300)])
neff3 <- effectiveSize(d$accepted_traj[d$proposal_sd=="Écart-type = 10"]
[-c(1:300)])
tibble("Écart-type" = c(0.1, 1, 10),
       "n.eff" = round(c(neff1, neff2, neff3)))
#> # A tibble : 3 × 2
#>   `Écart-type` n.eff
#>          <dbl> <dbl>
#> 1          0.1    73
#> 2          1     621
#> 3         10      52
```

Comme on pouvait s'y attendre, la valeur de n.eff est inférieure au nombre total d'itérations MCMC (3 000) en raison de l'autocorrélation. Ce n'est que lorsque l'écart-type de la distribution de proposition est égal à 1 que le mixing est bon (n.eff ≥ 400), ce qui permet d'obtenir une taille d'échantillon effective satisfaisante.

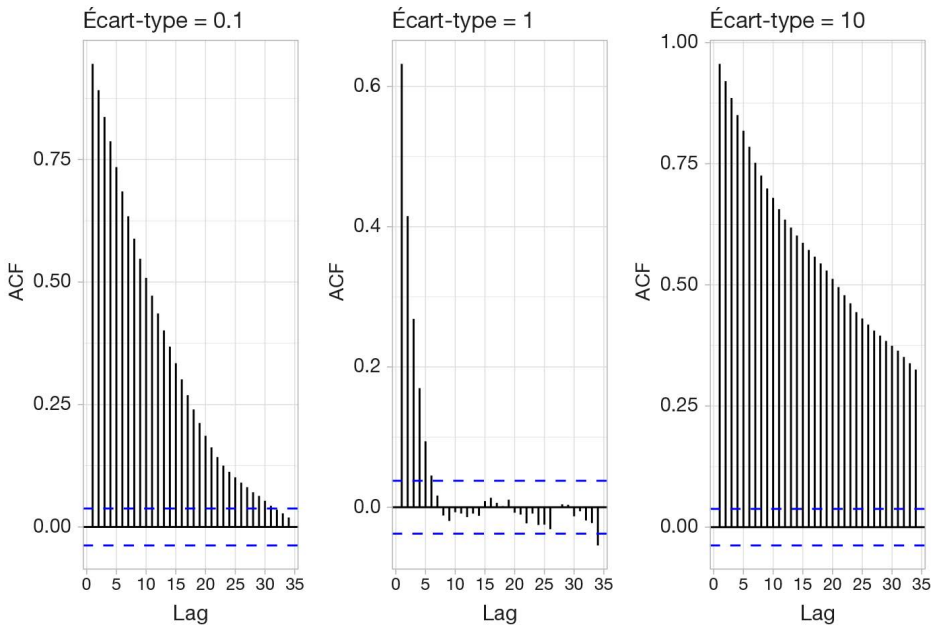

Figure 2.9. Fonctions d'autocorrélation (ACF) pour différentes valeurs de l'écart-type de la proposition. Une faible autocorrélation est un signe de bon mixing. Un burn-in de 300 itérations est appliqué.

2.3.3. Et si vous avez des problèmes de convergence ?

Lorsque vous diagnostiquez la convergence d'une chaîne MCMC, vous rencontrerez (très) souvent des difficultés. Cette section propose quelques conseils pratiques qui, je l'espère, vous seront utiles.

Lorsque le mixing est mauvais et que la taille effective de l'échantillon est faible, il peut suffire d'augmenter la période de burn-in et/ou d'augmenter le nombre de simulations. L'utilisation de priors plus informatifs peut également faciliter la convergence des chaînes de Markov, en aidant l'algorithme MCMC à explorer plus efficacement l'espace des paramètres (chapitre 4). Dans le même esprit, choisir de meilleures valeurs initiales pour démarrer la chaîne peut aussi améliorer les choses. Une stratégie utile consiste à utiliser les estimations d'un modèle plus simple, pour lequel vos chaînes MCMC convergent déjà.

Si les problèmes de convergence persistent, il y a souvent un problème avec le modèle lui-même. Un bug dans le code ? Une faute de frappe ? Une erreur dans les équations ? Comme souvent en programmation, le meilleur moyen d'identifier le problème est de réduire la complexité du modèle et de repartir d'un modèle plus simple, jusqu'à ce que vous trouviez ce qui ne va pas.

Un autre conseil est de considérer votre modèle avant tout comme un générateur de données. Simulez des données à partir de ce modèle, en utilisant des valeurs réalistes pour les paramètres, puis tentez de retrouver ces paramètres en ajustant le modèle aux données simulées. Cette approche vous aidera à mieux comprendre comment le modèle fonctionne, ce qu'il ne fait pas, et les types de données nécessaires pour obtenir des estimations de paramètres fiables. On reviendra sur cette technique dans les chapitres 5 et 6.

À RETENIR

- L'idée des méthodes MCMC est de simuler des valeurs à partir d'une chaîne de Markov dont la distribution stationnaire est précisément la distribution a posteriori des paramètres que l'on cherche à estimer.
- En pratique, on lance plusieurs chaînes de Markov en partant de valeurs initiales dispersées.
- On écarte les premières itérations (phase de préchauffage ou burn-in) et on considère que la convergence est atteinte lorsque toutes les chaînes convergent vers le même régime.
- À partir de là, on fait tourner les chaînes suffisamment longtemps, puis on calcule des estimations Monte Carlo de résumés numériques (par exemple, les moyennes a posteriori ou les intervalles de crédibilité) des paramètres.
- Évidemment, on n'a pas envie de construire et d'implémenter à la main les méthodes MCMC à chaque nouvelle analyse, et on verra dans le chapitre 3 comment se faciliter la tâche.

3. Mise en œuvre pratique

Dans ce chapitre, nous découvrirons brms, un outil très pratique pour faire de la statistique bayésienne sans trop d'efforts. Dans la version enrichie en ligne[7], vous trouverez aussi une introduction à NIMBLE. brms et NIMBLE sont deux packages R qui implémentent pour vous les algorithmes MCMC. Concrètement, il vous suffit de spécifier une vraisemblance et des priors pour que le théorème de Bayes s'applique automatiquement. Grâce à une syntaxe proche de celle de R, les deux packages rendent cette étape relativement simple, même pour des modèles complexes.

3.1. LA SYNTAXE DU PACKAGE BRMS

brms signifie *bayesian regression models using stan*. Ce package[8] permet de formuler et d'estimer des modèles de régression (voir la section suivante et les chapitres 5 et 6) de manière intuitive, grâce à une syntaxe proche de celle du package lme4 (la référence R pour les modèles mixtes), tout en s'appuyant sur Stan (un logiciel de référence en statistique bayésienne).

Pour utiliser brms, on commence par préparer les données :

```
dat <- data.frame(y = 19, n = 57)
```

Sans oublier de charger brms :

```
library(brms)
```

La vraisemblance est binomiale dans notre exemple fil rouge. Dans brms, on peut exprimer cela simplement :

```
bayes.brms <- brm(
  y | trials(n) ~ 1, # le nombre de succès est fonction d'un intercept
  family = binomial("logit"), # famille binomiale avec fonction de lien logit
  data = dat, # données utilisées
  chains = 3, # nombre de chaînes MCMC
  iter = 2000, # nombre total d'itérations par chaîne
  warmup = 300, # nombre d'itérations burn-in
  thin = 1 # pas de sous-échantillonnage (chaque itération est conservée)
)
```

La syntaxe est relativement simple, mais nécessite quelques explications. L'argument y | trials(n) ~ 1 permet de spécifier un modèle dans lequel on a y succès parmi n essais, et on estime une ordonnée à l'origine (ou intercept) seulement, dénoté par le 1 après le symbole ~. Pourquoi un intercept ici ? Pourquoi pas directement la survie θ ? Parce qu'on utilise family = binomial("logit") à la ligne d'après pour spécifier à brms que la variable réponse suit une loi binomiale. Autrement dit, on a un modèle linéaire généralisé (voir chapitre 6) avec logit(θ) = β et on estime β l'intercept. Les arguments iter = 2000, warmup = 300 et chains = 3 stipulent à brms d'utiliser 300 itérations pour l'adaptation (burn-in), et les 1 700 suivantes pour l'inférence, avec 3 chaînes.

Jetons un coup d'œil aux résultats :

```
summary(bayes.brms)
#>  Family: binomial
#>   Links: mu = logit
#> Formula : y | trials(n) ~ 1
```

7. https://oliviergimenez.github.io/statistique-bayes/logiciels.html
8. Le package est en constant développement, voir https://paul-buerkner.github.io/brms/. Vous pouvez obtenir de l'aide *via* https://discourse.mc-stan.org/

```
#>    Data: dat (Number of observations: 1)
#>   Draws: 3 chains, each with iter = 2000; warmup = 300; thin = 1;
#>          total post-warmup draws = 5100
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept    -0.69      0.28    -1.25    -0.15 1.00     1551     1997
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Cette commande affiche un tableau récapitulatif des estimations postérieures pour chaque paramètre du modèle. On y trouve :

– Estimate, la moyenne a posteriori ;

– Est.Error, l'écart-type de la distribution a posteriori ;

– l–95% CI et u–95% CI, les bornes de l'intervalle de crédibilité à 95 % ;

– le diagnostic de convergence Rhat ;

– Bulk_ESS, la taille effective de l'échantillon (Tail_ESS est une autre mesure de la taille effective de l'échantillon que l'on n'utilisera pas ici).

La moyenne a posteriori vaut –0.69, bien loin de la proportion de ragondins qui ont survécu à l'hiver (19/57 ≈ 0.33). Comme toujours dans R et l'implémentation des modèles linéaires généralisés (voir chapitre 6), les estimations des paramètres sont données sur l'échelle de la fonction de lien. Ici, l'intercept estimé est exprimé sur l'échelle logit. Pour le convertir en probabilité de survie (entre 0 et 1), on extrait d'abord les valeurs générées dans la distribution a posteriori de l'intercept β avec la fonction brms::as_draws_matrix() :

```
draws_fit <- as_draws_matrix(bayes.brms)
```

Puis, on applique la fonction logistique réciproque plogis() sur chacune de ces valeurs pour obtenir tout un tas de valeurs simulées dans la distribution a posteriori de la survie θ :

```
beta <- draws_fit[,'Intercept'] # sélectionne la colonne intercept
theta <- plogis(beta)  # conversion logit -> [0,1]
```

On obtient ainsi une estimation directe de la moyenne a posteriori de la probabilité de survie, accompagnée de son intervalle de crédibilité à 95 % :

```
mean(theta)
#> [1] 0.3378754
quantile(theta, probas = c(2.5,97.5)/100)
#>        0%       25%       50%       75%      100%
#> 0.1478069 0.2939825 0.3379197 0.3785993 0.5916079
```

Ou plus directement avec la fonction posterior::summarise_draws() :

```
summarise_draws(theta)
#> # A tibble : 1 × 10
#>   variable   mean median     sd    mad    q5   q95  rhat ess_bulk ess_tail
#>   <chr>     <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
#> 1 Intercept 0.338  0.338 0.0624 0.0628 0.238 0.441  1.00    1551.    1997.
```

3.2. VISUALISATION

Pour visualiser la distribution a posteriori de la probabilité de survie (**figure 3.1**), il suffit d'utiliser :

```
draws_fit %>%
  ggplot(aes(x = theta)) +
  geom_histogram(color = "white", fill = "steelblue", bins = 30) +
  labs(x = "Probabilité de survie", y = "Fréquence")
```

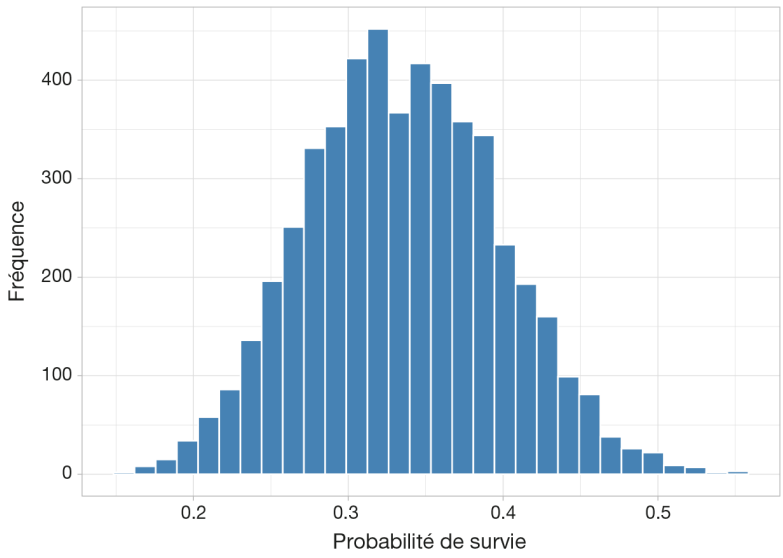

Figure 3.1. Histogramme de la distribution a posteriori de la probabilité de survie (θ).

Dans brms, on peut évaluer la convergence des chaînes MCMC (**figure 3.2**).

```
plot(bayes.brms)
```

Ce graphique affiche les *trace plots* (à droite) ainsi que les densités a posteriori (à gauche). Au passage, pour déterminer la longueur de la période de préchauffage ou burn-in, il suffit de faire tourner brms avec warmup = 0 pour quelques centaines ou milliers d'itérations et d'examiner la trace du paramètre pour décider du nombre d'itérations à utiliser pour atteindre la convergence.

Un avantage majeur des méthodes MCMC est qu'elles permettent d'obtenir la distribution a posteriori de n'importe quelle fonction des paramètres en appliquant cette fonction aux valeurs tirées dans les distributions a posteriori de ces paramètres. À noter qu'ici on estime l'intercept β et on a donc déjà utilisé cette idée pour obtenir la distribution a posteriori de la probabilité de survie en appliquant la fonction logit réciproque. Comme autre exemple, disons que j'aimerais calculer l'espérance de vie des ragondins, celle-ci étant donnée par $\lambda = -1/\log(\theta)$:

```
beta <- draws_fit[,'Intercept'] # sélectionne la colonne intercept
theta <- plogis(beta)  # conversion logit -> [0,1]
lambda <- -1 / log(theta) # transforme survie en espérance de vie
summarize_draws(lambda) # résumé des tirages : moyenne, médiane, intervalles
#> # A tibble : 1 × 10
#>   variable   mean median    sd   mad    q5   q95  rhat ess_bulk ess_tail
#>   <chr>     <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
#> 1 Intercept 0.934  0.922 0.165 0.157 0.697  1.22  1.00  1551.  1997.
```

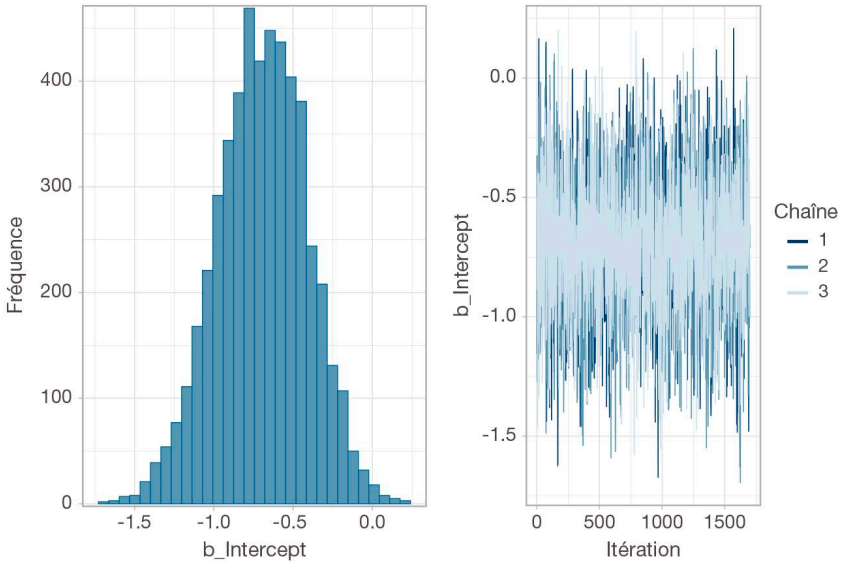

Figure 3.2. Histogramme de la distribution a posteriori et *trace plot* de la probabilité de survie sur l'échelle logit (b). Dans l'histogramme, l'axe des abscisses représente les valeurs possibles de l'intercept (échelle logit) et l'axe des ordonnées la fréquence des valeurs simulées. Dans le *trace plot*, l'axe des abscisses correspond au numéro de l'itération MCMC et l'axe des ordonnées aux valeurs simulées de l'intercept (échelle logit).

L'espérance de vie est d'un an approximativement. On peut également visualiser la distribution a posteriori de l'espérance de vie (**figure 3.3**).

```
lambda %>%
  as_tibble() %>%
  ggplot() +
  geom_histogram(aes(x = Intercept), color = "white") +
  labs(x = "Espérance de vie")
```

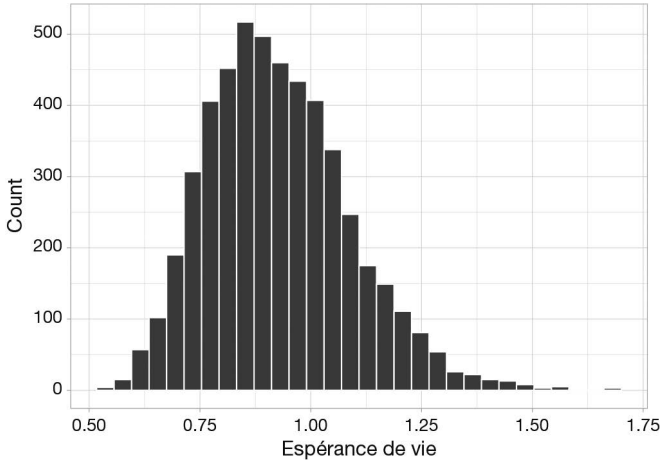

Figure 3.3. Histogramme de la distribution a posteriori de l'espérance de vie. L'axe des abscisses représente les différentes valeurs possibles de l'espérance de vie. L'axe vertical indique le nombre de tirages simulés (« count ») pour chaque valeur.

3.3. LES PRIORS

Il y a tout un tas de paramètres qui sont fixés par défaut dans brms, il est important d'en être conscient. Ça concerne les priors en particulier. Dans brms, les priors par défaut sont souvent non informatifs ou faiblement informatifs, mais il est toujours bon de les examiner explicitement. La commande suivante permet d'afficher le résumé des priors utilisés dans un modèle déjà ajusté :

```
prior_summary(bayes.brms)
#> Intercept ~ student_t(3, 0, 2.5)
```

Le package brms utilise comme a priori faiblement informatif une loi de Student à 3 degrés de liberté, centrée en 0, avec un écart-type de 2.5. Les 3 degrés de liberté donnent une distribution avec des queues plus épaisses qu'une normale, ce qui donne une certaine robustesse aux valeurs extrêmes. Le centre en 0 traduit une absence d'a priori fort sur la valeur de l'intercept. La largeur 2.5 autorise une variation raisonnablement large de l'intercept sans être complètement non informatif.

Dans certains cas, il est pertinent de définir soi-même un prior, par exemple pour refléter des connaissances issues de la littérature ou pour contraindre davantage l'estimation (prior informatif). Ici, on propose un prior normal centré en 0 avec un écart-type de 1.5 sur l'intercept, on en reparlera au chapitre 4 :

```
nlprior <- prior(normal(0, 1.5), class = "Intercept")
```

On peut ensuite l'utiliser dans la spécification du modèle :

```
bayes.brms <- brm(y | trials(n) ~ 1,
                  family = binomial("logit"),
                  data = dat,
                  prior = nlprior, # nos propres priors
                  chains = 3,
                  iter = 2000,
                  warmup = 300,
                  thin = 1)
```

Vous pouvez vérifier que les résultats sont proches de ceux obtenus avec le prior par défaut :

```
summary(bayes.brms)
#>  Family: binomial
#>   Links: mu = logit
#> Formula : y | trials(n) ~ 1
#>    Data: dat (Number of observations: 1)
#>   Draws: 3 chains, each with iter = 2000; warmup = 300; thin = 1;
#>          total post-warmup draws = 5100
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept    -0.68      0.28    -1.24    -0.15 1.00     1932     2284
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

À RETENIR

- brms permet de tirer parti des méthodes MCMC sans avoir à écrire le modèle soi-même (la vraisemblance en particulier).
- Sa syntaxe est simple et proche de celle de lme4, ce qui le rend particulièrement adapté pour les modèles linéaires généralisés (mixtes ou pas ; voir chapitre 6).
- En contrepartie, brms repose sur des composants préprogrammés (familles de modèles, etc.), et il est important d'être attentif aux choix faits par défaut, notamment en ce qui concerne les distributions a priori.
- Ce chapitre propose ainsi une première approche concrète de l'implémentation des modèles bayésiens, avant d'aborder des modèles plus riches, comme les modèles mixtes.

4. Les distributions a priori

Dans ce chapitre, nous allons explorer un aspect fondamental de la statistique bayésienne : le rôle des distributions a priori ou priors. Nous verrons comment ces priors interagissent avec les données *via* le théorème de Bayes pour produire la distribution a posteriori, et comment cette influence varie selon la quantité d'information apportée par les données. Nous apprendrons également à intégrer des informations extérieures pertinentes issues d'expertises ou d'études antérieures, et à évaluer de manière critique nos choix a priori à l'aide de simulations.

4.1. LE RÔLE DU PRIOR

En statistique bayésienne, le prior joue un rôle essentiel : il traduit nos connaissances, nos incertitudes ou, au contraire, notre absence d'information sur les paramètres d'un modèle. Bien choisir ses priors est donc une étape clé de toute analyse bayésienne. Pourquoi utiliser un prior ?

– Pour intégrer les connaissances existantes : on dispose souvent d'informations issues d'études antérieures, de méta-analyses ou d'avis d'experts. Le prior permet de formaliser et d'intégrer cette connaissance préalable, plutôt que de l'ignorer et de faire comme si l'on ne partait de rien. Nous verrons un exemple dans la section 4.3.

– Pour faire face à un manque de données : lorsque les données sont rares ou peu informatives, les méthodes fréquentistes peuvent échouer à estimer correctement certains paramètres (estimation aux bornes pour une probabilité, ou variance nulle d'un effet aléatoire). Dans ces situations, un prior bien choisi peut aider à stabiliser l'inférence, en apportant une information complémentaire.

– Pour encadrer les modèles complexes : dans les modèles mixtes, ou en présence de paramètres difficilement estimables, les priors permettent de borner l'espace des solutions à des valeurs plausibles et d'éviter des estimations aberrantes. Par exemple, dans un modèle mixte (voir chapitre 6) où l'on estime la variance entre groupes ou modalités d'un effet, l'absence de prior peut entraîner des valeurs irréalistes ou des instabilités numériques. Un prior faiblement informatif peut aider dans cette situation.

– Pour prévenir le surajustement : dans les modèles comportant de nombreuses variables explicatives, les priors jouent un rôle de régularisation en pénalisant les effets peu importants. Par exemple, dans une régression incluant de nombreuses covariables, un prior du type $N(0,1.5^2)$ empêche le modèle d'attribuer des effets trop forts à des variables peu informatives, réduisant ainsi le risque de surajustement.

Le choix d'un prior dépend directement du contexte et de la question scientifique.

– Un prior non informatif vise à exprimer un manque de connaissance : il est souvent utilisé lorsqu'on ne veut pas introduire d'hypothèses fortes. En pratique, cela se traduit par des distributions larges ou uniformes. Mais attention : même un prior apparemment vague peut être informatif une fois transformé sur l'échelle du modèle, comme on le verra dans la section 4.4.

– Un prior informatif reflète une connaissance crédible et externe au jeu de données analysé : il peut provenir d'une synthèse bibliographique, d'une expérience passée ou de l'avis d'un spécialiste. Il a pour avantage de réduire l'incertitude sur les paramètres, surtout avec peu de données. On verra un exemple dans la section 4.3.

– Un prior faiblement informatif (ou *weakly informative*) est un peu un compromis entre les priors non informatifs et informatifs. L'idée est d'exclure des valeurs manifestement aberrantes ou incompatibles avec ce que l'on sait du phénomène étudié, tout en laissant suffisamment de liberté au modèle pour apprendre des données. Ce type de prior est utilisé notamment dans brms. On verra un exemple dans le chapitre 6.

En pratique, une stratégie prudente consiste à commencer avec un prior faiblement informatif, comme une loi normale centrée avec une variance modérée, puis à tester des options plus

informatives (ou plus vagues) pour examiner l'impact sur les résultats a posteriori. C'est l'idée de l'analyse de sensibilité qu'on développe dans la section suivante.

4.2. SENSIBILITÉ AU PRIOR

Revenons à notre exemple fil rouge sur la survie des ragondins. Examinons comment différents choix de priors influencent la distribution a posteriori de cette probabilité de survie. Dans la **figure 4.1**, on a trois priors de plus en plus informatifs (en colonnes) et deux tailles d'échantillon (en lignes).

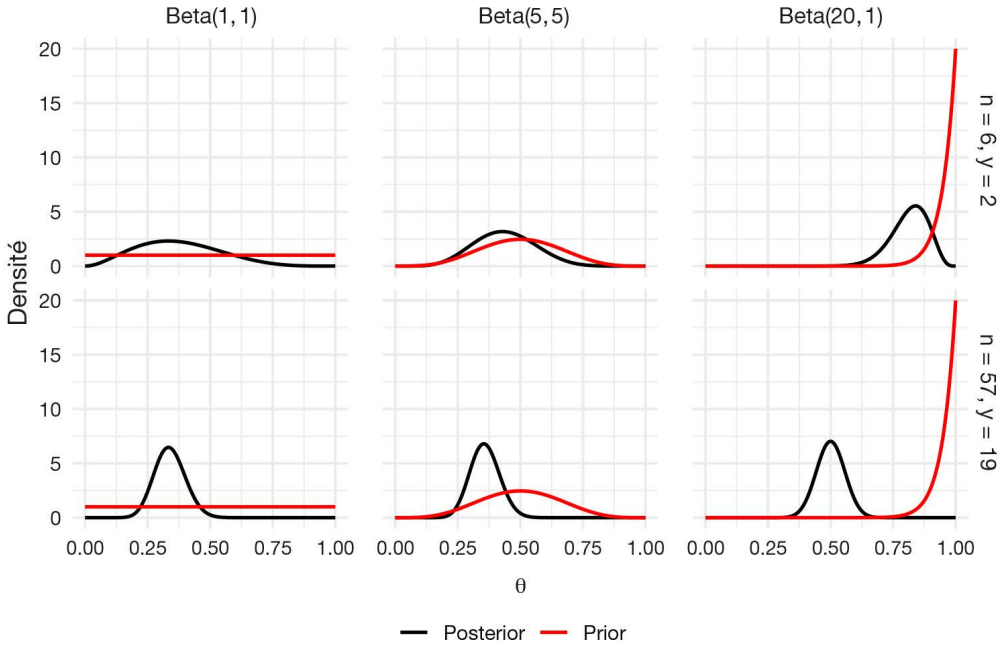

Figure 4.1. Effet combiné du prior et de la taille d'échantillon sur la distribution a posteriori avec une vraisemblance binomiale. En colonnes : trois lois bêta a priori Beta(1,1), Beta(5,5) et Beta(20,1). En lignes : petit (n = 6, y = 2) et grand (n = 57, y = 19) échantillon (facteur 10). Le trait rouge représente le prior ; le trait noir, la distribution a posteriori.

Avec peu de données (ligne du haut), l'effet du prior est visible : la distribution a posteriori de la survie reste proche du prior, en particulier avec la Beta(20,1) qui tire l'estimation vers des valeurs élevées. Avec plus de données (ligne du bas), la distribution a posteriori est dominée par la vraisemblance : elle se concentre autour de la proportion observée, à part pour le prior Beta(20,1) avec lequel la distribution a posteriori est centrée sur 0.5. On observe ainsi un principe fondamental de l'inférence bayésienne : plus les données sont nombreuses et informatives, moins le prior influence les résultats.

On peut formaliser les observations faites sur la **figure 4.1**. Rappelez-vous que lorsque la vraisemblance est Bin(n,θ) avec y succès, et que le prior est une loi Beta(a,b), la loi a posteriori est également bêta (conjugaison), et plus exactement Beta($a + y$, $b + n - y$). Or, la moyenne d'une Beta(a,b) est $\frac{a}{a+b}$, et donc la moyenne de la distribution a posteriori Beta($a + y$, $b + n - y$) est $\frac{a+y}{a+b+n}$, qui peut se réécrire comme une moyenne pondérée entre la moyenne de la distribution

a priori $\mu_{prior} = \frac{a}{a+b}$ et la proportion observée *y*/*n* qui n'est autre que l'estimateur du maximum de vraisemblance $\hat{\theta}$, avec le poids $w = \frac{n}{a+b+n}$. Attention, c'est un poids au sens statistique du terme, un facteur de pondération, pas au sens « kilos de ragondin ». Autrement dit, la moyenne de la distribution a posteriori vaut $(1-w)\mu_{prior} + w\hat{\theta}$. Ainsi, quand l'échantillon de taille *n* est grand, *w* tend vers 1, et la moyenne a posteriori se rapproche de l'estimateur du maximum de vraisemblance. À l'inverse, pour un petit échantillon ou un prior très informatif (la somme *a* + *b* est grande, voir **figure 1.3**), *w* est petit, et le prior tire davantage l'estimation. En résumé, quand les données sont limitées, on s'appuie davantage sur le prior ; quand elles sont riches, on laisse parler la vraisemblance.

En conclusion, c'est toujours une bonne idée de faire ce genre d'analyse de sensibilité. En comparant les résultats obtenus avec différents priors (non informatifs, peu informatifs, informatifs), on peut s'assurer que les conclusions ne dépendent pas excessivement des choix a priori. Si c'est le cas, pas de panique, ça veut simplement dire qu'on a peu d'informations sur le paramètre en question et qu'il faut redoubler de prudence et bien réfléchir au prior utilisé. On y reviendra par la suite.

4.3. COMMENT INTÉGRER L'INFORMATION A PRIORI ?

4.3.1. Méta-analyse

Repartons de notre exemple fil rouge sur l'estimation d'une probabilité de survie, mais en le complexifiant légèrement pour tenir compte d'un problème fréquent lorsqu'on étudie des populations animales : la détection imparfaite des individus. En effet, selon leur comportement ou les conditions de terrain, un animal peut très bien être vivant et présent, mais ne pas être détecté au moment de l'échantillonnage. Pour corriger ce biais, on utilise souvent des protocoles de capture-recapture, qui reposent sur l'identification individuelle des animaux, grâce à une bague, un motif de pelage, un profil génétique, etc.

Un individu peut ainsi être détecté (1) ou pas (0) ; on code par exemple 101, qui signifie : vu la première année, manqué la seconde, puis revu la troisième. Dans le modèle le plus simple, on suppose une probabilité de survie θ constante et une probabilité de détection p constante. La vraisemblance pour l'historique 101 est donc : $\Pr(101) = \theta\,(1-p)\,\theta\,p$. Pour obtenir la vraisemblance complète, on fait ce calcul pour chaque individu et on suppose que tous partagent les mêmes θ et p, et qu'ils sont indépendants.

Pour changer des ragondins, intéressons-nous au cincle plongeur (*Cinclus cinclus*), un oiseau étudié pendant plus de quarante ans par Gilbert Marzolin, un professeur de mathématiques passionné d'ornithologie avec qui j'ai eu la chance de travailler. Nous disposons ici de données de capture-recapture sur sept ans (1981-1987) pour plus de 200 oiseaux.

On va démarrer par un prior non informatif sur la probabilité de survie, au hasard une Beta(1,1). Ce sera notre modèle A. Comme autre prior, on peut s'appuyer sur des connaissances accumulées pour des espèces similaires. Chez les passereaux, on observe par exemple une relation entre la masse corporelle et la probabilité de survie : en moyenne, les oiseaux plus lourds vivent plus longtemps. Cette relation, dite allométrique, a été quantifiée par McCarthy (2007) grâce à une régression linéaire (voir chapitre 5), à partir des données de survie et de masse pour 27 espèces de passereaux européens. En utilisant cette régression sur les passereaux au cas particulier du cincle, et sachant que le cincle pèse en moyenne 59.8 grammes, on peut prédire sa probabilité de survie annuelle. Le modèle fournit ainsi une estimation de 0.57 avec une erreur standard de 0.075. Ces valeurs nous permettent de définir un prior informatif, sous la forme d'une loi normale centrée en 0.57 et de variance 0.075^2. Ce sera notre modèle B.

On obtient ainsi les résultats suivants pour le cincle :

Modèle	Prior pour θ	Survie moyenne a posteriori	Intervalle de crédibilité à 95 %
A	Beta(1,1)	0.56	[0.51 ; 0.61]
B	N(0.57, 0.075^2)	0.56	[0.52 ; 0.61]

Avec un jeu de données riche (sept années), l'information contenue dans la vraisemblance domine ; le prior informatif n'apporte presque aucune information et les deux modèles produisent des résultats très proches.

Imaginons maintenant qu'on a des données limitées. Que se passe-t-il si l'on ne dispose que des trois premières années par exemple ? On refait l'analyse et les résultats sont maintenant :

Modèle	Prior pour θ	Survie moyenne a posteriori	Intervalle de crédibilité à 95 %
A	Beta(1,1)	0.70	[0.47 ; 0.95]
B	N(0.57, 0.075^2)	0.60	[0.48 ; 0.72]

Cette fois, le prior informatif fait une vraie différence. On réduit la largeur de l'intervalle de près de 50 % (on est plus précis), tout en ramenant l'estimation moyenne vers une valeur plus réaliste pour un passereau. On note aussi que l'estimation postérieure du modèle B avec trois années de données est proche de celle obtenue avec sept années (**figure 4.2**).

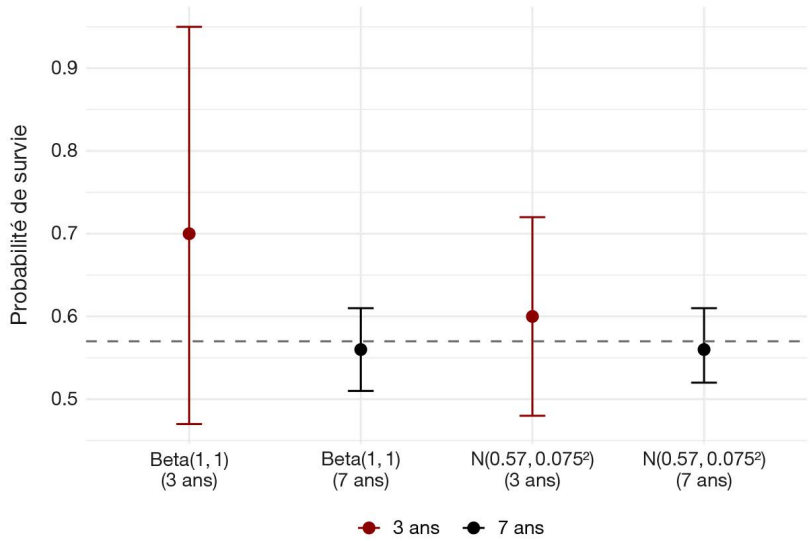

Figure 4.2. Comparaison des estimations a posteriori de la survie du cincle plongeur selon le type de prior (Beta(1,1) ou N(0.57, 0.075^2)) et la durée de l'étude (3 ou 7 ans). Chaque point représente la moyenne a posteriori, avec son intervalle de crédibilité à 95 %. La ligne grise indique la valeur de survie issue de la méta-analyse pour les passereaux (0.57).

Cet exemple montre que les données issues de la littérature (ici une relation allométrique masse-survie obtenue *via* une méta-analyse) peuvent être utilisées pour construire un prior informatif pertinent, capable d'améliorer sensiblement la précision des estimations, en particulier lorsque les données sont limitées. Cette approche offre une alternative peu coûteuse à l'allongement des protocoles de terrain, pour peu évidemment que la question (relativement simple) reste l'estimation d'une seule survie.

4.3.2. Méthode du moment-matching

Dans l'exemple du cincle, on a utilisé une distribution normale comme prior informatif pour un paramètre qui se trouve être une probabilité. Or, la normale peut prendre des valeurs négatives ou supérieures à 1, ce qui n'est pas souhaitable pour une probabilité. Dans l'exemple, le prior informatif $N(0.57,0.075^2)$ est en moyenne entre 0 et 1 avec une petite variance, donc peu de chance que cela tourne mal. Vous pouvez vous en rendre compte en simulant des données avec R et la commande summary(rnorm(n = 100, mean = 0.57, sd = 0.075)). Malgré tout, ça n'est pas très satisfaisant.

La bonne nouvelle, c'est que l'on peut construire un prior informatif plus adéquat pour une probabilité par la méthode dite du « moment-matching ». La méthode du moment-matching sert à choisir les paramètres d'une loi a priori en les calant sur les moments (souvent la moyenne et la variance) qui traduisent l'information a priori qu'on possède (avant de voir les données).

Lorsque l'information a priori est disponible sous forme d'une moyenne μ et d'un écart-type σ, on peut transformer ces moments en paramètres a, b d'une distribution bêta. Pour rappel, la moyenne et la variance d'une bêta de paramètres a et b sont $\mu = \frac{a}{a+b}$ et $\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$.

En inversant ces relations, on obtient $a = \left(\frac{1-\mu}{\sigma^2} - \frac{1}{\mu}\right)\mu^2$ et $b = a\left(\frac{1}{\mu} - 1\right)$. Dans notre exemple, on a $\mu = 0.57$ et $\sigma = 0.075$ dont on peut déduire $a = 24.3$ et $b = 18.3$ avec quelques lignes de code :

```
# paramètres de moyenne et d'écart-type souhaités pour la distribution bêta
mu <- 0.57 # moyenne de la probabilité
sigma <- 0.075 # écart-type sur cette probabilité
# formules inverses pour obtenir les paramètres a et b d'une distribution bêta
a <- ((1 - mu) / (sigma^2) - 1 / mu) * mu^2
b <- a * (1 / mu - 1)
# affichage des valeurs de a et b arrondies
c(a = round(a, 1), b = round(b, 1))
#>    a    b
#> 24.3 18.3
```

On peut vérifier que cette distribution bêta a bien, pour moyenne et écart-type, les valeurs données par la méta-analyse :

```
# on tire 1 0000 valeurs dans loi bêta avec a = 24.3 et b = 18.3
ech_prior <- rbeta(n = 10000, shape1 = 24.3, shape2 = 18.3)
# on calcule la moyenne empirique des tirages (doit approcher 0.57)
mean(ech_prior)
#> [1] 0.570905
# on calcule l'écart-type empirique des tirages (doit approcher 0.075)
sd(ech_prior)
#> [1] 0.0749117
```

On peut donc adopter un prior Beta(a = 24.3, b = 18.3) pour tenir compte de l'information moyenne et de sa variabilité obtenue par la relation allométrique survie-masse.

La méthode du moment-matching ne s'applique pas qu'aux probabilités. On peut aussi s'en servir pour construire un prior pour un paramètre réel, par exemple l'effet de la masse des ragondins sur leur survie (voir chapitre 5). Imaginons qu'un expert dise : « Je suis à 80 % sûr que le paramètre θ se trouve entre –0.15 et 0.25. » Cette phrase définit un intervalle de crédibilité à 80 % : $\Pr(\theta \in [-0.15, 0.25]) = 0.80$. On cherche un prior de type loi normale $\theta \sim N(\mu, \sigma^2)$ qui reflète exactement cette information.

On peut commencer par la moyenne μ. L'intervalle est symétrique, donc on peut déduire directement que la moyenne μ du prior est le milieu de l'intervalle : $\mu = \dfrac{-0.15 + 0.25}{2} = 0.05$.

Passons à l'écart-type σ. L'expert affirme que 80 % des valeurs de θ sont comprises entre –0.15 et 0.25. Or, pour une loi normale, cette proportion s'écrit $\Pr(\mu - z\,\sigma \leq \theta \leq \mu + z\,\sigma) = 0.80$. Cela signifie que 80 % de la masse de la distribution est contenue dans un intervalle centré sur μ et de largeur $2z\sigma$. Pour un niveau de 80 %, la valeur de z vaut environ 1.2816 (on l'obtient *via* qnorm(0.90) où 0.90 est le quantile supérieur $1 - \alpha/2 = 1 - 20/2$ avec $(1 - \alpha)\% = 80\%$ et donc $\alpha = 0.20$). Finalement, on obtient $\sigma = \dfrac{0.25 - \left(-0.15\right)}{2 \times 1.2816} \approx 0.156$. Voici le calcul dans R :

```
# bornes inférieure et supérieure données par l'expert
a <- -0.15
b <-  0.25

# niveau de confiance exprimé
level <- 0.80
alpha <- 1 - level

# valeur z correspondant à un intervalle de crédibilité à 80 %
z <- qnorm(1 - alpha / 2)  # ≈ 1.2816

# moyenne = centre de l'intervalle
mu <- (a + b) / 2

# écart-type déduit de la largeur de l'intervalle
sigma <- (b - a) / (2 * z)

mu
#> [1] 0.05
sigma
#> [1] 0.1560608
```

On conclut que le prior informatif voulu est une loi normale, $N(\mu = 0.05, \sigma = 0.156)$. On peut vérifier que tout s'est bien passé :

```
mu    <- 0.05
sigma <- 0.1560608
pnorm(c(-0.15, 0.25), mean = mu, sd = sigma)
#> [1] 0.09999996 0.90000004
#> 0.10 0.90    # Ok : 10 % à gauche, 90 % à droite → 80 % au centre
```

Visuellement, la **figure 4.3** représente la densité d'une loi normale avec une moyenne $\mu = 0.05$ et un écart-type $\sigma = 0.156$. L'intervalle en bleu clair correspond à l'intervalle crédible central à 80 %, c'est-à-dire l'intervalle [–0.15 ; 0.25] qui contient 80 % de la masse de probabilité. Le pointillé gris indique les bornes de cet intervalle, tandis que le pointillé noir marque la position de la moyenne. On observe que, grâce à la symétrie de la loi normale, l'intervalle est centré autour de la moyenne, et que 10 % de la masse est située de chaque côté en dehors de cet intervalle.

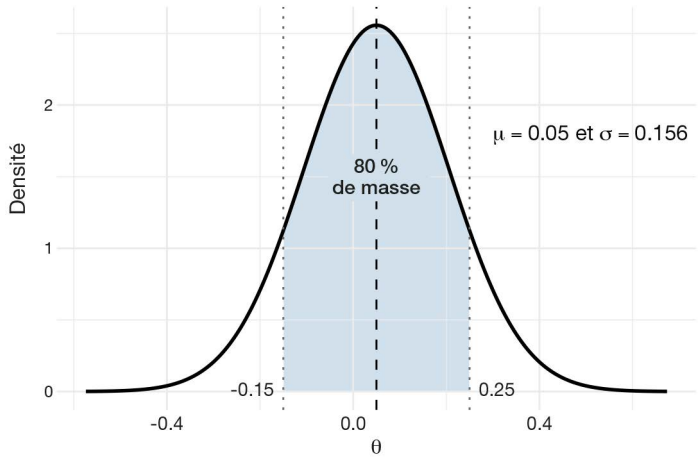

Figure 4.3. Distribution normale avec une moyenne de 0.05 et un écart-type de 0.156. L'intervalle ombré correspond à l'intervalle de crédibilité à 80 %, entre –0.15 et 0.25.

4.4. ATTENTION AUX PRIORS DITS NON INFORMATIFS

En statistique bayésienne, on utilise souvent des priors non informatifs. Mais attention, les apparences peuvent être trompeuses, surtout lorsqu'on travaille sur des paramètres définis sur des échelles transformées, comme le logit ou le log dans les modèles linéaires généralisés (voir chapitre 6). Prenons un exemple courant, où l'on modélise une probabilité θ sur l'échelle logit *via* un paramètre β tel que logit(θ) = β.

En pratique, on peut utiliser les simulations pour vérifier que des priors ne réservent pas de mauvaises surprises après transformation, c'est ce que l'on appelle des *prior predictive checks*. Ça se passe avant même d'ajuster un modèle, et pour ce faire on va :

– simuler des valeurs depuis le prior de β sur l'échelle logit ;
– appliquer la transformation réciproque du logit pour obtenir θ ;
– inspecter la distribution a priori induite sur θ et juger si elle semble réaliste.

Un premier choix est de prendre comme prior une loi normale avec une grande variance, par exemple $\beta \sim N(0, 10^2)$. Les étapes 1 et 2 s'obtiennent *via* :

```
logit_prior <- rnorm(n = 1000, mean = 0, sd = 10) # simulation
prior <- plogis(logit_prior) # transformation
```

Le problème est qu'après transformation avec la fonction réciproque du logit la majorité des valeurs simulées, et donc la probabilité θ, sont proches de 0 ou de 1 comme on le voit dans la **figure 4.4** (panneau de gauche), ce qui favorise de manière implicite des valeurs extrêmes. On passe d'un prior non informatif sur l'échelle logit à un prior très informatif (sans le vouloir) sur l'échelle naturelle de la probabilité.

Un autre choix consiste à prendre $\beta \sim N(0, 1.5^2)$. Les deux premières étapes de la simulation se résument à :

```
logit_prior2 <- rnorm(n = 1000, mean = 0, sd = 1.5)
prior2 <- plogis(logit_prior2)
```

Ici, la distribution induite sur θ est uniforme, couvrant surtout la plage de valeurs entre 0.05 et 0.95, comme on peut le voir dans la **figure 4.4** (panneau de droite), ce qui reflète mieux un manque d'information sur θ. Ce deuxième choix est le bon, on parle de priors faiblement informatifs (ou *weakly informative*).

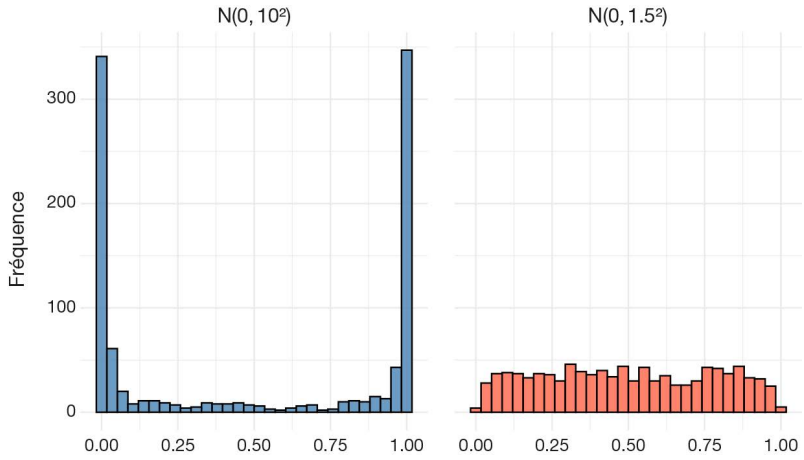

Figure 4.4. Comparaison de deux priors obtenus sur la probabilité $\theta = \text{logit}^{-1}(\beta)$ après transformation par la fonction logit réciproque de $\beta \sim N(0, 10^2)$ (à gauche) et $\beta \sim N(0, 1.5^2)$ (à droite). L'axe des abscisses représente les différentes valeurs possibles de la probabilité θ obtenues après transformation par la fonction logit inverse. L'axe des ordonnées indique la fréquence des tirages simulés pour chaque valeur.

Il existe aussi des priors invariants, c'est-à-dire dont la forme tient compte de l'échelle du paramètre. Le prior de Jeffreys en est un exemple : il maximise l'information apportée par les données, tout en restant invariant par reparamétrisation. Par exemple, pour une probabilité θ, le prior de Jeffreys est Beta(0.5,0.5). Ce prior est moins plat qu'une uniforme Beta(1,1). Il est souvent utilisé lorsqu'on souhaite une approche objective, sans introduire d'information subjective. En pratique toutefois, le prior de Jeffreys est difficile à calculer, et on privilégiera l'approche par simulations pour s'assurer que les paramètres transformés ont des priors raisonnables.

À RETENIR

- Plus les données sont riches, moins le prior influence l'estimation a posteriori.
- N'hésitez pas à prendre le temps de visualiser vos priors sur l'échelle naturelle des paramètres à l'aide de simulations.
- Les méthodes de moment-matching offrent un moyen pratique de transformer, d'encoder des connaissances dans les paramètres de distributions qui peuvent servir de priors (bêta ou normale par exemple).
- Quand utiliser quel type de prior ?

Type de prior	Usage recommandé	Avantages	Précautions
Informatif	Quand on dispose d'informations solides (expertise, méta-analyse, etc.)	Intègre les connaissances disponibles, utile avec peu de données	Risque de biais si mal calibré
Faiblement informatif	Par défaut, si l'on souhaite guider l'inférence sans la contraindre	Protège contre des valeurs aberrantes, améliore la stabilité numérique	Doit être adapté à l'échelle du paramètre
Non informatif	Cas exploratoires, ou pour laisser parler les données	Ne privilégie a priori aucune valeur particulière	Peut être trompeur sur des échelles transformées (logit, log)
De référence / Jeffreys	Quand on recherche une approche invariante (une Beta(0.5,0.5) dans l'exemple fil rouge)	Invariant par changement de paramétrisation	Parfois difficile à calculer ou à interpréter

5. La régression

Ce chapitre présente l'application de la statistique bayésienne à la régression linéaire. On prendra un exemple qui nous permettra d'aller un peu plus loin que notre exemple fil rouge sur la survie. Ce sera l'occasion d'aborder comment et pourquoi utiliser un modèle pour simuler des données. Nous en profiterons pour illustrer la comparaison et la validation des modèles. Nous utiliserons brms (l'équivalent en NIMBLE est disponible dans la version enrichie du livre en ligne[9]) et comparerons avec l'approche fréquentiste.

5.1. LA RÉGRESSION LINÉAIRE

5.1.1. Le modèle

Pour changer un peu, je vous propose d'utiliser brms sur un exemple différent de celui de l'estimation de la survie. Attardons-nous sur la régression linéaire.

Commençons par poser les bases de notre modèle linéaire. On a n mesures d'une variable réponse y_i avec i qui varie de 1 à n. Pensez par exemple à la masse (en kilogrammes) de nos ragondins dans l'exemple fil rouge. On associe chaque mesure à une variable explicative x_i, par exemple la température extérieure moyenne en hiver (en degrés Celsius ou °C) pour nos ragondins. On cherche à étudier l'effet de la température sur la masse. Le plus simple est de supposer une relation linéaire entre les deux, on utilise donc un modèle de régression linéaire. Le modèle comporte une ordonnée à l'origine (ou intercept) β_0, et une pente β_1 qui décrit l'effet de x_i sur y_i, ou de la température sur la masse des ragondins. On a aussi besoin d'un paramètre pour décrire la variabilité résiduelle représentée par un paramètre de variance σ^2, qui capte la part de variation dans les y_i non expliquée par les x_i. Vous avez probablement déjà rencontré ce modèle sous la forme : $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$ où les erreurs ε_i sont supposées indépendantes et distribuées selon une loi normale de moyenne 0 et de variance σ^2.

L'intercept β_0 nous donne la masse quand la température est de 0 °C ($x_i = 0$). Le paramètre β_1 nous renseigne sur le changement dans la variable réponse pour une augmentation d'une unité (ici, 1 °C) de la variable explicative (d'où le terme « pente » pour désigner ce paramètre). En général, on conseille (fortement) de centrer (soustraire la moyenne) et réduire (diviser par l'écart-type) les valeurs de la variable explicative pour des questions numériques et d'interprétation : numérique d'abord, car cela permet aux algorithmes, qu'ils soient fréquentistes ou bayésiens, de ne pas se perdre dans des recoins de l'espace du paramètre ; interprétation ensuite, car on interprète alors l'intercept β_0 comme la valeur de la variable réponse pour une valeur moyenne de la variable explicative.

Dans cette section, plutôt que d'analyser de « vraies » données, nous allons, à partir des paramètres β_0, β_1 et σ, simuler des données artificielles, comme si elles provenaient d'un vrai processus sous-jacent.

5.1.2. Simuler des données

Qu'est-ce que j'entends par simuler des données ? L'analyse et la simulation des données sont deux faces d'un même modèle. Dans l'analyse, on utilise les données pour estimer les paramètres d'un modèle. Dans la simulation, on fixe les paramètres et on utilise le modèle pour générer des données. Une raison d'utiliser les simulations est que cette gymnastique va nous obliger à bien comprendre le modèle ; si je n'arrive pas à simuler des données à partir d'un modèle, c'est que je n'ai pas complètement compris comment il marchait. Il y a des tas d'autres bonnes raisons pour utiliser les simulations. Comme la vérité (les paramètres et le modèle) est connue, on peut vérifier que le modèle est bien codé. On peut évaluer le biais et la précision des estimations de nos paramètres, évaluer les effets de ne pas respecter les hypothèses du modèle, planifier un protocole de récolte

9. https://oliviergimenez.github.io/statistique-bayes/lms.html

de données ou encore évaluer la puissance d'un test statistique. Bref, c'est une technique très utile à avoir dans votre boîte à outils !

Revenons à notre exemple. Pour simuler des données selon le modèle de régression linéaire, on commence par fixer nos paramètres : $\beta_0 = 0.1$, $\beta_1 = 1$ et $\sigma^2 = 0.5$:

```
beta0 <- 0.1 # valeur vraie de l'intercept
beta1 <- 1 # valeur vraie du coefficient de x
sigma <- 0.5 # écart-type des erreurs
```

Puis, on simule $n = 100$ valeurs x_i de notre variable explicative selon une loi normale de moyenne 0 et d'écart-type 1, autrement dit $N(0,1)$:

```
set.seed(666) # pour rendre la simulation reproductible
n <- 100 # nombre d'observations
x <- rnorm(n = n, mean = 0, sd = 1) # covariable x simulée
```

Enfin, on simule les valeurs de la variable réponse, en ajoutant une erreur normale epsilon à la relation linéaire qu'on a codée beta0 + beta1 * x :

```
epsilon <- rnorm(n, mean = 0, sd = sigma) # génère les erreurs normales
y <- beta0 + beta1 * x + epsilon # ajoute les erreurs à la relation linéaire
data <- data.frame(y = y, x = x)
```

La **figure 5.1** ci-dessous montre les données simulées, ainsi que la droite de régression correspondant au modèle utilisé pour les générer.

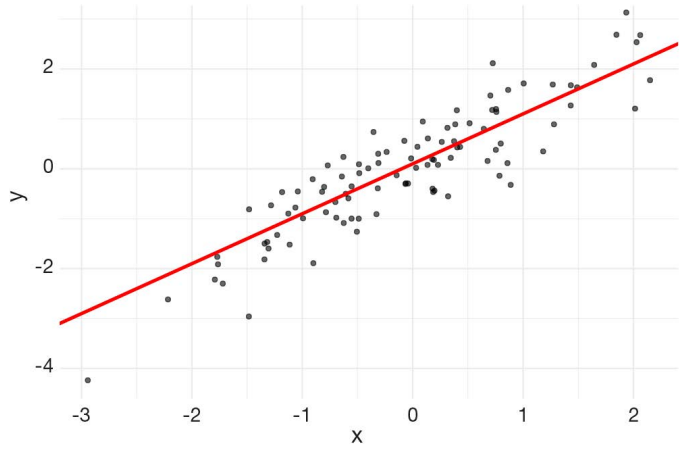

Figure 5.1. Données simulées (n = 100) selon le modèle $y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$, avec $\beta_0 = 0.1$, $\beta_1 = 1$ et $\sigma^2 = 0.5$. La droite rouge correspond à la droite de régression.

5.1.3. L'ajustement avec brms

Dans cette section, on utilise brms pour ajuster le modèle de régression linéaire aux données qu'on vient de générer. Si tout se passe bien, les paramètres estimés devraient être proches des valeurs utilisées pour générer les données. Je vais relativement vite ici, puisque l'on a couvert les différentes étapes dans le chapitre 3. La syntaxe est très proche de celle que l'on utiliserait pour ajuster le modèle par maximum de vraisemblance avec la fonction lm() dans R :

```
lm.brms <- brm(y ~ x, # formule : y en fonction de x
               data = data, # jeu de données
               family = gaussian) # distribution normale
```

Jetons un coup d'œil aux résumés numériques et aux diagnostics de convergence :

```
summary(lm.brms)
#>  Family: gaussian
#>   Links: mu = identity; sigma = identity
#> Formula: y ~ x
#>    Data: data (Number of observations: 100)
#>   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
#>          total post-warmup draws = 4000
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept     0.06      0.06    -0.05     0.17 1.00     4138     3110
#> x             1.10      0.06     0.99     1.21 1.00     3735     3192
#>
#> Further Distributional Parameters:
#>       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> sigma     0.57      0.04     0.49     0.66 1.00     3951     2849
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Par défaut, brms a utilisé quatre chaînes qui ont tourné pendant 2 000 itérations, chacune avec 1 000 itérations utilisées comme burn-in, soit au total 4 000 itérations pour l'inférence a posteriori. Dans les sorties, Intercept, x et sigma correspondent respectivement aux paramètres β_0, β_1 et σ du modèle. Le $\widehat{R}$ pour les trois paramètres vaut 1, et les tailles d'échantillon efficaces sont satisfaisantes. Les intervalles de crédibilité contiennent la vraie valeur du paramètre utilisée pour simuler les données.

On vérifie que le mixing est bon (**figure 5.2**).

```
plot(lm.brms)
```

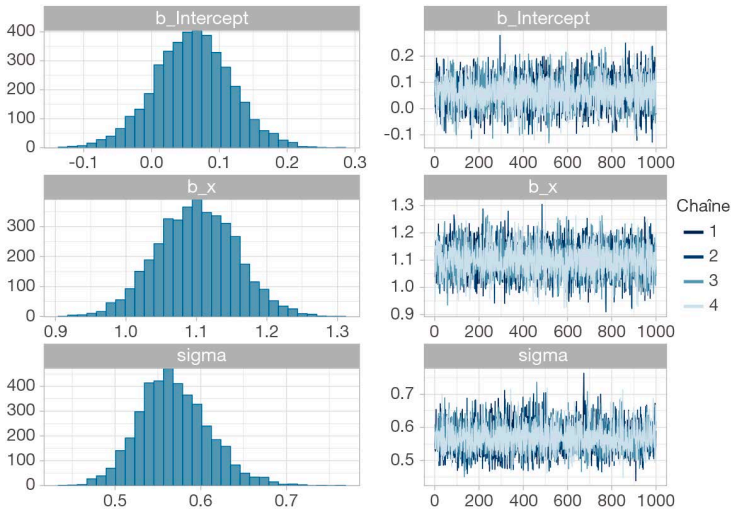

Figure 5.2. Histogrammes des distributions a posteriori (colonne de gauche) et traces (colonne de droite) des paramètres de la régression linéaire. Dans les histogrammes, l'axe des abscisses représente les valeurs possibles du paramètre estimé (intercept, pente ou écart-type) et l'axe des ordonnées correspond à leur fréquence dans l'échantillon a posteriori. Dans les *trace plots*, l'axe des abscisses indique le numéro d'itération du MCMC, tandis que l'axe des ordonnées représente la valeur simulée du paramètre à chaque itération.

5.1.4. Des priors faiblement informatifs

Plutôt que d'utiliser les priors par défaut de brms, choisissons d'autres priors. Nous allons utiliser des priors faiblement informatifs, et plus spécifiquement une normale avec moyenne 0 et écart-type 1.5 ou $N(0,1.5)$ pour les paramètres de régression β_0 et β_1. On a déjà parlé des priors faiblement informatifs dans le chapitre 4. L'idée est proche de celle des priors vagues ou non informatifs, dans le sens où l'on s'efforce de refléter *via* les priors faiblement informatifs le fait que l'on n'a pas vraiment d'information sur les paramètres du modèle. La différence est que les priors non informatifs peuvent induire des valeurs aberrantes comme on l'a vu au chapitre 4. C'est encore le cas ici. Prenez par exemple des $N(0,100)$ pour les paramètres de la relation linéaire qui lie la masse des ragondins à la température, et simulez tout un tas de valeurs dans ces priors, puis formez la relation linéaire :

```
# nombre de droites à simuler
n_lines <- 100

# tirages des intercepts et pentes selon les priors
intercepts <- rnorm(n_lines, mean = 0, sd = 100)
slopes <- rnorm(n_lines, mean = 0, sd = 100)

# création d'un data frame
lines_df <- data.frame()
for (i in 1:n_lines) {
  y_vals <- intercepts[i] + slopes[i] * x
  temp_df <- data.frame(x = x, y = y_vals, line = as.factor(i))
  lines_df <- rbind(lines_df, temp_df)
}

# tracé avec ggplot2
ggplot(lines_df, aes(x = x, y = y, group = line)) +
  geom_line(alpha = 0.3) +
  theme_minimal() +
  labs(x = "x", y = "y")
```

On voit dans la **figure 5.3** qu'on obtient des valeurs aberrantes pour les y_i, avec des ragondins de plus de 400 kg et des valeurs (très) négatives pour la masse. On vient de faire un *prior predictive check*, comme au chapitre 4.

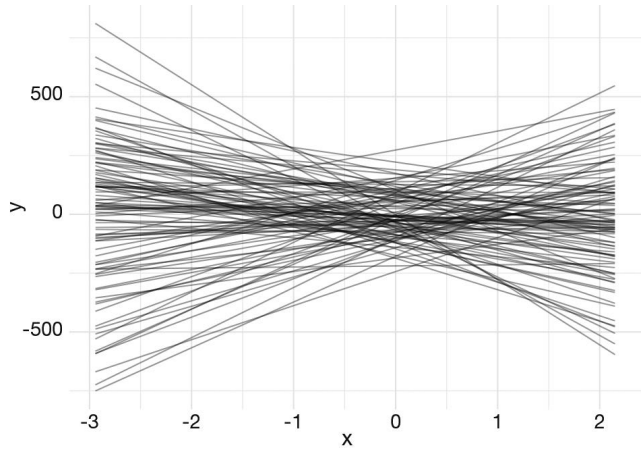

Figure 5.3. Simulation de droites de régression issues des distributions a priori. Chaque ligne correspond à un tirage des paramètres : intercept et pente ~ N(0, 100).

On fait la même chose avec notre prior faiblement informatif $N(0,1.5)$ et on obtient la **figure 5.4**.

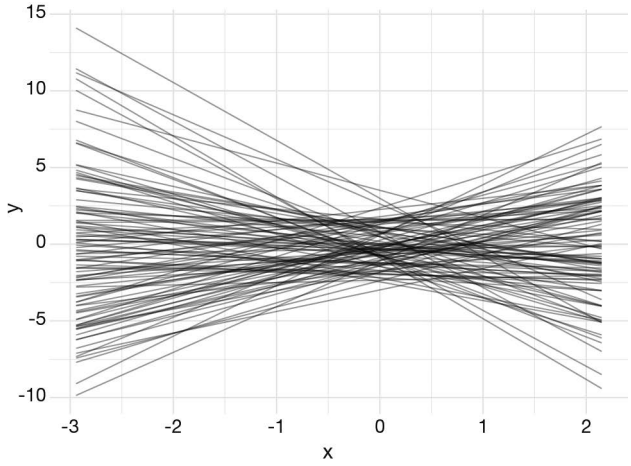

Figure 5.4. Simulation de droites de régression issues des distributions a priori. Chaque ligne correspond à un tirage des paramètres : intercept et pente ~ N(0, 1.5).

On obtient des valeurs plus raisonnables pour la masse des ragondins qui dépassent rarement 10 kg. On a toujours des valeurs négatives, mais moindres, l'algorithme MCMC devrait s'en sortir. Il y a aussi un avantage numérique à utiliser des priors faiblement informatifs, ils aident les méthodes MCMC à ne pas se perdre dans l'espace de toutes les valeurs possibles pour les paramètres à estimer, et leur permettent de se focaliser sur les valeurs réalistes de ces paramètres. En faisant cela, vous avez peut-être l'impression que l'on utilise les données pour construire les priors, alors que le prior est censé refléter l'information disponible avant de voir les données. C'est l'occasion de préciser un peu ce point. L'important est surtout que le prior représente l'information indépendante des données qui sont utilisées dans la vraisemblance.

On s'est jusqu'ici concentré sur les paramètres de régression, l'intercept β_0 et la pente β_1. Mais qu'en est-il de l'écart-type, σ? Ce paramètre est tout aussi important : il reflète à quel point les observations s'écartent de la tendance moyenne décrite par la droite de régression.

Une option souvent envisagée est de lui attribuer une loi uniforme, par exemple $\sigma \sim U(0, B)$, avec une borne inférieure naturelle (0, puisque σ est toujours positive), mais une borne supérieure B difficile à choisir. Quelle valeur maximale donner à un écart-type? Dans certains cas, une valeur apparemment raisonnable peut se révéler trop large. Par exemple, si l'on modélise des tailles humaines et que l'on fixe $\sigma \sim U(0, 50)$ (en cm), cela revient à supposer que 95 % des tailles sont réparties sur une plage de 100 cm autour de la moyenne – ce qui est très improbable.

Une option plus souple et plus réaliste consiste à utiliser une loi exponentielle $\sigma \sim \exp(\lambda)$, où $\lambda > 0$ est un paramètre de taux. Cette loi est définie uniquement pour des valeurs positives, ce qui est cohérent avec la nature de σ, et elle favorise les petites valeurs d'écart-type, tout en laissant la possibilité à σ d'être plus grande si les données le justifient.

Par défaut, on prend souvent $\lambda = 1$. Avec $\lambda = 1$, la moyenne et l'écart-type de cette loi sont tous deux égaux à 1, ce qui induit une loi a priori modeste mais non restrictive (**figure 5.5**).

On peut formaliser ce modèle comme suit :

$$
\begin{aligned}
y_i &\sim \text{Normale}(\mu_i, \sigma^2) && \text{[vraisemblance]}\\
\mu_i &= \beta_0 + \beta_1 x_i && \text{[relation linéaire]}\\
\beta_0, \beta_1 &\sim \text{Normale}(0, 1.5) && \text{[prior sur les paramètres]}\\
\sigma &\sim \text{Exp}(1) && \text{[prior sur les paramètres]}
\end{aligned}
$$

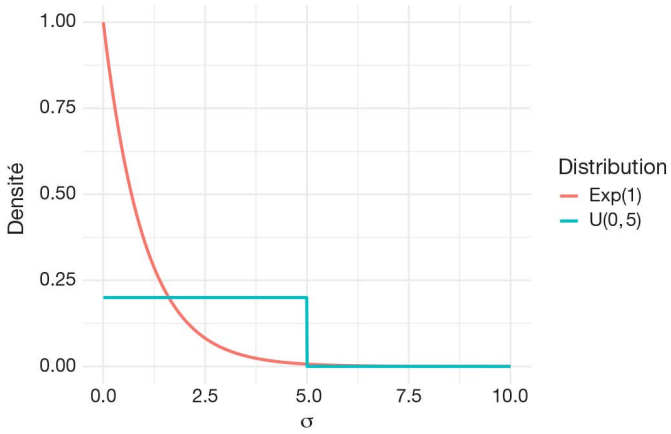

Figure 5.5. Comparaison entre deux lois a priori pour l'écart-type σ : une loi uniforme U(0,5), qui donne la même densité entre 0 et 5, et une loi exponentielle Exp(1), qui favorise les petites valeurs tout en conservant une queue plus lourde.

Spécifions ces priors :

```
myprior <- c(
  prior(normal(0, 1.5), class = b), # prior sur le coefficient de x
  prior(normal(0, 1.5), class = Intercept), # prior sur l'intercept
  prior(exponential(1), class = sigma) # prior sur l'écart-type de l'erreur
)
```

Puis, refaisons l'ajustement avec brms :

```
lm.brms <- brm(y ~ x,
               data = data,
               family = gaussian,
               prior = myprior)
```

On vérifie que les résumés numériques obtenus sont proches de ceux obtenus avec les priors par défaut, et surtout des valeurs utilisées pour simuler les données :

```
summary(lm.brms)
#>  Family: gaussian
#>   Links: mu = identity; sigma = identity
#> Formula : y ~ x
#>    Data: data (Number of observations: 100)
#>   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
#>          total post-warmup draws = 4000
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept     0.06      0.06    -0.06     0.18 1.00     3648     2815
#> x             1.10      0.06     0.99     1.20 1.00     3778     2918
#>
#> Further Distributional Parameters:
#>       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> sigma     0.57      0.04     0.49     0.66 1.00     3555     2593
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1)
```

Ici, les deux modèles donnent quasiment la même chose, ce qui n'a rien de surprenant, car les données sont suffisamment informatives pour qu'elles « prennent le dessus sur » le prior. L'intérêt des priors faiblement informatifs ne se voit pas tant dans ce petit exemple que dans d'autres situations : ils évitent les valeurs aberrantes, stabilisent les calculs MCMC et restent utiles quand on a moins de données ou des modèles plus complexes.

5.1.5. L'ajustement par maximum de vraisemblance

Et pour finir, on peut comparer avec l'ajustement par maximum de vraisemblance qu'on obtient simplement avec la commande lm(y ~ x, data = data). Tout est dans la **figure 5.6**.

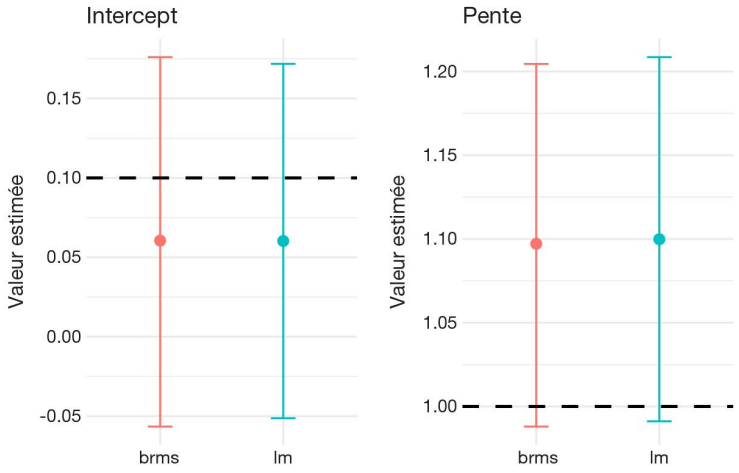

Figure 5.6. Comparaison des estimations des paramètres du modèle (intercept ou ordonnée à l'origine et pente) selon les différentes méthodes (brms et lm). Les points donnent les moyennes a posteriori pour brms et l'estimation du maximum de vraisemblance pour lm. On donne également les intervalles de crédibilité (pour brms) et de confiance (pour lm) à 95 %. La ligne en tirets noirs indique la vraie valeur utilisée pour simuler les données.

Les moyennes a posteriori obtenues avec brms sont proches des estimations par maximum de vraisemblance pour l'intercept et la pente, dans une moindre mesure. Les intervalles de crédibilité obtenus avec brms et l'intervalle de confiance obtenu par maximum de vraisemblance englobent tous les vraies valeurs des paramètres qui ont servi à simuler les données. Gardez à l'esprit qu'il s'agit d'une seule simulation, il faudrait répéter l'exercice un grand nombre de fois pour évaluer formellement la distance entre les vraies valeurs et les estimations des paramètres (le biais).

5.2. L'ÉVALUATION DES MODÈLES

La qualité de l'ajustement d'un modèle aux données est essentielle pour évaluer la confiance que l'on peut accorder aux estimations des paramètres. Les tests de qualité d'ajustement (ou *goodness-of-fit* en anglais) sont bien établis en statistique fréquentiste, et beaucoup d'entre eux peuvent aussi être utilisés dans des modèles bayésiens simples. C'est le cas par exemple de l'analyse des résidus.

Dans le cas d'une régression linéaire, il y a plusieurs hypothèses sur lesquelles repose le modèle. Ce sont les hypothèses d'indépendance, de normalité, de linéarité et d'homoscédasticité (σ ne varie pas avec la variable explicative). On peut en général évaluer les deux premières avec le contexte. Concernant les deux autres, on peut visualiser l'ajustement en superposant la droite de régression estimée au nuage de points observés. Avec le package brms, cela donne la **figure 5.7**.

```r
# extrait les valeurs tirées dans les distributions a posteriori des paramètres
post <- as_draws_df(lm.brms)

# crée une grille de x pour tracer l'intervalle de crédibilité
grille_x <- tibble(x = seq(min(data$x), max(data$x), length.out = 100))

# pour chaque x, simule des valeurs de y à partir des échantillons
pred <- post %>%
  select(b_Intercept, b_x) %>%
  expand_grid(grille_x) %>%
  mutate(y = b_Intercept + b_x * x) %>%
  group_by(x) %>%
  summarise(
    mean = mean(y),
    lower = quantile(y, 0.025),
    upper = quantile(y, 0.975),
    .groups = "drop"
  )

# extrait les moyennes a posteriori des paramètres
intercept <- summary(lm.brms)$fixed[1,1]
slope <- summary(lm.brms)$fixed[2,1]

# tracé
ggplot(data, aes(x = x, y = y)) +
  geom_point(alpha = 0.6) +
  geom_ribbon(data = pred, aes(x = x, ymin = lower, ymax = upper), fill =
"blue", alpha = 0.2, inherit.aes = FALSE) +
  geom_line(data = pred, aes(x = x, y = mean), color = "blue", size = 1.2) +
  geom_line(data = true_line, aes(x = x, y = y), color = "red", size = 1.2) +
  labs(x = "x", y = "y") +
  coord_cartesian(xlim = range(grille_x$x)) +
  theme_minimal()
```

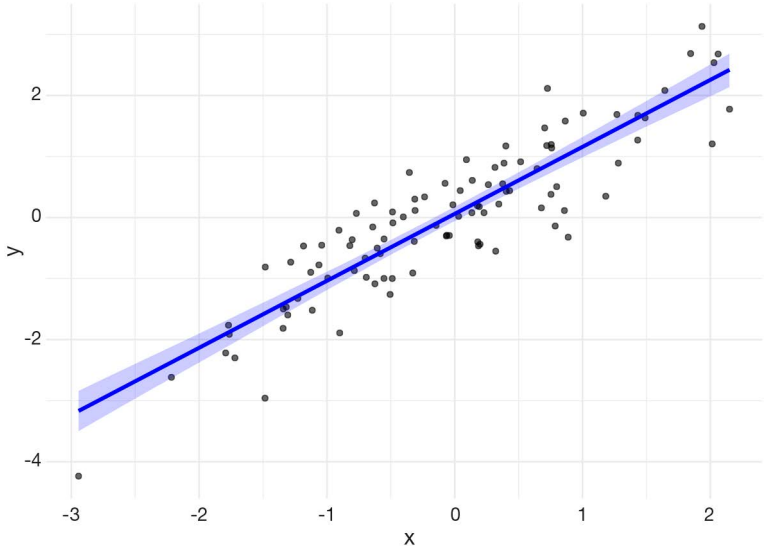

Figure 5.7. Ajustement du modèle linéaire par brms. La droite bleue est la régression estimée, obtenue en fixant l'ordonnée à l'origine et la pente à leur moyenne a posteriori, entourée de son intervalle de crédibilité à 95 %.

Les méthodes bayésiennes sont souvent utilisées pour des modèles plus complexes que la régression linéaire (comme les modèles mixtes, voir chapitre 6), pour lesquels il n'existe pas de tests de qualité d'ajustement standards « clé en main ». Dans ces situations, on utilise couramment ce qu'on appelle des *posterior predictive checks*. L'idée est de simuler de nouveaux jeux de données à partir de la distribution a posteriori des paramètres du modèle, puis de les comparer aux données observées. Plus les données simulées ressemblent aux données réelles, plus cela suggère que le modèle s'ajuste bien. Cette comparaison peut se faire de manière visuelle ou à l'aide d'une *Bayesian p-value* (ou p-valeur bayésienne) qui quantifie l'écart entre données simulées et observées.

Dans brms, il suffit de faire :

```
pp_check(lm.brms)
```

La fonction pp_check() génère des graphiques de *posterior predictive checks* (**figure 5.8**). Elle compare les données observées à des données simulées à partir du modèle ajusté. Si le modèle est bien ajusté aux données, alors on devrait pouvoir l'utiliser pour générer des données qui ressemblent aux données observées. Par conséquent, si les courbes simulées recouvrent bien les observations, cela indique que le modèle capte correctement la structure des données. Dans le cas contraire, cela peut suggérer un problème de spécification du modèle, par exemple un lien ou une famille de distribution inadaptée (voir chapitre 6).

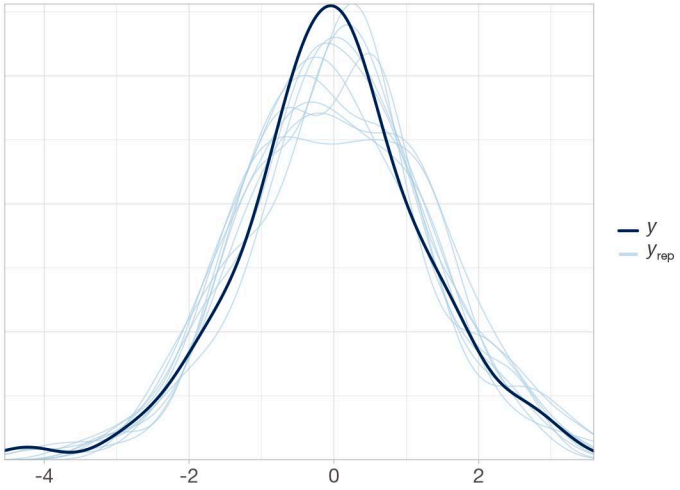

Figure 5.8. *Posterior predictive checks* réalisés avec brms. La courbe noire correspond aux données observées ; les courbes bleues, aux données simulées selon le modèle. L'axe des abscisses représente les valeurs possibles de la variable réponse simulée ou observée. L'axe des ordonnées indique leur densité estimée.

On peut également calculer une *Bayesian p-value* qui représente la proportion de jeux de données simulés sous le modèle et pour lesquels la statistique choisie (ici la moyenne) est aussi grande ou plus grande que celle observée. Une valeur proche de 0 ou de 1 peut indiquer un mauvais ajustement du modèle pour cette statistique particulière, tandis qu'une valeur proche de 0.5 suggère un bon ajustement. Avec brms, cette *Bayesian p-value* s'obtient comme suit :

```
# Extraire les simulations de y_rep
y_rep <- posterior_predict(lm.brms)

# Calculer la statistique de test sur les données simulées (moyenne ici)
T_sim <- rowMeans(y_rep)
```

```
# Calculer la statistique observée
T_obs <- mean(lm.brms$data$y)

# Calculer la Bayesian p-value
bayes_pval <- mean(T_sim >= T_obs)

# Afficher le résultat
bayes_pval
#> [1] 0.5
```

5.3. LA COMPARAISON DE MODÈLES

Comme on l'a vu dans le chapitre 1, la statistique bayésienne permet de comparer plusieurs hypothèses entre elles et de savoir à quel point une hypothèse est plausible à partir des données collectées.

Il est essentiel, avant de comparer des modèles, de se demander quel est l'objectif de l'analyse : s'agit-il de mieux comprendre un phénomène (approche explicative) ou plutôt de faire des prédictions (approche prédictive) ?

Une stratégie consiste à construire un modèle unique incluant les variables jugées pertinentes, puis à l'ajuster, l'examiner, le tester et l'améliorer progressivement. Cette approche vise moins à identifier le meilleur modèle qu'à explorer différentes variantes pour mieux comprendre le système étudié.

Pour évaluer la capacité prédictive d'un modèle, on peut s'appuyer sur des données déjà utilisées pour l'ajustement (prédiction interne) ou, de manière plus fiable, sur de nouvelles données (prédiction externe). Cette dernière approche nécessite toutefois de diviser les données en un jeu d'apprentissage et un jeu de test. À défaut, il est possible d'estimer les performances prédictives sur les données d'apprentissage elles-mêmes à l'aide d'outils comme le WAIC ou le LOO-CV.

Le WAIC (Watanabe-Akaike Information Criterion) et le LOO-CV (Leave-One-Out Cross-Validation) permettent de comparer des modèles en estimant leur capacité à prédire de nouvelles données. Ils combinent l'ajustement aux données observées avec une pénalisation de la complexité du modèle. Une valeur de WAIC ou de LOO-CV plus faible indique un meilleur modèle. Le WAIC est basé sur une approximation théorique, tandis que le LOO-CV repose sur une validation croisée. Le LOO-CV est généralement plus précis, surtout pour les modèles complexes ou les jeux de données de taille limitée, mais il est aussi plus coûteux en calcul. En pratique, lorsque les modèles sont bien spécifiés et que l'échantillon est grand, WAIC et LOO-CV donnent souvent des résultats très proches pour un même modèle.

Je reviens à l'exemple de la régression linéaire. On aimerait tester l'hypothèse que la variable *x* explique bien une part importante de la variation dans *y*. Cela revient à comparer les modèles avec et sans cette variable.

Dans brms, on ajuste ces deux modèles avec des priors faiblement informatifs :

```
# Modèle avec covariable
fit1 <- brm(y ~ x, data = data, family = gaussian(),
            prior = c(
              prior(normal(0, 1.5), class = Intercept),
              prior(normal(0, 1.5), class = b),
              prior(exponential(1), class = sigma)
            ))

# Modèle sans covariable
fit0 <- brm(y ~ 1, data = data, family = gaussian(),
            prior = c(
              prior(normal(0, 1.5), class = Intercept),
              prior(exponential(1), class = sigma))
```

La fonction waic() permet d'extraire le WAIC, où le modèle avec la plus petite valeur est préféré. Si le modèle avec x est bien le bon (c'est ce qu'on attend puisque c'est comme ça que les données ont été simulées), on devrait voir qu'il est nettement meilleur que celui sans covariable :

```
# Calcul du WAIC pour chaque modèle
waic1 <- waic(fit1)
waic0 <- waic(fit0)

# Comparaison
waic1$estimates['waic',]
#>  Estimate        SE
#> 172.43145  13.14333
waic0$estimates['waic',]
#>  Estimate        SE
#> 334.03145  17.13167
```

Ouf, c'est bien le cas. La fonction loo() permet de calculer le LOO-CV (une approximation en fait) :

```
# Leave-one-out cross-validation
loo1 <- loo(fit1)
loo0 <- loo(fit0)

# Comparaison
loo_compare(loo0, loo1)
#>      elpd_diff se_diff
#> fit1   0.0       0.0
#> fit0 -80.8       9.1
```

Dans cette sortie R, elpd_diff donne l'écart de LOO-CV entre chaque modèle et celui qui a la plus grande valeur. Ainsi, le meilleur modèle est sur la première ligne avec un elpd_diff égal à zéro ; ici, c'est le modèle avec la covariable. On arrive donc à la même conclusion qu'avec le WAIC.

À RETENIR

- La régression linéaire permet de modéliser la relation entre une variable réponse continue et une ou plusieurs variables explicatives, en tenant compte d'une variabilité résiduelle.
- Simuler des données à partir d'un modèle est un excellent moyen de comprendre son fonctionnement et de tester son code.
- Les lois a priori faiblement informatives (comme $N(0, 1.5)$ pour les coefficients ou Exp(1) pour σ) aident à encadrer les valeurs réalistes tout en laissant au modèle la liberté d'apprendre des données.
- La validation et la comparaison des modèles peuvent se faire à l'aide de *posterior predictive checks* et de critères comme le WAIC. Ces outils permettent d'évaluer la qualité du modèle au regard des données et d'arbitrer entre plusieurs modèles concurrents.

6. Modèles linéaires généralisés et modèles généralisés mixtes

Ce chapitre présente l'application de la statistique bayésienne à des extensions du modèle linéaire vu au chapitre précédent, les modèles linéaires généralisés (GLM) et les modèles linéaires généralisés mixtes (GLMM). On commencera par un GLM qui nous permettra de revisiter notre exemple fil rouge sur la survie des ragondins et les données binaires. Nous utiliserons ensuite un GLMM pour analyser des données de comptage. Nous utiliserons brms et comparerons avec l'approche fréquentiste[10].

6.1. MODÈLES LINÉAIRES GÉNÉRALISÉS (GLM)

Dans le chapitre 5, on a introduit la régression linéaire $y_i \sim N(\mu_i, \sigma^2)$ avec $\mu_i = \beta_0 + \beta_1 x_i$, où on modélise la moyenne μ de la variable réponse y en fonction d'une variable explicative x. Ce modèle dit linéaire est bien adapté à une variable réponse continue. Mais que se passe-t-il lorsque la variable réponse est discrète ? Revenons à notre exemple sur le ragondin dans lequel on étudie le nombre d'animaux qui survivent. Si l'on applique la régression linéaire sur ces données, on va obtenir un nombre décimal de ragondins, ce qui est un peu embêtant pour un effectif par définition discret. De plus, si l'on introduit une variable explicative x_i comme la masse pour expliquer les variations dans le nombre de ragondins qui survivent, on peut se retrouver avec une probabilité de survie négative, ou plus grande que 1. Pourquoi ? Et bien, car rien n'oblige le modèle linéaire à ne considérer que des valeurs positives et plus petites que 1.

On a vu au chapitre 1 la solution. On note $z_i = 1$ quand le ragondin i a survécu, et $z_i = 0$ sinon, et on suppose que l'événement de survie est comme une expérience de pile ou face avec une probabilité θ, autrement dit chaque z_i suit une Bernoulli de paramètre θ. Si l'on suppose que les individus sont indépendants et ont la même distribution, alors le nombre total de ragondins qui survivent à l'hiver $\sum_{i=1}^{n} z_i = y$ suit une binomiale $y \sim \text{Bin}(n, \theta)$ avec θ la probabilité de survie.

On a aussi vu aux chapitres 2 et 3 que l'on pouvait utiliser la fonction logit pour forcer un paramètre à être bien estimé entre 0 et 1. Il s'agit d'écrire que $\text{logit}(\theta_i) = \beta_0 + \beta_1 x_i$, comme expliqué dans la **figure 6.1**.

Pour formaliser un peu, on a :

$$z_i \sim \text{Bernoulli}(\theta_i) \quad \text{[vraisemblance]}$$
$$\text{logit}(\theta_i) = \beta_0 + \beta_1 x_i \quad \text{[relation linéaire]}$$
$$\theta_i = \text{logit}^{-1}(\beta_0 + \beta_1 x_i) = \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}} \quad \text{[relation transformée]}$$
$$\beta_0, \beta_1 \sim \text{Normale}(0,1.5) \quad \text{[prior sur les paramètres]}$$

Pour illustrer tout ça, on peut reprendre les données ragondins, auxquelles on ajoute des données de masse des individus et, pour ce faire, on recrée les données brutes, c'est-à-dire les z_i :

```
# Nombre total de ragondins suivis, survivants
n <- 57
y <- 19

# Créer les données individuelles (0 = mort, 1 = vivant)
z <- c(rep(1, y), rep(0, n - y))
```

10 Pour NIMBLE, rendez-vous en ligne : https://oliviergimenez.github.io/statistique-bayes/index.html

```
# Ajouter une covariable continue (ex : masse)
set.seed(123)
masse <- rnorm(n, mean = 5, sd = 1)  # masse simulée en kg

df_bern <- data.frame(survie = z, masse = masse)
```

On peut maintenant ajuster les deux modèles avec brms par exemple, la régression linéaire et la régression logistique :

```
# Ajustement de la régression linéaire
fit_lm <- brm(survie ~ masse,
              data = df_bern,
              family = gaussian())

# Ajustement de la régression logistique
fit_logit <- brm(survie ~ masse,
                 data = df_bern,
                 family = bernoulli())
```

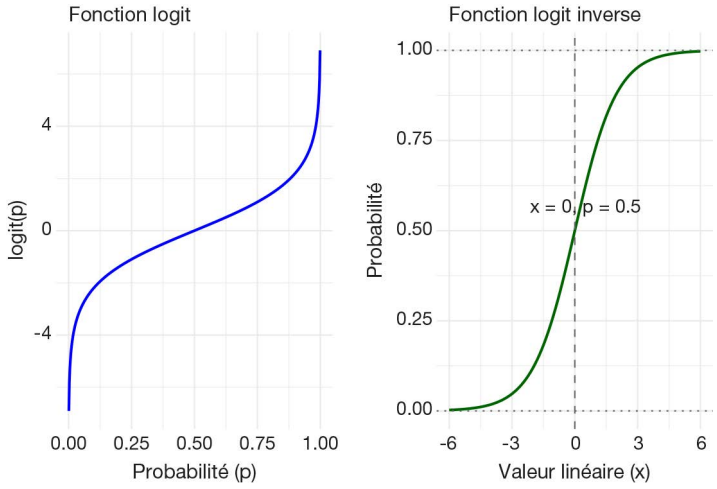

Figure 6.1. La fonction logit et son inverse reliant probabilité et variable. À gauche : la fonction logit transforme une probabilité p en une valeur continue non bornée logit(p) qui vit entre moins l'infini et plus l'infini. À droite : la fonction logit inverse transforme une combinaison linéaire de prédicteurs (« valeur linéaire » sur la figure) en probabilité qui vit entre 0 et 1. La fonction logit est utilisée dans la régression logistique (GLM avec distribution binomiale) pour transformer une probabilité (entre 0 et 1) en une variable continue définie sur l'ensemble des réels. Puis, la fonction logit inverse permet ensuite de revenir à l'échelle des probabilités.

Au passage, l'interprétation des coefficients de la régression logistique n'est pas facile. On introduit souvent la notion de rapport des chances pour aider, mais personnellement, ça ne me parle pas plus que ça. Je reviens toujours à une représentation graphique de la relation entre la probabilité de succès (la survie ici) et les variables explicatives (la masse ici), comme dans la figure 6.3. On observe ici une tendance positive, mais elle est due uniquement au hasard de la simulation (puisque les données ont été générées sans effet de la masse). Une autre façon intuitive de s'en sortir est d'utiliser la règle du 4, proposée par Andrew Gelman et ses collègues (2013). L'astuce consiste à diviser la pente de la régression logistique par 4. Cela donne une estimation approximative du changement de probabilité attendu pour une variation d'une unité de la variable explicative, au point où la courbe est la plus pentue. Si la pente est estimée à 0.23 par exemple, alors la pente maximale de la courbe logistique (autour du point d'inflexion, là où elle change de

forme) est approximativement de 0.23/4 = 0.06. Cela signifie qu'une augmentation d'une unité de la variable explicative (ici, la masse du ragondin augmente de 1 kg) augmente la probabilité de survie d'environ 6 %, au point où la pente est la plus forte (on passe d'une probabilité de survie de 0.5 à 0.53), comme illustré dans la **figure 6.2**.

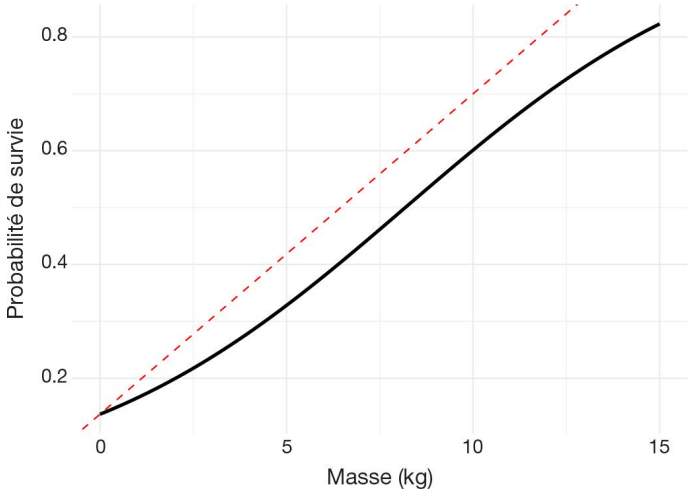

Figure 6.2. Illustration de la règle du 4 de Gelman. Ici, on approxime l'effet de la masse du ragondin sur la probabilité de survie (la courbe logistique en noir) autour du point d'inflexion par une droite dont la pente est donnée par le coefficient estimé divisé par 4 (la droite en tirets rouges).

Mais je m'égare, revenons au problème de la régression linéaire appliquée à des données binaires. Comme on peut le voir dans la **figure 6.3**, la régression linéaire consiste à faire passer une droite sans borne dans les données binaires, ce qui peut conduire à des survies plus grandes

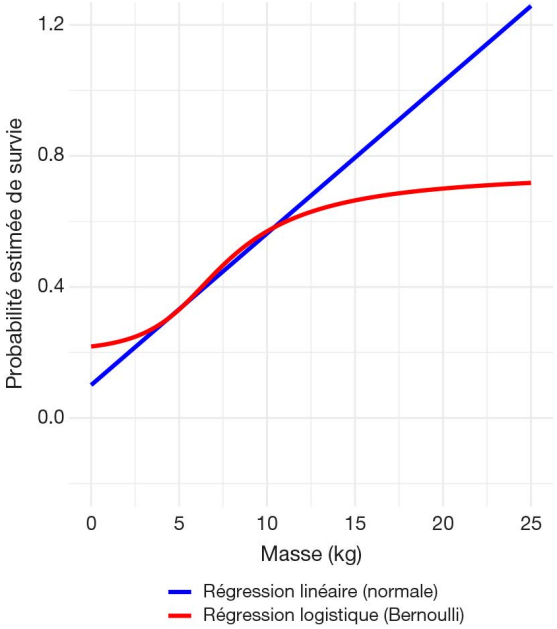

Figure 6.3. Comparaison entre une régression linéaire et une régression logistique, ajustées sur des données binaires. La régression linéaire (en bleu) produit des prédictions plus grandes que 1 (embêtant pour une probabilité de survie), tandis que la régression logistique (en rouge) garantit une estimation de probabilité valide.

que 1 (et/ou plus petites que 0 même si ça n'est pas le cas ici). La régression logistique, en revanche, contraint naturellement les prédictions entre 0 et 1 grâce à la transformation logit, ce qui en fait un choix adapté aux variables de type succès/échec. Au passage, j'ai utilisé la formulation Bernoulli pour introduire une variable explicative mesurée à l'échelle de l'individu, mais si ça n'est pas nécessaire, on peut repasser à la formulation groupée avec la binomiale comme dans les chapitres précédents.

6.2. MODÈLES LINÉAIRES GÉNÉRALISÉS MIXTES (GLMM)

Souvent, les données sont récoltées ou mesurées avec une certaine structure, elles sont hiérarchisées ou groupées, par exemple la relation entre la survie de ragondins et leur masse dans différentes populations de différents bassins-versants. Il est alors pertinent de modéliser cette structure dans les données. Cela permet de mieux expliquer la variabilité dans la survie moyenne qui n'est pas expliquée par la masse, et donc d'obtenir de meilleures estimations. Pour ce faire, on introduit les modèles linéaires généralisés mixtes (GLMM), qui combinent des effets fixes comme dans les GLM, représentant l'effet moyen d'une variable explicative (la masse dans l'exemple des ragondins) et des effets aléatoires représentant la variabilité entre groupes ou niveaux hiérarchiques.

Qu'est-ce qu'un effet aléatoire ? Un effet est aléatoire lorsqu'il représente une sélection aléatoire d'unités dans une population plus vaste, par exemple des sites d'échantillonnage ou des individus ; si l'on devait refaire l'expérience, peu importe les sites ou les individus, l'important est de pouvoir généraliser l'interprétation des effets. En ce sens, le sexe des ragondins par exemple ne peut pas être considéré comme un effet aléatoire ; si l'on refait l'expérience, la variable sexe a toujours les deux mêmes modalités, mâle et femelle. Au contraire, considérer les sites d'une aire d'étude de nos ragondins comme un effet fixe permet seulement de dire des choses sur ces sites précis, sans possibilité de généraliser à la « population » de sites, ou à l'aire d'étude.

Au passage, vous verrez les termes *modèles hiérarchiques*, *multiniveaux* ou *à effets aléatoires* utilisés pour désigner un GLMM dans la littérature scientifique. Parfois, il s'agit de la même chose, parfois il s'agit de GLMM un peu modifiés. Pour éviter les confusions, souvenez-vous que les GLMM sont utilisés pour analyser des données qui viennent avec une structure en groupes.

6.2.1. Exemple

Pour illustrer concrètement un GLMM, imaginez la situation où l'on cherche à estimer l'abondance de ragondins dans le bassin-versant du Lez, à Montpellier, où le Lez est un fleuve qui traverse la ville. On répartit dix transects sur la zone d'étude. Sur chaque transect, on compte le nombre de ragondins présents à dix points espacés régulièrement. On s'intéresse à la réponse du nombre de ragondins (comptages) en fonction de la température. Les mesures sont bien hiérarchisées, on fait une mesure du nombre de ragondins sur chacun des dix points que contient chacun des dix transects. Le protocole est illustré dans la **figure 6.4** et s'inspire du livre de mon collègue Jason Matthiopoulos (2011).

À partir de ce protocole, simulons des données avec le script suivant. On va corser le tout en supposant que sur nos dix transects on a eu des soucis d'échantillonnage sur trois d'entre eux, pour lesquels on n'a pu faire que deux ou trois points :

```
set.seed(123) # pour la reproductibilité
transects <- 10 # nombre total de transects
nb_points <- c(10, 10, 10, 3, 2, 10, 10, 3, 10, 10) # nombre de points par transect
data <- NULL # objet qui stockera les données simulées
for (tr in 1:transects){
  ref <- rnorm(1, 0, .3) # effet aléatoire du transect (N(0,0.3²))
```

```
  # température simulée le long du transect :
  # point de départ aléatoire entre 18 et 22 °C puis légère pente par segment
  t <- runif(1, 18, 22) + runif(1, -0.2, 0.2) * 1:10
  # intensité attendue (échelle log) : relation linéaire avec la température
  ans <- exp(ref + 0.2 * t)
  # comptage Poisson de ragondins pour chaque point
  an <- rpois(nb_points[tr], ans)
  # empile les 10 points du transect courant
  data <- rbind(data, cbind(rep(tr, nb_points[tr]), t[1:nb_points[tr]], an))
}
# on met tout dans un data.frame
sim_simple <- data.frame(
  Transect    = data[, 1],
  Temperature = data[, 2],
  Ragondins    = data[, 3]
)
head(sim_simple)
#>   Transect Temperature Ragondins
#> 1        1    19.78911        54
#> 2        1    19.94232        46
#> 3        1    20.09553        47
#> 4        1    20.24874        60
#> 5        1    20.40194        53
#> 6        1    20.55515        42
```

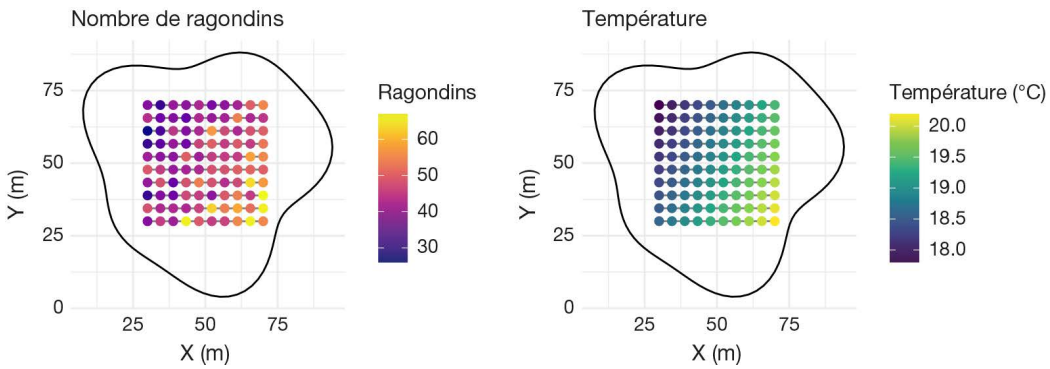

Figure 6.4. Schéma des données sur les ragondins selon un protocole d'échantillonnage avec 10 points dans 10 transects. L'aire d'étude est en noir. À gauche, on a le nombre de ragondins ; à droite, la température.

J'ai commenté le code, ce qui devrait en faciliter la lecture. Malgré tout, quelques explications sur les différentes étapes s'imposent. On commence par une boucle for (tr in 1:transects), qui simule les données pour chacun des dix transects, un par un. À chaque fois, on tire un effet aléatoire spécifique (ref), qui va faire varier un peu l'intercept de la relation entre température et nombre de ragondins selon le transect. Ensuite, on génère une séquence de températures (t) avec un point de départ tiré au hasard et une petite pente qui change légèrement la température d'un point à l'autre. À partir de cette température, on calcule l'intensité attendue du processus de comptage (ans) en supposant une relation linéaire (sur l'échelle log), puis on génère les données observées (an) en tirant des valeurs dans une loi de Poisson de moyenne « ans ». Enfin, on regroupe tout dans un tableau (sim_simple) pour pouvoir ensuite analyser tout ça. Voici la **figure 6.5** qui illustre les données qu'on obtient.

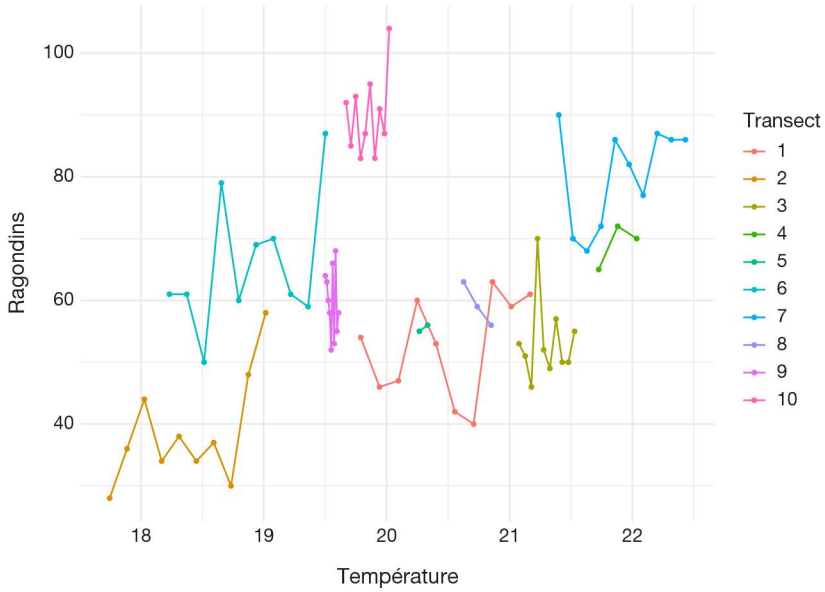

Figure 6.5. Relation entre le nombre de ragondins et la température par transect, avec plusieurs points de comptage (10 pour tous, sauf les transects 4, 5 et 8 pour lesquels on a 3, 2 et 3 points) par transect.

6.2.2. L'approche GLM

On souhaite analyser ces données. On n'a pas affaire à des données binaires ici, comme dans le début de ce chapitre, mais à des données de comptage. Pour modéliser ce type de données, on utilise une distribution de Poisson avec une fonction de lien logarithmique $\log(\theta_i) = \beta_0 + \beta_1 \text{temp}_i$, où temp_i est la température :

$$
\begin{aligned}
y_i &\sim \text{Poisson}(\theta_i) && \text{[vraisemblance]}\\
\log(\theta_i) &= \beta_0 + \beta_1 \text{temp}_i && \text{[relation linéaire]}\\
\theta_i &= e^{\beta_0 + \beta_1 \text{temp}_i} && \text{[relation transformée]}\\
\beta_0, \beta_1 &\sim \text{Normale}(0,1.5) && \text{[prior sur les paramètres]}
\end{aligned}
$$

Cette distribution est relativement facile à manipuler, puisque, entre autres, elle a un seul paramètre θ qui donne le taux d'apparition de l'événement modélisé, et qu'en moyenne le nombre attendu de ragondins ici devrait être égal à ce paramètre.

Dans ce GLM avec distribution de Poisson, les coefficients β_0 et β_1 s'interprètent sur l'échelle logarithmique. Plus précisément, une variation d'une unité de température entraîne une multiplication du nombre moyen de ragondins par $\exp(\beta_1)$. Par exemple, si $\beta_1 = 0.3$, alors une augmentation d'un degré correspond à une hausse attendue d'environ 35 % du nombre moyen de ragondins, puisque $\exp(0.3) \approx 1.35$. On peut aussi visualiser graphiquement la relation entre le nombre de ragondins et la température, comme dans les figures 6.8 et 6.10 à venir.

Dans un premier modèle, oublions la structure en groupes/niveaux dans les données, ici les transects. On fait passer une seule droite dans le nuage de points, c'est le modèle avec *complete pooling* ou regroupement complet :

```
# on n'oublie pas de centrer-réduire la covariable température
sim_simple$Temp <- scale(sim_simple$Temperature)
# modèle avec complete pooling
fit_complete <- brm(Ragondins ~ Temp,
                    data = sim_simple, # données simulées
                    family = poisson("log")) # distribution de Poisson, lien log
```

Les résultats sont les suivants :

```
summary(fit_complete)
#>  Family: poisson
#>   Links: mu = log
#> Formula: Ragondins ~ Temp
#>    Data: sim_simple (Number of observations: 78)
#>   Draws: 2 chains, each with iter = 5000; warmup = 1000; thin = 1;
#>          total post-warmup draws = 8000
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept     4.13      0.01     4.11     4.16 1.00     5374     5177
#> Temp          0.10      0.01     0.07     0.13 1.00     5936     5368
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Ici, on ignore que les données sont mesurées par transect, et on suppose à tort que toutes les observations sont indépendantes. Le risque, c'est de tirer de mauvaises conclusions : on peut croire qu'une seule relation existe, alors que les différences ne sont pas dues à la température, mais aux variations d'un transect à l'autre, ou au contraire on peut passer à côté d'une vraie tendance. Un test d'ajustement permet de voir dans la **figure 6.6** que l'ajustement n'est pas bon.

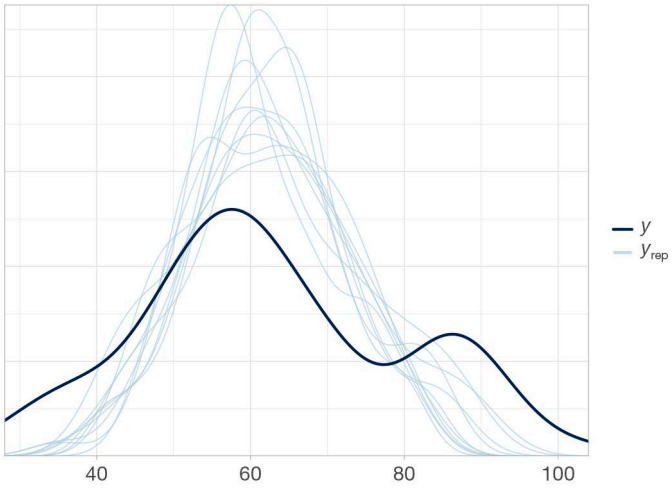

Figure 6.6. Vérification de l'adéquation du modèle avec *complete pooling* ou regroupement complet. L'axe des abscisses représente les valeurs possibles de la variable réponse simulée ou observée. L'axe des ordonnées indique leur densité estimée. Les distributions simulées (en bleu) sont comparées aux données observées (en noir). Le mauvais recouvrement indique une mauvaise adéquation du modèle aux données.

Pour prendre en compte la structuration des données, on peut ajuster un autre modèle dans lequel le transect est traité comme un effet fixe. Autrement dit, on ajuste une droite séparée pour chaque transect, avec son propre intercept, mais avec la même pente :

```
# modèle avec no pooling / pas de regroupement (effet fixe transect)
fit_nopool <- brm(Ragondins ~ Temp + as.factor(Transect),
                  data = sim_simple, # données simulées
                  family = poisson("log")) # distribution de Poisson, lien log
```

Les résultats sont les suivants :

```
summary(fit_nopool)
#>  Family: poisson
#>   Links: mu = log
#> Formula: Ragondins ~ Temp + as.factor(Transect)
#>    Data: sim_simple (Number of observations: 78)
#>   Draws: 2 chains, each with iter = 5000; warmup = 1000; thin = 1;
#>          total post-warmup draws = 8000
#>
#> Regression Coefficients:
#>                 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept           3.90      0.05     3.81     3.99 1.00     3659     4650
#> Temp                0.20      0.06     0.08     0.32 1.00     2684     3611
#> as.factorTransect2  0.04      0.12    -0.21     0.28 1.00     3001     3973
#> as.factorTransect3 -0.12      0.07    -0.26     0.03 1.00     4207     5527
#> as.factorTransect4  0.05      0.11    -0.16     0.26 1.00     3737     4859
#> as.factorTransect5  0.09      0.10    -0.12     0.29 1.00     5687     5484
#> as.factorTransect6  0.49      0.10     0.29     0.68 1.00     3009     4224
#> as.factorTransect7  0.19      0.09     0.02     0.37 1.00     3322     4124
#> as.factorTransect8  0.08      0.09    -0.09     0.26 1.00     5512     5480
#> as.factorTransect9  0.28      0.08     0.13     0.43 1.00     3452     4601
#> as.factorTransect10 0.64      0.06     0.53     0.77 1.00     3729     4715
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Ici, j'ai indiqué à brms de considérer le transect comme une variable catégorielle, un facteur, c'est le as.factor(Transect) dans l'appel à la fonction brm(). Par défaut, le premier niveau du facteur (ici le transect 1) est utilisé comme niveau de référence. Cela signifie que l'intercept β_0 estimé dans le modèle correspond au transect 1, et que les coefficients associés aux autres transects représentent les écarts (sur l'échelle log) par rapport à ce transect 1. Par exemple, on estime β_0 à 3.90, c'est l'intercept pour le transect 1. On estime aussi le décalage entre le transect 1 et le transect 2 à 0.04. Alors l'intercept pour le transect 2 est donné par 3.94. Ce calcul peut être répété pour chaque transect pour obtenir les intercepts spécifiques, que l'on peut ensuite transformer *via* l'exponentielle pour retrouver le nombre moyen attendu de ragondins (sur l'échelle d'origine) pour une température moyenne (qui vaut 0 ici puisqu'on a standardisé la variable température) :

```
# extraire l'intercept (référence = Transect 1)
beta0 <- fixef(fit_nopool)["Intercept", "Estimate"]

# tous les coefficients du modèle
coefs <- fixef(fit_nopool)
# effets associés aux autres transects
coefs_transects <- coefs[grep("as.factor", rownames(coefs)), "Estimate"]

# calcul des intercepts par transect
intercepts_log <- c(
  Transect1 = beta0,
  beta0 + coefs_transects
)

# calcul des nombres moyens attendus de ragondins sur l'échelle d'origine
nombres <- exp(intercepts_log)
```

```
# le tableau qui résume
df_intercepts <- data.frame(
  Transect = names(intercepts_log),
  Intercept_log = round(intercepts_log, 2),
  Nombre = round(nombres, 2)
)

# affichage
df_intercepts
#>                              Transect Intercept_log Nombre
#> Transect1                    Transect1          3.90  49.60
#> as.factorTransect2   as.factorTransect2          3.94  51.48
#> as.factorTransect3   as.factorTransect3          3.79  44.15
#> as.factorTransect4   as.factorTransect4          3.95  51.94
#> as.factorTransect5   as.factorTransect5          3.99  54.02
#> as.factorTransect6   as.factorTransect6          4.39  80.75
#> as.factorTransect7   as.factorTransect7          4.10  60.22
#> as.factorTransect8   as.factorTransect8          3.98  53.74
#> as.factorTransect9   as.factorTransect9          4.19  65.76
#> as.factorTransect10 as.factorTransect10          4.55  94.53
```

On estime bien un intercept pour chaque transect, donc dix intercepts, et la pente, c'est-à-dire l'effet de la température, est la même pour tous les transects. Notez aussi que les valeurs obtenues, Nombre ici, sont des moyennes attendues issues du modèle de Poisson. Ce sont donc des valeurs continues, décimales, bien que les données observées soient des comptages entiers. C'est une caractéristique des modèles de Poisson : la variable à prédire est discrète, mais le modèle s'appuie sur une moyenne continue pour modéliser sa distribution.

La qualité de l'ajustement est meilleure, comme on le voit dans la **figure 6.7**.

Ce modèle *no pooling* fait mieux que le modèle *complete pooling*, comme on peut le voir dans la **figure 6.8**, mais il reste insatisfaisant. L'approche *no pooling* consiste à ajuster un modèle indépendant pour chaque transect, sans partager d'information entre ces groupes. Cela pose deux problèmes : d'une part, on ne peut pas généraliser les résultats obtenus à d'autres transects que ceux observés ; d'autre part, on ignore des informations potentiellement utiles en supposant que chaque transect n'a rien à apprendre des autres. Cette stratégie devient particulièrement inefficace lorsque chaque groupe comporte peu d'observations.

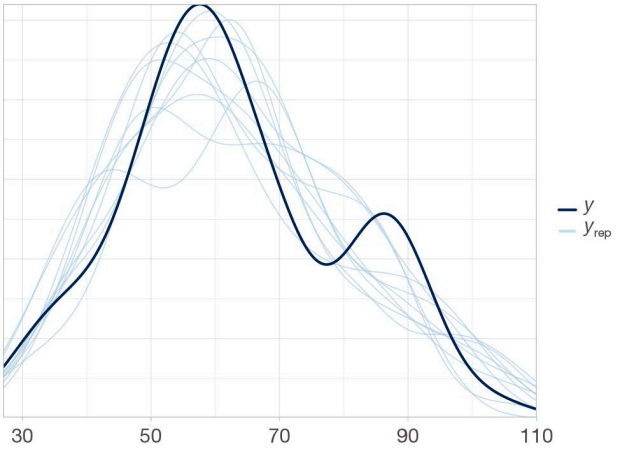

Figure 6.7. Vérification de l'adéquation du modèle avec *no pooling* ou pas de regroupement. L'axe des abscisses représente les valeurs possibles de la variable réponse simulée ou observée. L'axe des ordonnées indique leur densité estimée. Les distributions simulées (en bleu) sont comparées aux données observées (en noir).

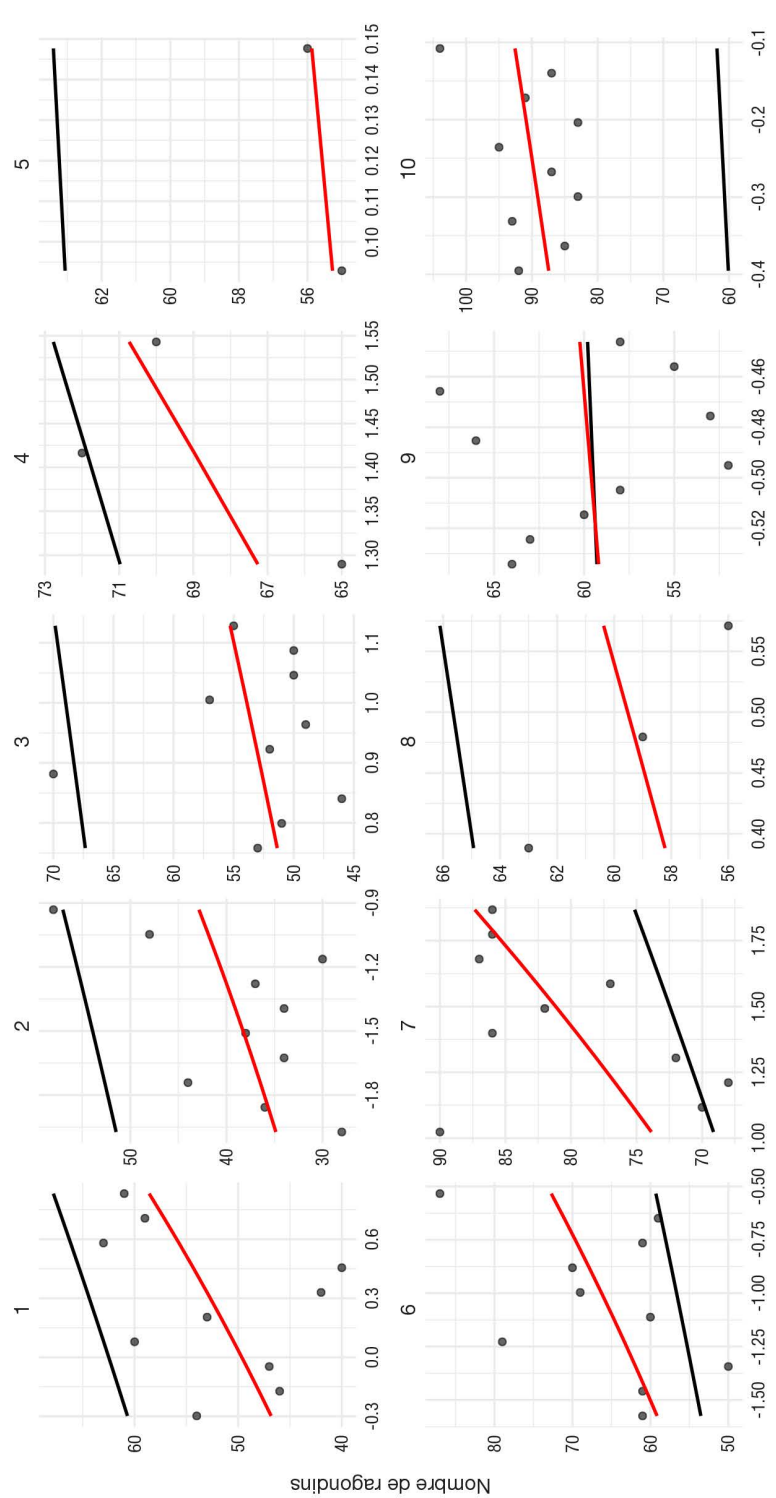

Figure 6.8. Comparaison entre les modèles *complete pooling* (noir) et *no pooling* (rouge) pour prédire le nombre de ragondins en fonction de la température, par transect. Le modèle *no pooling* ajuste une courbe indépendante pour chaque transect, tandis que le *complete pooling* suppose une relation commune.

6.2.3. L'approche GLMM

Revenons à notre objectif : évaluer l'effet de la température sur l'abondance des ragondins, tout en prenant en compte la structure hiérarchique des données (segments imbriqués dans des transects). Jusqu'ici, les modèles *complete pooling* et *no pooling* représentaient deux extrêmes : soit on supposait que tous les transects partageaient exactement la même relation température-abondance, soit on estimait une relation totalement indépendante pour chacun d'eux (*via* un intercept spécifique à chaque transect). Les modèles linéaires généralisés mixtes (GLMM), ou *partial pooling*, permettent un compromis plus réaliste.

On construit un GLMM dans lequel on autorise chaque transect à avoir un intercept propre – c'est-à-dire une abondance moyenne spécifique – mais on suppose que ces intercepts ne sont pas totalement indépendants. On les considère plutôt comme des variations aléatoires autour d'un intercept moyen β_0, issues d'une même distribution normale. Cela revient à dire que nos transects β_{0j} (où j varie de 1 à 10) sont tirés d'une population plus large de transects possibles, dans laquelle l'abondance moyenne varie d'un transect à l'autre. On modélise cette variabilité spatiale à l'aide d'un effet aléatoire, noté ici $\beta_{0j} \sim N(\beta_0, \sigma)$, où σ est la variation entre transects.

Autrement dit, chaque intercept par transect β_{0j} peut s'écrire comme une déviation b_j autour de l'intercept global $\beta_{0j} = \beta_0 + b_j$ avec $b_j \sim N(0, \sigma)$, où β_0 représente l'intercept moyen (pour un transect « typique ») et où σ quantifie la variabilité entre transects. Par exemple, si l'intercept moyen est $\beta_0 = 2$, mais que le transect 4 a un intercept de $\beta_{04} = 3$, on dira que cet effet spécifique $b_4 = 1$ correspond à une abondance plus élevée que la moyenne.

Cette modélisation hiérarchique permet de capturer l'hétérogénéité spatiale tout en partageant l'information entre groupes, ce qui est particulièrement utile lorsque certains transects comportent peu d'observations. On peut aussi voir le modèle *partial pooling* (3 paramètres estimés dans l'exemple ragondin : β_0, β_1, σ) comme un compromis entre le modèle *complete pooling* (2 paramètres estimés dans l'exemple ragondin : β_0, β_1) et le modèle *no pooling* (11 paramètres estimés dans l'exemple ragondin : 10 intercept et 1 pente β_1).

Si on formalise un peu, le GLMM correspondant s'écrit :

$y_i \sim \text{Poisson}(\theta_i)$	[vraisemblance]
$\log(\theta_i) = \beta_{0j} + \beta_1 \text{temp}_i$	[relation linéaire]
$\beta_{0j} \sim \text{Normale}(\beta_0, \sigma)$	[effet aléatoire]
$\beta_0 \sim \text{Normale}(0, 1.5)$	[prior sur l'intercept moyen]
$\sigma \sim \text{Exp}(1)$	[prior pour l'erreur standard de l'effet aléatoire]
$\beta_1 \sim \text{Normale}(0, 1.5)$	[prior pour la pente]

Ajustement du modèle en bayésien avec brms

On ajuste d'abord le GLMM avec *partial pooling* avec brms :

```
# modèle avec partial pooling (effet aléatoire transect)
fit_partial <- brm(Ragondins ~ Temp + (1 | Transect), # relation nombre de
ragondins vs température avec effet aléatoire transect sur l'intercept
                   data = sim_simple, # données simulées
                   family = poisson("log")) # distribution de Poisson, lien log
```

Dans la syntaxe, l'effet aléatoire sur l'intercept est spécifié avec (1 | Transect), où le 1 signifie qu'on travaille sur l'intercept, et qu'il y a un intercept par (c'est le |) transect. Si on voulait ajouter un effet aléatoire sur la pente, on écrirait (1 + Temp | Transect).

Les résultats sont :

```
summary(fit_partial)
#>  Family: poisson
#>   Links: mu = log
```

```
#> Formula: Ragondins ~ Temp + (1 | Transect)
#>    Data: sim_simple (Number of observations: 78)
#>   Draws: 2 chains, each with iter = 5000; warmup = 1000; thin = 1;
#>          total post-warmup draws = 8000
#>
#> Multilevel Hyperparameters:
#> ~Transect (Number of levels: 10)
#>               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> sd(Intercept)     0.27      0.08     0.16     0.47 1.00     2119     3183
#>
#> Regression Coefficients:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept     4.09      0.09     3.92     4.27 1.00     2087     2747
#> Temp          0.17      0.05     0.06     0.27 1.00     3559     4119
#>
#> Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
#> and Tail_ESS are effective sample size measures, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Ce résumé fournit les estimations a posteriori des effets fixes ainsi que des écarts-types des effets aléatoires. La ligne sd(Intercept) donne l'estimation de σ, proche du 0.3 utilisé pour simuler les données (l'intervalle de crédibilité contient la vraie valeur). Les lignes Intercept et Temp donnent les estimations de β_0 et β_1 sur l'échelle log. On verra un peu plus tard comment vérifier que ces estimations sont proches des valeurs utilisées pour simuler les données.

On peut aussi jeter un coup d'œil aux distributions et traces des paramètres (**figure 6.9**).

```
plot(fit_partial)
```

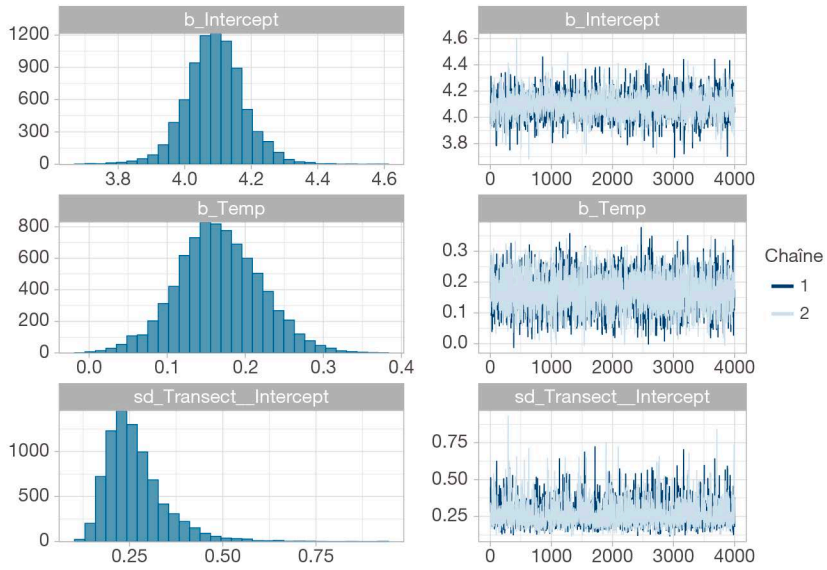

Figure 6.9. Vérification de la convergence des chaînes MCMC pour le modèle avec *partial pooling*. Dans les histogrammes (colonne de gauche), l'axe des abscisses représente les valeurs possibles du paramètre estimé (intercept, pente ou écart-type) et l'axe des ordonnées correspond à leur fréquence dans l'échantillon a posteriori. Dans les *trace plots* (colonne de droite), l'axe des abscisses indique le numéro d'itération du MCMC, tandis que l'axe des ordonnées représente la valeur simulée du paramètre à chaque itération.

On peut alors mettre à jour la figure 6.8 avec la **figure 6.10**.

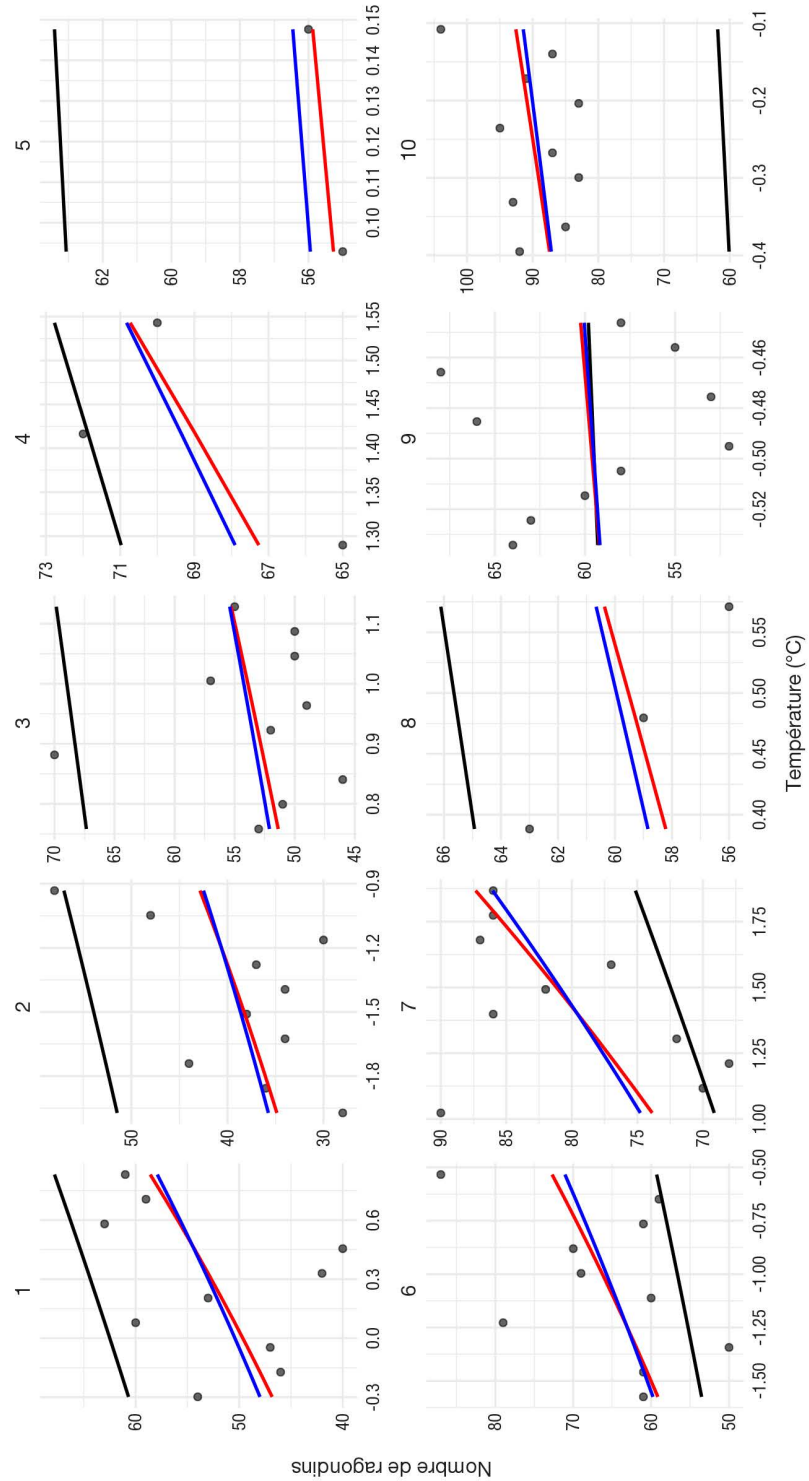

Figure 6.10. Comparaison entre les modèles *complete pooling* (noir), *no pooling* (rouge) et *partial pooling* (bleu) pour prédire le nombre de ragondins en fonction de la température, par transect. Le modèle *no pooling* ajuste une courbe indépendante pour chaque transect, le *complete pooling* suppose une relation commune, tandis que le *partial pooling* fournit un compromis grâce à l'effet aléatoire transect.

On voit que l'ajustement fourni par le *partial pooling* est très similaire au *no pooling*, et bien meilleur que le *complete pooling*. Il y a une petite différence pour les transects avec peu de points d'échantillonnage, les transects 4, 5 et 8, pour lesquels le *partial pooling* se rapproche du *complete pooling*. En l'absence de plus d'information (de données) pour ces transects, il est plutôt raisonnable que les estimations se rapprochent de la moyenne donnée par le modèle avec *complete pooling* plutôt que vers des valeurs extrêmes, atypiques. Rappelez-vous que dans un GLMM, les niveaux de l'effet aléatoire sont caractéristiques d'une population avec une moyenne commune. C'est ce partage d'information entre les transects avec dix points d'échantillonnage et ceux avec deux ou trois points, on parle de *borrowing strength* qui permet d'atténuer, de tamponner, on parle de *shrinkage*, la tendance à surajuster du modèle avec *no pooling* ou à effet fixe, et en ce sens on parle parfois de *regularization*.

La qualité de l'ajustement est validée comme on le voit dans la **figure 6.11**.

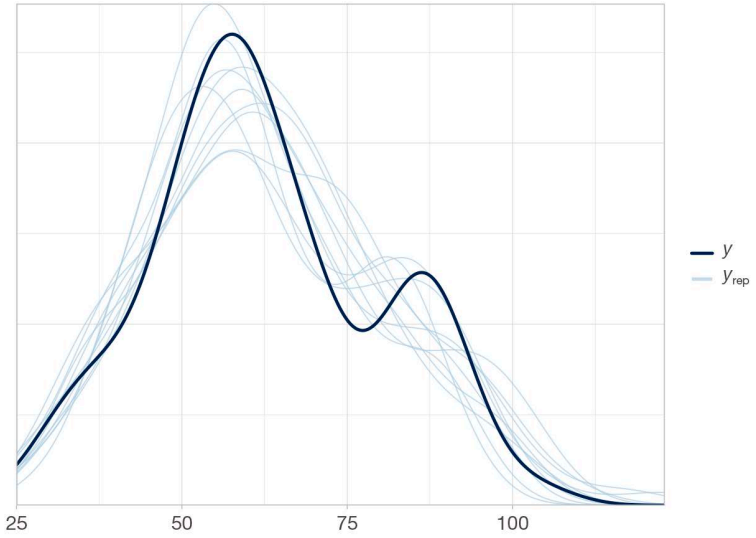

Figure 6.11. Vérification de l'adéquation du modèle avec *partial pooling* ou regroupement partiel. L'axe des abscisses représente les valeurs possibles de la variable réponse simulée ou observée. L'axe des ordonnées indique leur densité estimée. Les distributions simulées (en bleu) sont comparées aux données observées (en noir).

Lorsqu'on ajuste un modèle sur une variable standardisée, comme ici la température centrée-réduite, les coefficients estimés (β_0, β_1) s'interprètent sur cette échelle modifiée : β_1 représente l'effet d'un écart-type de température, et β_0 correspond à la valeur attendue quand la température standardisée est 0, c'est-à-dire à la température moyenne. On a souvent envie d'exprimer les effets sur des unités compréhensibles, ici les degrés Celsius, plutôt qu'en écart-type de température, qui est plus abstrait. Pour revenir à une interprétation sur l'échelle réelle (en degré Celsius donc), les coefficients estimés sur la température centrée-réduite peuvent être transformés pour revenir à l'échelle réelle à l'aide des formules suivantes :

$$\beta_1^{\text{réel}} = \frac{\beta_1^{\text{standardisé}}}{\text{écart-type de la température}}$$

$$\beta_0^{\text{réel}} = \beta_0^{\text{standardisé}} - \beta_1^{\text{standardisé}} \times \frac{\text{moyenne de la température}}{\text{écart-type de la température}}$$

Dans R, vous pouvez utiliser le code suivant :

```
# récupère valeurs simulées dans distributions a posteriori des paramètres
post <- as_draws_matrix(fit_partial)
sbzero <- post[, "b_Intercept"]
sbun <- post[, "b_Temp"]

# récupère moyenne et écart-type de la température
mu <- attr(scale(sim_simple$Temperature), "scaled:center")
sg <- attr(scale(sim_simple$Temperature), "scaled:scale")

# convertit les coefficients standardisés
bun   <- sbun / sg # beta1 réel
bzero <- sbzero - sbun * mu / sg # beta0 réel
```

On peut alors visualiser les coefficients sur l'échelle originale et comparer à la valeur utilisée pour simuler les données comme dans les **figures 6.12** et **6.13**.

```
tibble(b0 = bzero) %>%
  ggplot(aes(x = b0)) +
  geom_histogram(color = "white", fill = "skyblue", bins = 30) +
  geom_vline(xintercept = 0, color = "red", linewidth = 1.2) +
  labs(
    x = expression(beta[0]),
    y = "Fréquence"
  ) +
  theme_minimal()
```

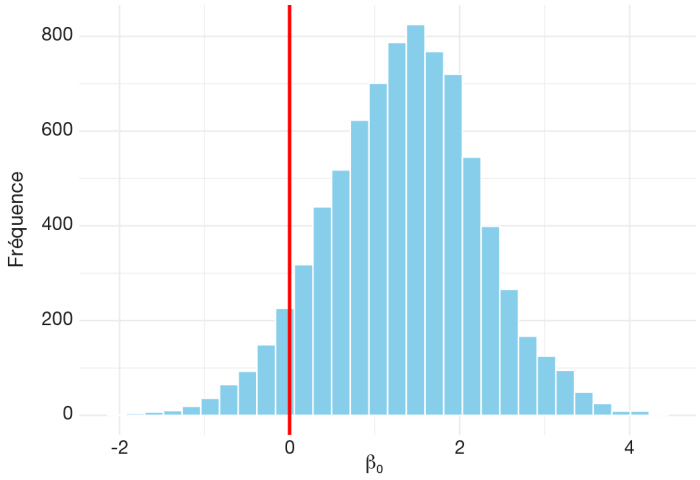

Figure 6.12. Distribution a posteriori de l'intercept moyen (échelle réelle). La ligne rouge indique la vraie valeur (0).

```
tibble(b1 = bun) %>%
  ggplot(aes(x = b1)) +
  geom_histogram(color = "white", fill = "skyblue", bins = 30) +
  geom_vline(xintercept = 0.2, color = "red", linewidth = 1.2) +
  labs(
    x = expression(beta[1]),
    y = "Fréquence"
  ) +
  theme_minimal()
```

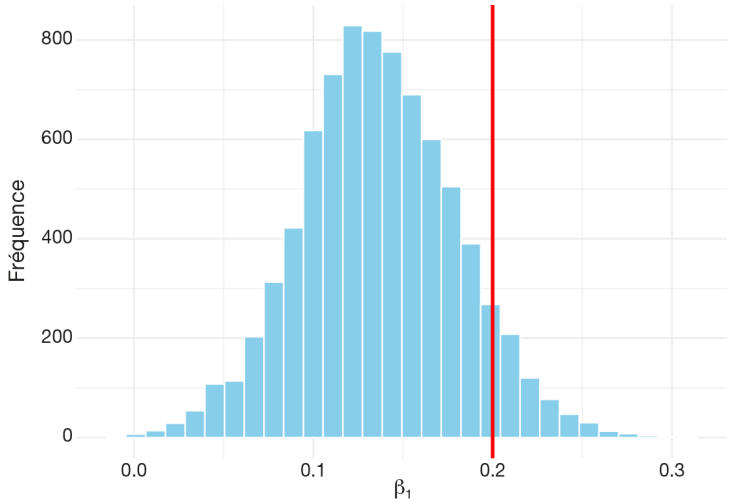

Figure 6.13. Distribution a posteriori de l'effet de la température (échelle réelle). La ligne rouge indique la vraie valeur (0.2).

On retrouve les paramètres ayant servi à simuler les données (en rouge). Il ne s'agit que d'une seule simulation, c'est donc normal qu'en moyenne on n'obtienne pas exactement la même chose (c'est-à-dire que le trait rouge ne coïncide pas plus à la plus grande valeur de l'histogramme), c'est seulement en répétant l'expérience de simulations un grand nombre de fois que cela se produirait.

En bonus, comparons les modèles avec et sans effet de la température. Cela permet de tester la pertinence de la température comme variable explicative :

```
# modèle avec partial pooling (effet aléatoire transect)
fit_partial2 <- brm(Ragondins ~ 1 + (1 | Transect), # un intercept moyen, pas
d'effet de la température, avec l'effet aléatoire transect
                    data = sim_simple, # données simulées
                    family = poisson("log")) # distribution de Poisson, lien log
```

On calcule le WAIC pour chaque modèle et on les compare :

```
waic1 <- waic(fit_partial)
waic2 <- waic(fit_partial2)
tibble(
  Modèle = c("Avec température", "Sans température"),
  WAIC = c(waic1$estimates["waic", "Estimate"],
           waic2$estimates["waic", "Estimate"])
)
#> # A tibble : 2 × 2
#>   Modèle            WAIC
#>   <chr>            <dbl>
#> 1 Avec température  542.
#> 2 Sans température  551.
```

En conclusion, le modèle incluant la température offre un meilleur ajustement aux données selon le critère WAIC. Et fort heureusement puisque l'on a simulé selon ce modèle!

Ajustement du modèle en fréquentiste avec lme4

Pour clore ce chapitre, je vous propose de faire la même analyse avec le package lme4 en fréquentiste.

On charge ledit package :

```
library(lme4)
```

Puis, on applique le GLMM. Notez que la syntaxe de brms est inspirée de celle utilisée dans lme4 :

```
fit_lme4 <- glmer(
  Ragondins ~ Temp + (1 | Transect), # formule complète
  data   = sim_simple,         # jeu de données simulé précédemment
  family = poisson      # distribution adaptée à un comptage
)
```

Voici les résultats :

```
summary(fit_lme4)
#> Generalized linear mixed model fit by maximum likelihood (Laplace
#>   Approximation) [glmerMod]
#>  Family: poisson  ( log )
#> Formula: Ragondins ~ Temp + (1 | Transect)
#>    Data: sim_simple
#>
#>       AIC       BIC    logLik -2*log(L)  df.resid
#>     568.3     575.3    -281.1     562.3        75
#>
#> Scaled residuals:
#>     Min      1Q  Median      3Q     Max
#> -1.9501 -0.6223 -0.1098  0.4779  2.3897
#>
#> Random effects:
#>  Groups   Name        Variance Std.Dev.
#>  Transect (Intercept) 0.04402  0.2098
#> Number of obs: 78, groups:  Transect, 10
#>
#> Fixed effects:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  4.08804    0.06898  59.266  < 2e-16 ***
#> Temp         0.15797    0.04863   3.248  0.00116 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Correlation of Fixed Effects:
#>      (Intr)
#> Temp -0.106
```

Comment lire les sorties ?

Élément	Signification
(Intercept)	Moyenne de ragondins pour un transect moyen, à température moyenne (échelle log).
Temp	Effet linéaire de la température.
Random effects	Écart-type (Std.Dev) de l'intercept aléatoire.

On peut noter que les estimations des paramètres obtenus sont très proches des estimations obtenues avec brms.

À RETENIR

- Les modèles linéaires généralisés (GLM) permettent d'étendre les modèles linéaires aux situations où l'on ne peut pas supposer une erreur normale.
- L'idée générale est d'utiliser une distribution adaptée à la variable réponse – Bernoulli ou binomiale pour les variables binaires (0/1), Poisson pour les comptages (0, 1, 2, etc.) – et de relier la moyenne de cette distribution aux variables explicatives par une fonction de lien (comme logit ou log).
- L'introduction d'effets aléatoires permet de modéliser des groupes hiérarchiques dans les données (par exemple des sites, individus ou transects), en tenant compte de leur hétérogénéité tout en partageant l'information entre eux.
- Les modèles linéaires généralisés mixtes (GLMM) permettent d'estimer simultanément des effets fixes (valables pour toute la population) et des effets aléatoires (propres à chaque groupe, mais supposés tirés d'une distribution commune).
- Dans un modèle à *complete pooling*, on ignore la structure en groupes : on suppose que toutes les données suivent exactement la même relation avec les variables explicatives. Cela peut mener à des conclusions biaisées si les groupes diffèrent réellement. Dans un modèle à *no pooling*, on estime une relation distincte pour chaque groupe, sans partage d'information. Cela produit des estimations très variables, surtout si certains groupes ont des tailles d'échantillon faibles. Les modèles à *partial pooling*, ou GLMM, ou modèles hiérarchiques, représentent un compromis entre ces deux extrêmes : les groupes ont leurs propres paramètres, mais ceux-ci sont liés *via* une distribution commune. Cela permet d'améliorer la stabilité des estimations tout en respectant les différences entre groupes.

CONCLUSIONS

CE QUE L'ON A VU

J'espère, avec ce livre, avoir démythifié (un peu) la statistique bayésienne et les méthodes MCMC. J'espère aussi vous avoir donné les clés pour comprendre la différence entre les approches fréquentiste et bayésienne, aidé à mieux lire la section Méthodes des articles recourant à la statistique bayésienne et à acquérir une certaine autonomie dans la conduite d'analyses bayésiennes.

Tout au long du livre, nous avons abordé plusieurs étapes essentielles. Nous avons commencé par explorer les motivations qui justifient le recours à l'approche bayésienne. Nous avons ensuite introduit le théorème de Bayes et discuté de son interprétation. Nous avons découvert les méthodes de Monte Carlo par chaînes de Markov (MCMC), puis manipulé brms pour ajuster des modèles complexes. Une attention particulière a été portée au rôle des distributions a priori, qu'elles soient non informatives ou informatives, ainsi qu'à l'utilisation de ces approches dans des études de cas autour des GLM et GLMM.

LA STATISTIQUE BAYÉSIENNE, EN RÉSUMÉ

L'approche bayésienne offre de nombreux atouts. Elle permet de quantifier l'incertitude de manière cohérente à l'aide de la probabilité, elle autorise l'intégration explicite de connaissances a priori, et elle rend possible l'ajustement de modèles complexes *via* MCMC. De plus, les intervalles de crédibilité bayésiens sont plus intuitifs que les intervalles de confiance fréquentistes.

Certaines précautions sont toutefois de mise. La vérification de la convergence des chaînes MCMC est une étape cruciale, mais parfois laborieuse. Le choix des distributions a priori nécessite de prendre certaines précautions. L'adéquation du modèle aux données doit être systématiquement évaluée. Enfin, le coût computationnel n'est pas négligeable, en particulier pour les modèles les plus complexes et/ou les gros jeux de données.

QUELQUES CONSEILS

Avant de terminer, je voudrais vous laisser avec quelques conseils inspirés de ma propre expérience. Ces conseils ne sont pas forcément spécifiques à la statistique bayésienne, et ils valent ce qu'ils valent.

Tout d'abord, prenez le temps de formuler clairement votre question. Cela paraît évident, mais cette étape est essentielle pour rester sur la bonne voie et faire les bons choix, par exemple celui de n'utiliser qu'un sous-ensemble des données pour répondre à une question spécifique.

Ensuite, réfléchissez d'abord à votre modèle, à le formaliser soit avec des équations, soit en le dessinant, soit avec des mots. Quelle est la nature de vos données, et donc, si vous êtes dans un cadre de régression, quelle famille de distributions utiliser, comme on l'a vu dans les chapitres 5 (normale) et 6 (Bernoulli/binomiale et Poisson) ? Ne vous précipitez pas sur le clavier. Vérifiez que vous le comprenez en l'expliquant par exemple à des collègues.

À ce sujet, pensez à faire des simulations. Simuler des données à partir de votre modèle permet souvent de mieux le comprendre, comme dans les chapitres 5 et 6. C'est une excellente manière de tester vos hypothèses et de diagnostiquer d'éventuels problèmes.

Choisissez l'environnement R dans lequel vous êtes à l'aise ; j'ai illustré brms (chapitre 2), mais d'autres solutions existent (par exemple NIMBLE[11]).

11 Comme expliqué dans la version enrichie de ce livre disponible en ligne à https://oliviergimenez.github.io/statistique-bayes/

Lors de l'ajustement du modèle, commencez simple. Un modèle avec tous les paramètres constants est une bonne base. Cela permet de s'assurer que les données sont bien lues et formatées, qu'il n'y a pas de données aberrantes (un zéro en trop, une virgule mal placée) ou que les a priori ne génèrent pas des comportements atypiques (voir chapitre 4). Cette approche est particulièrement importante pour la statistique bayésienne pour s'assurer de bonnes performances et de la convergence de l'algorithme MCMC (chapitre 2), tout en se faisant une idée du temps nécessaire à faire tourner l'analyse. Une fois que tous les voyants sont au vert, ajoutez de la complexité progressivement, des effets aléatoires par exemple (chapitre 6), jusqu'à parvenir à la structure de modèle qui vous semble la plus adaptée pour répondre à votre question. Cela sous-entend sûrement que vous devrez faire plusieurs itérations des différentes étapes d'ajustement, de comparaison et de validation de vos modèles (chapitres 5 et 6).

Pour approfondir ces aspects pratiques, je recommande la lecture des articles de Gimenez *et al.* (2025) et de Gelman *et al.* (2020).

POUR FINIR

Adoptez une approche pragmatique. Le choix de l'approche statistique (fréquentiste ou bayésienne) dépend de vos objectifs, qu'il s'agisse de la rapidité, de la complexité du modèle ou du type d'incertitude que vous souhaitez quantifier. Discutez de vos options avec des collègues plus expérimentés si besoin. Le bayésien n'est pas un dogme : c'est un outil puissant parmi d'autres dans votre boîte à outils.

N'hésitez pas à m'écrire si vous avez des questions ou si vous voudriez voir un aspect particulier développé dans une nouvelle édition de cet ouvrage. Et bonne découverte de la statistique bayésienne !

BIBLIOGRAPHIE COMMENTÉE

Albert J., 2009. *Bayesian computation with R*. 2de éd., Springer, New York, 300 p.

Biobayes collectif, 2015. *Initiation à la statistique bayésienne. Bases théoriques et applications en alimentation, environnement, epidémiologie et génétique*. Éditions Ellipses, 360 p.

Si vous cherchez une première approche en français, claire et illustrée par des exemples concrets, ce livre est une très bonne porte d'entrée : https://biobayes.mathnum.inrae.fr/ouvrage

Gelman A., Carlin J.B., Stern H.S., Dunson D.B., Vehtari A., Rubin D.B., 2013. *Bayesian data analysis*. 3e éd., Chapman and Hall/CRC, 677 p.

L'ouvrage de référence pour celles et ceux qui souhaitent acquérir une compréhension théorique et appliquée solide de la statistique bayésienne : https://sites.stat.columbia.edu/gelman/book/

Gelman A., Vehtari A., Simpson D., Margossian C.C., Carpenter B., Yao Y., *et al*., 2020. Bayesian workflow. arXiv:2011.01808, 77 p.

Gimenez O., Royle A., Kéry M., Nater C.R., 2025. Ten quick tips to get you started with Bayesian statistics. *PLOS Computational Biology*, 21(4):1-13.

Johnson A.A., Ott M.Q., Dogucu M., 2022. *Bayes rules!: An introduction to applied Bayesian modeling*. 1re éd., Chapman & Hall/CRC, 544 p.

Un livre très pédagogique pour découvrir les principes et les applications de la statistique bayésienne de manière intuitive et progressive : https://www.bayesrulesbook.com/

Kéry M., Kellner K.F., 2010. *Applied statistical modelling for ecologists: A practical guide to Bayesian and likelihood inference using R, JAGS, NIMBLE, Stan and TMB*. 1re éd., Elsevier, 550 p.

Un manuel pratique pour apprendre à modéliser avec les principaux outils bayésiens dans R (JAGS, NIMBLE, Stan ou TMB), à partir d'exemples écologiques concrets et de comparaisons des résultats : https://www.elsevier.com/books-and-journals/book-companion/9780443137150

Kruschke J.K., 2010. *Doing Bayesian data analysis: A tutorial with R and BUGS*. Academic Press Inc., 672 p.

Un tutoriel approfondi et visuel qui accompagne pas-à-pas l'apprentissage de la statistique bayésienne avec de nombreux exemples pratiques : https://sites.google.com/site/doingbayesiandataanalysis/

Matthiopoulos J., 2011. *How to be a quantitative ecologist: The 'A to R' of green mathematics and statistics*. Chichester, UK, Wiley, 496 p.

McCarthy M.A., 2007. *Bayesian methods for ecology*. Cambridge University Press, 312 p.

Un petit livre vraiment accessible pour comprendre comment appliquer la statistique bayésienne en écologie, sans se noyer dans les mathématiques : https://bit.ly/4jSlfQL

McElreath R., 2020. *Statistical rethinking: A Bayesian course with example in R and Stan*. 2de éd., Chapman & Hall/CRC, 594 p.

Un livre captivant pour apprendre à construire et à interpréter des modèles bayésiens en développant d'abord l'intuition statistique : https://xcelab.net/rm/. Je recommande chaudement le cours en vidéo : https://github.com/rmcelreath/stat_rethinking_2024

REMERCIEMENTS

Merci à mon employeur, le Centre national de la recherche scientifique (CNRS). Chercheur et enseignant-chercheur sont de beaux métiers. Des métiers utiles. On assiste néanmoins à la dégradation des conditions de travail dans le monde académique. Plus de compétition, plus de précarité, moins de postes pérennes. J'ai la chance d'évoluer dans un environnement bienveillant, le Centre d'écologie fonctionnelle et évolutive (CEFE), dont les employés résistent en cultivant le collectif et les collaborations. Merci à eux.

Mon intérêt pour la statistique bayésienne remonte à mes années en Angleterre et en Écosse dans le cadre d'un postdoctorat. Merci à Byron Morgan de m'avoir laissé la liberté d'explorer cette voie qui n'était pas encore à la mode. Merci à Ruth King pour nos échanges et ma première expérience d'écriture d'un livre, et à Steve Brooks pour les séances de remue-méninges. C'est avec Byron, Ruth et Steve que nous avons organisé les premières formations (ou workshops) à la statistique bayésienne pour l'écologie. Je remercie également les collègues pour le matériel qu'ils ont mis à disposition et dont je me suis inspiré pour écrire ce livre.

Je remercie les étudiants et étudiantes de master qui subissent mon enseignement depuis plus de dix ans. Ils ont été mes cobayes et m'ont permis de mûrir (inconsciemment) ce projet de livre. Merci aussi aux doctorantes et doctorants et aux postdoctorantes et postdoctorants qui ont partagé un bout de vie avec moi.

Merci aux personnes qui ont bien voulu relire des parties de ce livre.

Et parce que je ne suis pas grand-chose sans eux, ce livre est dédié à Eleni, Gabriel et Mélina.

Relecture : Sophie De Decker
Couverture et mise en page : Hélène Bonnet Studio 9
Création maquette intérieure : Paul Mounier-Piron

Achevé d'imprimer en
par

Numéro d'impression :
Dépôt légal :

La statistique bayésienne est partout : prévisions météo, analyses d'épidémies, préservation de la biodiversité… Dans un monde incertain, elle permet d'estimer, de prédire et de décider, en donnant un sens aux données.

Ce livre propose une introduction accessible et concrète à la statistique bayésienne. L'auteur explique pas-à-pas les fondements de l'approche, sa singularité et la logique qui sous-tend le raisonnement bayésien. L'apprentissage s'appuie sur le logiciel libre R et se construit à partir de questions de recherche liées à l'écologie du ragondin qui constitue le fil rouge de l'ouvrage.

Chaque chapitre aborde un pilier essentiel : le théorème de Bayes, les méthodes de Monte Carlo par chaînes de Markov (MCMC), le choix et le rôle des distributions a priori, la régression linéaire et ses extensions, les modèles linéaires généralisés (mixtes ou pas), jusqu'à la comparaison et la validation de modèles. Le lecteur est invité à coder, simuler, tester et visualiser pour comprendre, grâce aux exemples et au matériel en ligne.

Rédigé dans un style direct et didactique, comme un échange entre enseignant et étudiant, l'ouvrage démystifie la statistique bayésienne. Il s'adresse à tous ceux qui souhaitent se former à cette approche : étudiants, universitaires… et toute personne souhaitant appliquer ces méthodes en sciences du vivant, des données ou de l'environnement.

Olivier Gimenez est directeur de recherche au CNRS. Après des études universitaires en mathématiques, il a soutenu une thèse en statistique appliquée à l'écologie puis obtenu son habilitation à diriger des recherches en écologie et évolution. Il a complété sa formation avec une initiation à la sociologie. Il a écrit des articles scientifiques et coécrit des ouvrages, dont plusieurs abordent la statistique bayésienne.

UNIVERSITÉ DE MONTPELLIER

cnrs

éditions Quæ

Éditions Cirad, Ifremer, INRAE
www.quae.com

16 €

ISBN : 978-2-7592-4257-3

9 782759 242573

Réf. : 03051