Développement d'un modèle joint de distribution des espèces pour la réalisation d'une carte de biodiversité à Madagascar

Jeanne Clément

Rapport de stage, Février à Août 2019

Enseignant référent : Benoite De Saporta

Encadrant : Ghislain Vieilledent

Master Maths-Biostatistique

Université Montpellier 2

UMR AMAP - Montpellier

botAnique et Modélisation de l'Architecture des Plantes et des végétations

Remerciements

J'aimerai adresser mes plus sincères remerciements à G. Vieilledent qui m'a encadrée et conseillée durant ce stage riche en découvertes puisque le language C++, la construction de packages R ainsi que les modèles joints de distribution des espèces m'étaient inconnus. Il m'a beaucoup appris et encouragée à trouver des solutions par moi même. Je remercie également les chercheurs et autres stagiaires de l'UMR AMAP pour leur accueil chaleureux et leur bonne humeur communicative qui font du laboratoire un cadre de travail idéal et tout particulièrement G. Le Moguedec qui fut une référence précieuse en statistiques ainsi que l'instigateur de pic-niques au lac du Crès qui nous ont bien aidé à supporter la canicule.

Sommaire

Liste des figures

Liste des tableaux

Introduction

J'ai effectué mon stage au sein de l'UMR AMAP (botAnique et Modélisation de l'Architecture des Plantes et des végétations), qui se trouve à Montpellier. Il s'agit d'une unité interdisciplinaire hébergée par le Cirad ou « Centre de Coopération Internationale en Recherche Agronomique pour le Développement» et qui mène des recherches sur les plantes et les végétations, dans le but de prévoir la réponse des écosystèmes aux forçages environnementaux.

Ce stage s'inscrit dans le cadre du projet BioSceneMada qui vise à fournir des scénarios d'évolution de la biodiversité sous l'effet conjoint du changement climatique et de la déforestation à Madagascar. Pour ce faire, plusieurs jeux de données sur la biodiversité ont été collectés et regroupés pour différents groupes taxonomiques (mammifères, oiseaux, reptiles, amphibiens, arbres, plantes herbacées, invertébrés), parmi lesquels j'ai utilisé des inventaires forestiers répertoriant l'absence ou la présence d'espèces d'arbres sur différents sites de l'île ainsi que des variables bioclimatques afin d'ajuster un modèle joint de distribution des espèces permettent d'estimer la niche des espèces, de prédire leur distribution, tout en prenant en compte les intéractions entre espèces (Warton et al. (2015)).
Dans un premier temps j'ai implémenté différents échantillonneurs de Gibbs en C++ permettant d'estimer les paramètres de modèles joints de distribution des espèces (JSDM) comportant des variables latentes, à l'aide du package Rcpp. La construction du package R https://ecology.ghislainv.fr/jSDM/ autour de l'une de ces fonctions ainsi que sa présentation à la conférence useR 2019 ont constitué une partie importante de mon stage. L'ajustement d'un JSDM sur des données d'inventaires forestiers collectées à Madagascar ainsi que des variables climatiques et environnementales, m'a permis d'obtenir des carte reflétant la biodiversité sur l'île afin de par la suite identifier des zones refuges de la biodiversité sous l'effet du changement climatique en utilisant les variables bioclimatiques fournies par les scénarios du GIECC. Ces résultats seront utilisés pour des préconisations de gestion de la biodiversité dans le cadre du projet BioSceneMada.

1 Définition des modèles joints de distribution des espèces envisagés

Les données dont on dispose pour ajuster ce type de modèle sont les réalisations d'une variable réponse, $Y = (y_{ij})_{j=1,\dots,J}^{i=1,\dots,I}$ telle que :

$$y_{ij} = \begin{cases} 0 & \text{si l'espèce } j \text{ est absente du site } i \\ 1 & \text{si l'espèce } j \text{ est présente sur le site } i, \end{cases}$$

ainsi que de variables explicatives $X = (X_i)_{i=1,\dots,I}$ avec $X_i = (X_{i1}, \dots, X_{ip}) \in \mathbb{R}^p$ où p est le nombre de variables bioclimatiques considérées pour chaque site.
On note θ_{ij}, la probabilité de présence de l'espèce j sur le site i.

L'article Warton et al. (2015) développe deux approches hiérarchiques pouvant être utilisées à la spécification d'un modèle joint de distribution des espèces.

1.1 Modèle linéaire mixte généralisé (GLMM)

D'une part on pourrait utiliser un modèle linéaire mixte généréralisé **(GLMM)** de la forme :

$$g(\theta_{ij}) = \alpha_i + \beta_{j0} + X_i\beta_j + u_{ij},$$

$$y_{ij} \mid u_{ij}, \alpha_i \sim \mathcal{B}ernoulli(\theta_{ij}),$$

$$u_i \sim \mathcal{N}_J(0_{\mathbb{R}^J}, \Sigma) \; iid,$$

$$\alpha_i \sim \mathcal{N}(0, V_\alpha) \; iid \text{ et indépendant de } u_i.$$

où $g :]0,1[\to]-\infty, +\infty[$ est une fonction de lien, $\beta_j = (\beta_{j1}, \dots, \beta_{jp})'$ et β_{j0} sont les coefficients de régression correspondants aux variables bioclimatiques et l'intercept pour l'espèce j qui est supposé être un effet fixe, α_i représente l'effet aléatoire du site i, et $u_i = (u_{i1}, \dots, u_{iJ})$ est un effets aléatoires multivariés corrélés dont la matrice de variance covariance Σ controle la correlation entre les espèces et est supposée être complètement non structurée.
Cette dernière partie du modèle est problématique lorsque le nombre d'espèces J est important car le nombre de paramètres dans Σ augmente quadratiquement avec J.

1.2 Modèle à variable latente (LVM)

D'autre part en posant $u_{ij} = W_i\lambda_j$, avec $W_i = (W_{i1}, \dots, W_{iq})$ les q prédicteurs non mesurés (ou "variables latentes") considérés et $\lambda_j = (\lambda_{j1}, \dots, \lambda_{jq})'$ les coefficients associés, on obtient le modèle à variables latentes **(LVM)** suivant :

$$g(\theta_{ij}) = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j$$

$$y_{ij} \mid W_i, \alpha_i \sim \mathcal{B}ernoulli(\theta_{ij}),$$

$$W_i \sim \mathcal{N}(0, I_q) \; iid$$

$$\alpha_i \sim \mathcal{N}(0, V_\alpha) \; iid \text{ et indépendant de } W_i$$

Ce qui revient à un cas particulier de GLMM multivarié auquel on impose la contrainte $\Sigma = \Lambda\Lambda'$ avec

$$\Lambda := \begin{pmatrix} \lambda_{11} & \dots & \lambda_{1q} \\ \vdots & \vdots & \vdots \\ \lambda_{J1} & \dots & \lambda_{Jq} \end{pmatrix}$$

On préferera ce dernier modèle, en effet il comporte potentiellement beaucoup moins de paramètres que le GLMM précédent car Λ a autant de colonne qu'il y a de variables latentes (q) tandis que Σ présente autant de colonnes de paramètres qu'il y a d'espèces (J).

On peut choisir de modéliser l'abondance absolue plutôt que l'abondance relative en supprimant l'effet site aléatoire α_i du modèle.

2 Méthodes d'inférence bayesienne selon la fonction de lien choisie

2.1 Principe d'un échantillonneur de Gibbs

Dans le cadre bayésien, l'algorithme de Gibbs permet d'obtenir une réalisation du paramètre $\theta = (\theta_), \ldots, \theta_m)$ suivant la loi *a posteriori* $\pi(\theta \mid x)$ dès que l'on est capable d'exprimer les lois conditionnelles : $\pi(\theta_i \mid \theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_m, x)$ pour $i = 1, \ldots, m$.

L'**échantillonnage de Gibbs** consiste à :

— **Initialisation** : choix arbitraire de $\theta^{(0)} = (\theta_1^{(0)}, \ldots, \theta_m^{(0)})$.

— **Itération** t : Générer $\theta^{(t)}$ de la manière suivante :

- $\theta_1^{(t)} \sim \pi\left(\theta_1 \mid \theta_2^{(t-1)}, \ldots, \theta_m^{(t-1)}, x\right)$
- $\theta_2^{(t)} \sim \pi\left((\theta_2 \mid (\theta_1^{(t)}, \theta_3^{(t-1)}, \ldots, \theta_m^{(t-1)}, x\right)$
- $\theta_m^{(t)} \sim \pi\left(\theta_m \mid \theta_1^{(t)}, \ldots, \theta_{m-1}^{(t)}, x\right)$

Les itérations successives de cet algorithme génèrent les états d'une chaîne de Markov $\{\theta^{(t)}, t > 0\}$ à valeurs dans $\mathbb{R}^m$, on montre que cette chaîne admet une mesure invariante qui est la *loi a posteriori*.
Pour un nombre d'itérations suffisamment grand, le vecteur θ obtenu peut donc être considéré comme étant une réalisation de la loi *a posteriori* $\pi(\theta \mid x)$.

Par conséquent l'implémentation d'un échantillonneur de Gibbs nécessite la connaissance des ditributions *a posteriori* de chacun des paramètres conditionnellement aux autres paramètres du modèle, qui se déduisent des formules de priors conjugués dans le cas du modèle probit mais ne sont pas explicitement exprimables dans le cas où on utilise une fonction de lien logit.

2.2 Modèle probit : échantillonneur de Gibbs et priors conjugués

D'une part, on utilise un fonction de lien probit : $p \to \Phi^{-1}(p)$ où Φ correspond à la fonction de répartition d'une loi normale centrée réduite.

2.2.1 Définition du modèle probit

D'après l'article Albert and Siddhartha (1993), une modélisation possible est de supposer l'existence d'une variable latente sous-jacente liée à notre variable binaire observées en utilisant la proposition suivante :

Proposition 2.2.1.1 (Modèle probit par l'intermédiaire une variable latente)**.**

Si $Z_{ij} = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j + \epsilon_{ij},\ \forall i, j$ *avec* $\epsilon_{ij} \sim \mathcal{N}(0, 1)$ *iid et tel que :*

$$y_{ij} = \begin{cases} 1 & \text{si } Z_{ij} > 0 \\ 0 & \text{sinon.} \end{cases}$$

Alors on a $y_{ij} \mid Z_{ij} \sim \mathcal{B}ernoulli(\theta_{ij})$ *avec* $\text{probit}(\theta_{\text{ij}}) = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j$.

Preuve 2.2.1.1.

$$\begin{aligned} \mathbb{P}(y_{ij} = 1) &= \mathbb{P}(Z_{ij} > 0) \\ &= \mathbb{P}(\alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j + \epsilon_{ij} > 0) \\ &= \mathbb{P}(\epsilon_{ij} > -(\alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j)\ ) \\ &= \mathbb{P}(\epsilon_{ij} \leq \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j) \\ &= \Phi(\alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j) \end{aligned}$$

De la même façon on a :

$$\begin{aligned}\mathbb{P}(y_{ij}=0) &= \mathbb{P}(Z_{ij}\leq 0)\\ &= 1-\Phi(\alpha_i+\beta_{j0}+X_i\beta_j+W_i\lambda_j)\end{aligned}$$

On définit le modèle probit à l'aide d'une variable latente afin d'être en mesure d'uiliser les propriétés des priors conjugués pour échantillonner les paramètres du modèle selon leur distributions conditionnelles *a posteriori*.

2.2.2 Priors utilisés

Afin d'utiliser une méthode d'inférence bayesienne on détermine une distribution *a priori* pour chacun des paramètres du modèle :

$$\begin{array}{lcl} V_\alpha & \sim & \mathcal{IG}(\text{shape}=0.5,\text{rate}=0.005) \text{ avec rate} = \frac{1}{\text{scale}},\\ \beta_{jk} & \sim & \mathcal{N}(0,10^6) \text{ pour } j=1,\dots,J \text{ et } k=1,\dots,p,\\ \lambda_{jl} & \sim & \begin{cases}\mathcal{N}(0,10) & \text{si } l<j\\ \mathcal{N}(0,10) \text{ tronquée à gauche par 0} & \text{si } l=j\\ P \text{ tel que } \mathbb{P}(\lambda_{jl}=0)=1 & \text{si } l>j\end{cases}\\ & & \text{pour } j=1,\dots,J \text{ et } l=1,\dots,q.\end{array}$$

En effet pour assurer l'identifiabilité du modèle les valeurs de Λ sont contraintes à des valeurs strictements positives sur la diagonale et nulles au dessus de celle-ci, Λ est ainsi supposée être triangulaire inférieure d'après l'article Warton et al. (2015).
La fonction *boral()* du package du même nom, permettant d'ajuster toutes sortes de modèles utilise ces distributions *a priori* pour le modèle qui nous intéresse. Dans l'article Warton et al. (2015) l'ajustement de modèle joints de distributions des espèces est réalisé avec *boral()* qui fonctionne avec JAGS (Just Another Gibbs Sampler) un programme de simulation à partir de modèles hiérarchiques bayésiens utilisant des méthodes MCMC, implémenté en C++. Cependant la fonction *jSDM_probit_block()* du package jSDM que j'ai implémentée utilise une distribution *a priori* jointe pour les effets espèces fixes de la manière qui suit.

2.2.3 Propositions sur les priors conjugués

Effets espèces fixes :

On se ramène à un modèle de la forme $Z^*=X\beta+\epsilon$, en posant $Z^*_{ij}=Z_{ij}-\alpha_i=\beta_{j0}+X_i\beta_j+W_i\lambda_j+\epsilon_{ij}$, afin d'estimer simultanément les β_j et λ_j pour chacune des espèces j, ce qui revient en écriture matricielle à :

$$\begin{aligned}Z^*_j=\begin{pmatrix}Z^*_{1j}\\ \vdots\\ Z^*_{Ij}\end{pmatrix} &= \underbrace{\begin{pmatrix}1 & X_{11} & \dots & X_{1p} & W_{11} & \dots & W_{1q}\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 1 & X_{I1} & \dots & X_{Ip} & W_{I1} & \dots & W_{Iq}\end{pmatrix}}_{D} \underbrace{\begin{pmatrix}\beta_{j0}\\ \beta_{j1}\\ \vdots\\ \beta_{jp}\\ \lambda_{j1}\\ \vdots\\ \lambda_{jq}\end{pmatrix}}_{P_j} + \begin{pmatrix}\epsilon_{1j}\\ \vdots\\ \epsilon_{Ij}\end{pmatrix}\\ &= DP_j+\epsilon_j \quad \text{avec } \epsilon_j\sim\mathcal{N}_I(0_{\mathbb{R}^I},I_I).\end{aligned}$$

On suppose que $P_j\sim\mathcal{N}_{p+q+1}(m,V)$ avec $m=0_{\mathbb{R}^{p+q+1}}$ et $V=diag(\underbrace{10^6,\dots,10^6}_{\times p+1},\underbrace{10,\dots,10}_{\times q})$, par exemple.

Bien que cette distribution *a priori* ne prennent pas en compte les contraintes sur Λ, elle permet l'échantillonage selon une loi normale multivariée des effets espèce fixes. On imposera les contraintes aux λ_{jl} concernés après les avoir simulés.

On applique la proposition suivante :

Proposition 2.2.3.1.

$$\begin{cases}Y\mid\beta\sim\mathcal{N}_n(X\beta,I_n)\\ \beta\sim\mathcal{N}_p(m,V)\end{cases}\Rightarrow\begin{cases}\beta|Y\sim\mathcal{N}_p(m^*,V^*) \textit{ avec}\\ m^*=(V^{-1}+X'X)^{-1}(V^{-1}m+X'Y)\\ V^*=(V^{-1}+X'X)^{-1}\end{cases}$$

.

Preuve 2.2.3.1.

$$
\begin{aligned}
p(\beta \mid Y) &\propto p(Y \mid \beta)\ p(\beta) \\
&\propto \frac{1}{(2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2}(Y - X\beta)'(Y - X\beta)\right) \frac{1}{(2\pi)^{\frac{p}{2}}|V|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\beta - m)'V^{-1}(\beta - m)\right) \\
&\propto \exp\left(-\frac{1}{2}\left((\beta - m)'V^{-1}(\beta - m) + (Y - X\beta)'(Y - X\beta)\right)\right) \\
&\propto \exp\left(-\frac{1}{2}\left(\beta'V^{-1}\beta + m'V^{-1}m - m'V^{-1}\beta - \beta'V^{-1}m + Y'Y + \beta'X'X\beta - Y'X\beta - \beta'X'Y\right)\right) \\
&\propto \exp\left(-\frac{1}{2}\left(\beta'(V^{-1} + X'X)\beta - \beta'(V^{-1}m + X'Y) - (Y'X + m'V^{-1})\beta + m'V^{-1}m + Y'Y\right)\right) \\
&\propto \exp\left(-\frac{1}{2}\left(\beta'(V^{-1} + X'X)\beta - \beta'(V^{-1}m + X'Y) - (X'Y + V^{-1}m)'\beta + m'V^{-1}m + Y'Y\right)\right) \\
&\propto \exp(-\frac{1}{2}\left(\beta - (V^{-1} + X'X)^{-1}(V^{-1}m + X'Y)\right)'(V^{-1} + X'X)\left(\beta - (V^{-1} + X'X)^{-1}(V^{-1}m + X'Y)\right) \\
&\quad - (V^{-1}m + X'Y)'(V^{-1} + X'X)^{-1}(V^{-1}m + X'Y) + m'V^{-1}m + Y'Y) \\
&\propto \exp\left(-\frac{1}{2}\left(\beta - \underbrace{(V^{-1} + X'X)^{-1}(V^{-1}m + X'Y)}_{m^*}\right)' \underbrace{(V^{-1} + X'X)}_{V^{*-1}}\left(\beta - (V^{-1} + X'X)^{-1}(V^{-1}m + X'Y)\right)\right)
\end{aligned}
$$

On obtient :

$$
\begin{cases} Z_j^* \mid P_j \sim \mathcal{N}_I(DP_j, I_I) \\ P_j \sim \mathcal{N}_{p+q+1}(m, V) \end{cases} \Rightarrow \begin{cases} P_j \mid Z_j^* \sim \mathcal{N}_{p+q+1}(m^*, V^*) \text{ avec} \\ m^* = (V^{-1} + D'D)^{-1}(V^{-1}m + D'Z_j^*) \\ V^* = (V^{-1} + D'D)^{-1} \end{cases}
$$

.

Prédicteurs non mesurés (ou "variables latentes") :

De la même façon, on pose : $Z_{ij}^\star = Z_{ij} - \alpha_i - \beta_{j0} - X_i\beta_j = W_i\lambda_j + \epsilon_{ij}$, afin d'estimer W_i pour chaque site i.
En appliquant la proposition précédente, on obtient :

$$
\begin{cases} Z_i^* := (Z_{i1}^*, \ldots, Z_{iJ}^*)' \mid W_i \sim \mathcal{N}_J(\Lambda W_i', I_J) \\ W_i' \sim \mathcal{N}_q(0_{\mathbb{R}^q}, I_q) \end{cases} \Rightarrow \begin{cases} W_i' \mid Z_i^* \sim \mathcal{N}_q(m^*, V^*) \text{ avec} \\ m^* = (I_q + \Lambda'\Lambda)^{-1}(\Lambda' Z_i^*) \\ V^* = (I_q + \Lambda'\Lambda)^{-1} \end{cases}
$$

.

Effets site aléatoires et variance associée :

En ce qui concerne l'effet site aléatoire $(\alpha_i)_{i=1,\ldots,I}$, on pose $Z_{ij}^* = Z_{ij} - D_iP_j = \alpha_i + \epsilon_{ij}$, avec $D_i = (1, X_{i1}, \ldots, X_{ip}, W_{i1}, \ldots, W_{iq})$.
On a ainsi $Z_{ij}^* \mid \alpha_i \ \sim \mathcal{N}(\alpha_i, 1)$ *iid* pour $j = 1, \ldots, J$, puis on applique la proposition suivante :

Proposition 2.2.3.2.

$$
\begin{cases} x_i \mid \theta \sim \mathcal{N}(\theta,\ \sigma^2)\ \textit{iid pour } i = 1, \ldots, n \\ \theta \sim \mathcal{N}(\mu_0, {\tau_0}^2) \\ \sigma^2\ \textit{connu} \end{cases} \Rightarrow \begin{cases} \theta |\ x_1, \ldots, x_n \sim \mathcal{N}(\mu_1, {\tau_1}^2)\ \textit{avec} \\ \mu_1 = \dfrac{{\tau_0}^{-2}\mu_0 + \sigma^{-2}\sum_{i=1}^n x_i}{{\tau_0}^{-2} + n\sigma^{-2}} \\ {\tau_1}^{-2} = {\tau_0}^{-2} + n\sigma^{-2} \end{cases}
$$

.

Preuve 2.2.3.2.

$$\begin{aligned}
p(\theta \mid x_1,\dots,x_n) &\propto p(\theta)p(x_1,\dots,x_n \mid \theta)\\
&\propto \frac{1}{(2\pi {\tau_0}^2)^{\frac{1}{2}}}\exp\left(-\frac{1}{2{\tau_0}^2}(\theta-\mu_0)^2\right)\prod_{i=1}^{n}\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}\exp\left(-\frac{1}{2\sigma^2}(x_i-\theta)^2\right)\\
&\propto \exp\left(-\frac{1}{2{\tau_0}^2}(\theta-\mu_0)^2-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\theta)^2\right)\\
&\propto \exp\left(-\frac{1}{2{\tau_0}^2}(\theta^2-2\mu_0\theta)-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(\theta^2-2\theta x_i)\right)\\
&\propto \exp\left(-\frac{1}{2}\left(\theta^2({\tau_0}^{-2}+n\sigma^{-2})-2\mu_0\theta{\tau_0}^{-2}-2\theta\sigma^{-2}\sum_{i=1}^{n}x_i\right)\right)\\
&\propto \exp\left(-\frac{1}{2(\tau_0^{-2}+n\sigma^{-2})^{-1}}\left(\theta-\frac{\mu_0{\tau_0}^{-2}+\sigma^{-2}\sum_{i=1}^{n}x_i}{{\tau_0}^{-2}+n\sigma^{-2}}\right)^2\right)
\end{aligned}$$

On obtient ainsi :

$$\begin{cases} Z^*_{ij} \mid \alpha_i \sim \mathcal{N}(\alpha_i,\ 1),\ \text{iid}\ \forall j=1,\dots,J \\ \alpha_i \sim \mathcal{N}(0,V_\alpha)\end{cases} \Rightarrow \begin{cases}\alpha_i|\ Z^*_{i1},\dots,Z^*_{iJ}\sim\mathcal{N}(\mu_1,{\tau_1}^2)\ \text{avec}\\ \mu_1=\dfrac{\sum_{j=1}^{J}Z^*_{ij}}{V_\alpha^{-1}+J}\ \text{et}\ {\tau_1}^{-2}=V_\alpha^{-1}+J.\end{cases}$$

Finalement pour estimer V_α, la variance des effets site aléatoires $(\alpha_i)_{i=1,\dots,I}$, on utilise la proposition suivante :

Proposition 2.2.3.3. *Si*

$$\begin{cases} x \mid \sigma^2 \sim \mathcal{N}_n(\theta,\ \sigma^2 I_n)\\ \sigma^2 \sim \mathcal{IG}(a,b)\\ \theta\ \textit{connu}\end{cases} \Rightarrow \begin{cases}\sigma^2|x\sim\mathcal{IG}(a',b')\ \textit{avec}\\ a'=a+\frac{n}{2}\ \textit{et}\ b'=\frac{1}{2}\sum_{i=1}^{n}(x_i-\theta)^2+b.\end{cases}$$

Preuve 2.2.3.3.

$$\begin{aligned}
p(\sigma^2 \mid x) &\propto p(x \mid \sigma^2)\ p(\sigma^2)\\
&\propto \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}}\exp\left(-\frac{1}{2\sigma^2}(x-\theta)'(x-\theta)\right)\frac{b^a}{\Gamma(a)}(\sigma^2)^{-(a+1)}\exp\left(-\frac{b}{\sigma^2}\right)\\
&\propto (\sigma^2)^{-\left(\frac{n}{2}+a+1\right)}\exp\left(-\frac{1}{\sigma^2}\left(b+\frac{1}{2}\sum_{i=1}^{n}(x_i-\theta)^2\right)\right)
\end{aligned}$$

On a donc :

$$\begin{cases}(\alpha_1,\dots,\alpha_I)' \mid V_\alpha \sim \mathcal{N}_n(0_{\mathbb{R}^I},V_\alpha I_n)\\ V_\alpha\sim\mathcal{IG}(a,b)\end{cases} \Rightarrow \begin{cases}V_\alpha \mid \alpha_1,\dots,\alpha_I\sim\mathcal{IG}(a',b')\ \text{avec}\\ a'=a+\frac{I}{2}\ \text{et}\ b'=b+\frac{1}{2}\sum_{i=1}^{I}\alpha_i^2.\end{cases}$$

2.2.4 Echantilloneur de Gibbs et priors conjugués

L'algorithme utilisé pour estimer les paramètres du modèle logit est donc le suivant :

— Définir les constantes N_{Gibbs}, N_{burn}, N_{thin} telles que N_{Gibbs} correspond au nombre d'itérations effectuées par l'échantillonneur de Gibbs, N_{burn} au nombre d'itérations nécessaires pour le burn-in ou temps de chauffe et $N_{samp} = \frac{N_{Gibbs} - N_{burn}}{N_{thin}}$ au nombre de valeurs estimées retenues pour chaque paramètre. En effet on enregistre les paramètres estimés à certaines itérations, afin d'obtenir un échantillon de N_{samp} valeurs distribuées selon la distribution a posteriori pour chacun des paramètres.

— Initialiser tous les paramètres à 0 par exemple, excepté les valeurs diagonales de Λ initialisées à 1 et $V_\alpha^{(0)} = 1$.

— Gibbs sampler : à chaque itération t pour $t = 1, \ldots, N_{Gibbs}$ on répète chacune de ces étapes :

- Génerer la **variable latente** $Z^{(t)} = \left(Z_{ij}^{(t)}\right)_{i=1,\ldots,I}^{j=1,\ldots,J}$ telle que

$$Z_{ij}^{(t)} \sim \begin{cases} \mathcal{N}\left(\alpha_i^{(t1)} + \beta_{j0}^{(t1)} + X_i\beta_j(t1) + W_i^{(t1)}\lambda_j^{(t1)},\ 1\right) & \text{tronquée à droite par 0} \quad \text{si } y_{ij} = 0 \\ \mathcal{N}\left(\alpha_i^{(t1)} + \beta_{j0}^{(t1)} + X_i\beta_j(t1) + W_i^{(t1)}\lambda_j^{(t1)},\ 1\right) & \text{tronquée à gauche par 0} \quad \text{si } y_{ij} = 1 \end{cases}$$

  , la variable latente est ainsi initialisée à la première itération en la générant selon ces lois normales centrées.

- Générer les **effets espèces fixes** $P_j^{(t)} = (\beta_{j0}^{(t)}, \beta_{j1}^{(t)} \ldots, \beta_{jp}^{(t)}, \lambda_{j1}^{(t)}, \ldots, \lambda_{jq}^{(t)})'$ pour $j = 1, \ldots, J$ selon :

$$P_j^{(t)} \mid Z^{(t)}, W_1^{(t-1)}, \alpha_1^{(t-1)}, \ldots, W_I^{(t1)}, \alpha_I^{(t-1)} \sim \mathcal{N}_{p+q+1}(m^\star, V^\star), \text{ avec}$$

$$m^\star = (V^{-1} + {D^{(t)}}'D^{(t)})^{-1}(V^{-1}m + {D^{(t)}}'Z_j^\star) \text{ et } V^\star = \left(V^{-1} + {D^{(t)}}'D^{(t)}\right)^{-1},$$

$$\text{où } Z_j^\star = (Z_{1j}^\star, \ldots, Z_{Ij}^\star)' \text{ tel que } Z_{ij}^\star = Z_{ij}^{(t)} - \alpha_i^{(t-1)}.$$

  Afin de contraindre les valeurs diagonales de $\Lambda = (\lambda_{jl})_{j=1,\ldots,J}^{l=1,\ldots,q}$ à des valeurs positives et de rendre la matrice triangulaire inférieure, on modifie les valeurs des $P^{(t)}$ simulées aléatoirement selon les conditions suivantes :

$$P_{jp+1+l}^{(t)} = \lambda_{jl}^{(t)} \leftarrow \begin{cases} 0 & \text{si } l > j \\ \lambda_{jl}^{(t-1)} & \text{si } l = j \text{ et } \lambda_{jl}^{(t)} < 0. \end{cases}$$

  On pose $P^{(t)} = \left(P_1^{(t)} | \ldots | P_J^{(t)}\right)$.

- Générer les **prédicteurs non mesurés** (ou "variables latentes") $W_i^{(t)}$ pour $i = 1, \ldots, I$ selon :

$$W_i^{(t)} \mid Z^{(t)}, P^{(t)}, \alpha_i^{(t-1)} \sim \mathcal{N}_q\left((I_q + {\Lambda^{(t)}}'\Lambda^{(t)})^{-1}({\Lambda^{(t)}}'Z_i^{\star\star}), (I_q + {\Lambda^{(t)}}'\Lambda^{(t)})^{-1}\right),$$

$$\text{où } Z_i^{\star\star} = (Z_{i1}^{\star\star}, \ldots, Z_{iJ}^{\star\star}) \text{ tel que } Z_{ij}^{\star\star} = Z_{ij}^{(t)} - \alpha_i^{(t-1)}\beta_{j0}^{(t)} - X_i\beta_j^{(t)}.$$

  On pose $D_i^{(t)} = \left(1, X_{i1}, \ldots, X_{ip}, W_{i1}^{(t)}, \ldots, W_{iq}^{(t)}\right)$.

- Générer les **effets sites aléatoires** $\alpha_i^{(t)}$ pour $i = 1, \ldots, I$ selon :

$$\alpha_i | \ Z^{(t)}, P^{(t)}, W_i^{(t)} \sim \mathcal{N}\left(\frac{\sum_{j=1}^J Z_{ij}^{(t)} - D_i^{(t)}P_j^{(t)}}{{V_\alpha^{(t-1)}}^{-1} + J}, \left(\frac{1}{V_\alpha^{(t-1)}} + J\right)^{-1}\right)$$

- Générer la **variance des effets site aléatoires** $V_\alpha^{(t)}$ selon :

$$V_\alpha^{(t)} \mid \alpha_1^{(t)}, \ldots, \alpha_I^{(t)} \sim \mathcal{IG}\left(\text{shape} = 0.5 + \frac{I}{2}, \text{rate} = 0.005 + \frac{1}{2}\sum_{i=1}^I \left(\alpha_i^{(t)}\right)^2\right)$$

2.3 Modèle logit : échantillonneur de Gibbs et algorithme de Metropolis adaptatif

D'autre part on considère une fonction de lien logit : $p \to \ln\left(\frac{p}{1-p}\right) = \mathrm{F}^{-1}(p)$, avec $\mathrm{F} : x \to \frac{1}{1+e^{-x}}$ la fonction de répartition appelée sigmoïde d'une loi logistique standard.

2.3.1 Définition du modèle logit

De la même façon que pour le modèle probit, d'après Marine Guillerm (2016) on peut définir le modèle logit par l'intermediaire d'une variable latente : $Z_{ij} = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j + \epsilon_{ij}$ pour $i = 1, \ldots, I$ et $j = 1, \ldots, J$, avec $\epsilon_{ij} \sim$ logistique$(0, 1)$ *iid* et telle que :

$$y_{ij} = \begin{cases} 1 & \text{si } Z_{ij} > 0 \\ 0 & \text{sinon.} \end{cases}$$

Cependant dans ce cas les distributions *a priori* de la variable latente et des paramètres n'étant pas conjuguées, on n'est pas en mesure d'utiliser les propriétés des priors conjugués donc la modélisation à l'aide d'une variable latente ne présente pas d'intérêt.

Dans ce cas on suppose que

$$y_{ij} \mid \theta_{ij} \sim \mathcal{B}(n_i, \theta_{ij})$$

, avec probit$(\theta_{\mathrm{ij}}) = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j$ et n_i le nombre de visites du site i.

Par conséquent on échantillonnera les paramètres de ce modèle selon une estimation de leurs distributions conditionnelles *a posteriori* à l'aide d'un algorithme de Metropolis adaptatif.

2.3.2 Priors utilisés

On détermine une distribution *a priori* pour chacun des paramètres du modèle :

$$\begin{array}{lcl} V_\alpha & \sim & \mathcal{IG}(\text{shape} = 0.5, \text{rate} = 0.005) \text{ avec rate} = \frac{1}{\text{scale}}, \\ \beta_{jk} & \sim & \mathcal{N}(0, 10^6) \text{ pour } j = 1, \ldots, J \text{ et } k = 1, \ldots, p, \\ \lambda_{jl} & \sim & \begin{cases} \mathcal{N}(0, 10) & \text{si } l < j \\ \mathcal{U}(0, 10) & \text{si } l = j \\ P \text{ tel que } \mathbb{P}(\lambda_{jl} = 0) = 1 & \text{si } l > j \end{cases} \\ & & \text{pour } j = 1, \ldots, J \text{ et } l = 1, \ldots, q. \end{array}$$

2.3.3 Principe d'un algorithme de Metropolis adaptatif

Cet algorithme appartient aux méthode MCMC et permet d'obtenir une réalisation du paramètre $\theta = (\theta_), \ldots, \theta_m)$ selon leurs distributions conditionnelles *a posterirori* $\pi(\theta_i \mid \theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_m, x)$, pour $i = 1, \ldots, m$ connues à une constante multiplicative près.
On le qualifie d'adaptatif car la variance de la densité instrumentale conditionnelle utilisée est adaptée en fonction du nombre d'acceptation lors des dernières itérations.

- **Initialisation** : $\theta^{(0)} = (\theta_1^{(0)}, \ldots, \theta_m^{(0)})$ fixés arbitrairement, les nombres d'acceptation $(n_i^A)_{i=1,\ldots,m}$ sont intialisés à 0 et les variances $(\sigma_i^2)_{i=1,\ldots,m}$ sont intialisées à 1.
- **Itération t** : pour $i = 1, \ldots, m$
  - Générer $\theta_i^\star \sim q(\theta_i^{(t-1)}, .)$, avec une densité instrumentale conditionnelle $q(\theta_i^{(t-1)}, \theta_i^\star)$ symétrique, on choisira une loi $\mathcal{N}(\theta_i^{(t-1)}, \sigma_i^2)$ par exemple.
  - Calculer la probabilité d'acceptation :

$$\gamma = min\left(1, \frac{\pi\left(\theta_i^\star \mid \theta_1^{(t-1)}, \ldots, \theta_{i-1}^{(t-1)}, \theta_{i+1}^{(t-1)}, \ldots, \theta_m^{(t-1)}, x\right)}{\pi\left(\theta_i^{(t-1)} \mid \theta_1^{(t-1)}, \ldots, \theta_{i-1}^{(t-1)}, \theta_{i+1}^{(t-1)}, \ldots, \theta_m^{(t-1)}, x\right)}\right)$$

    .
  - Retenir

$$\theta_i^{(t)} = \begin{cases} \theta_i^\star & \text{avec probabilité } \gamma \\ & \text{si on est dans ce cas le nombre d'acceptation devient : } n_i^A \leftarrow n_i^A + 1 \\ \theta_i^{(t-1)} & \text{avec probabilité } 1 - \gamma. \end{cases}$$

— **Durant le burnin**, toutes les DIV itérations, avec

$$\text{DIV} = \begin{cases} 100 & \text{si } N_{Gibbs} \geq 1000 \\ \dfrac{N_{Gibbs}}{10} & \text{sinon} \end{cases}$$

, où N_{Gibbs} est le nombre total d'itération effectuées.
On modifie les variances en fonction des nombres d'acceptation de la manière suivante pour $i = 1, \ldots, m$:

- On calcule le taux d'acceptation : $r_i^A = \dfrac{n_i^A}{\text{DIV}}$.
- On adapte les variances selon le taux d'acceptation et une constante fixée R_{opt} :

$$\sigma_i \leftarrow \begin{cases} \sigma_i \left(2 - \dfrac{1 - r_i^A}{1 - R_{opt}}\right) & \text{si } r_i^A \geq R_{opt} \\ \dfrac{\sigma_i}{2 - \dfrac{1 - r_i^A}{1 - R_{opt}}} & \text{sinon} \end{cases}$$

- On réinitialise les nombres d'acceptation : $n_i^A \leftarrow 0$.

— Toutes les $\dfrac{N_{Gibbs}}{10}$ itérations, on calcule et affiche les taux d'acceptation moyen $m^A = \dfrac{1}{m} \sum\limits_{i=1,\ldots,m} r_i^A$.

2.3.4 Echantilloneur de Gibbs et algorithme de Metropolis adapatatif

On utilise un algorithme de Metropolis adapatatif pour échantillonner les paramètres du modèle selon leurs distributions conditionnelles *a posteriori* estimées à une constante multiplicative près.

Dans un premier temps on définit la fonction f permettant de calculer la vraisemblance du modèle en fonction des paramètres estimés :

$$f : \lambda_j, \beta_{j0}, \beta_j, \alpha_i, W_i, X_i, y_{ij}, n_i \to f(\lambda_j, \beta_{j0}, \beta_j, \alpha_i, W_i, X_i, y_{ij}, n_i) = \mathrm{L}(\theta_{ij})$$

— Calcule de $\text{logit}(\theta_{ij}) = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j$.

— Calcule de $\theta_{ij} = \dfrac{1}{1 + \exp\left(-\text{logit}(\theta_{ij})\right)}$.

— Retourne $\mathrm{L}(\theta_{ij}) = p(y_{ij} \mid \theta_{ij}, n_i) = \binom{n_i}{y_{ij}} (\theta_{ij})^{y_{ij}} (1 - \theta_{ij})^{n_i - y_{ij}}$.

On répète ces étapes pour $i = 1, \ldots, I$ et $j = 1, \ldots, J$, et on pose $\theta = (\theta ij)_{i=1,\ldots I}^{j=1,\ldots,J}$.

On peut ainsi calculer la vraisemblance du modèle : $\mathrm{L}(\theta) = \sum\limits_{\substack{1 \leq i \leq I \\ 1 \leq j \leq I}} \mathrm{L}(\theta_{ij})$.

D'après la formules de Bayes on a

$$\mathrm{p}(\theta \mid Y) \propto \pi(\theta)\mathrm{L}(\theta).$$

On utilise donc les relations suivantes pour approcher les densités conditionnelles a posteriori de chacun des paramètres avec $\pi(.)$ les densités correspondants à leurs lois *a priori*.

$$p(\beta_{jk} \mid \beta_{j0}, \beta_{j1}, \ldots, \beta_{jk-1}, \beta_{jk+1}, \ldots, \beta_{jp}, \lambda_j, \alpha_1, \ldots, \alpha_I, W_1, \ldots, W_I, Y) \propto \pi(\beta_{jk}) \prod_{1 \leq i \leq I} \mathrm{L}(\theta_{ij})$$

$$p(\lambda_{jl} \mid \lambda_{j1}, \ldots, \lambda_{jl-1}, \lambda_{jl+1}, \ldots, \lambda_{jq}, \beta_j, \beta_{j0}, \alpha_1, \ldots, \alpha_I, W_1, \ldots, W_I, Y) \propto \pi(\lambda_{jl}) \prod_{1 \leq i \leq I} \mathrm{L}(\theta_{ij})$$

$$p(W_{il} \mid W_{i1}, \ldots, W_{il-1}, W_{il+1}, \ldots, W_{iq}, \alpha_i, \beta_{10}, \ldots, \beta_{J0}, \beta_1, \ldots, \beta_J, \lambda_1, \ldots, \lambda_J, Y) \propto \pi(W_{il}) \prod_{1 \leq j \leq J} \mathrm{L}(\theta_{ij})$$

$$p(\alpha_i \mid W_i, \beta_{10}, \ldots, \beta_{J0}, \beta_1, \ldots, \beta_J, \lambda_1, \ldots, \lambda_j, V_\alpha, Y) \propto \pi(\alpha_i \mid V_\alpha) \prod_{1 \leq j \leq J} \mathrm{L}(\theta_{ij})$$

, pour $i = 1, \ldots, I$, $j = 1, \ldots, J$, $k = 1, \ldots, p$ et $l = 1, \ldots, q$.

L'algorithme utilisé pour estimer les paramètres du modèle logit est donc le suivant :

— Définition des constantes N_{Gibbs}, N_{burn}, N_{thin} et R_{opt} tels que N_{Gibbs} correspond au nombre d'itérations effectuées par l'algorithme, N_{burn} au nombre d'itérations nécessaires pour le burn-in ou temps de chauffe, $N_{samp} = \dfrac{N_{Gibbs} - N_{burn}}{N_{thin}}$ correspondant au nombre de valeurs estimées retenues pour chaque paramètre. En effet on enregistre les paramètres estimés à certaines itérations afin d'obtenir N_{samp} valeurs, nous permettant de représenter une distribution a posteriori pour chacun des paramètres.
On fixe R_{opt} le ratio d'acceptation optimal utilisé dans les algorithmes de Metropolis adaptatifs implémentés pour chacun des paramètres du modèle.

— Initialiser tous les paramètres à 0 par exemple, excepté les valeurs diagonales de Λ initialisées à 1 et $V_\alpha^{(0)} = 1$. Le nombre d'acceptation de chaque paramètre est intialisé à 0 et les variances de leur densités instrumententales conditionnelles prennent la valeur 1.

— Gibbs sampler : à chaque itération t pour $t = 1, \ldots, N_{Gibbs}$ on répète chacune de ces étapes :

- Générer les **effets sites aléatoires** $\alpha_i^{(t)}$ pour $i = 1, \ldots, I$ selon un algorithme de Metropolis adaptatif simulant $\alpha_i^\star \sim \mathcal{N}(\alpha_i^{(t-1)}, \sigma^2_{\alpha_i})$ puis calculant le taux d'acceptation de la manière suivante :

$$\gamma = min\left(1,\ \frac{\pi\left(\alpha_i^\star \mid V_\alpha^{(t-1)}\right) \prod\limits_{1\le j\le J} f\left(\alpha_i^\star, W_i^{(t-1)}, \beta_{j0}^{(t-1)}, \beta_j^{(t-1)}, \lambda_j^{(t-1)}, X_i, y_{ij}, n_i\right)}{\pi\left(\alpha_i^{(t-1)} \mid V_\alpha^{(t-1)}\right) \prod\limits_{1\le j\le J} f\left(\alpha_i^{(t-1)}, W_i^{(t-1)}, \beta_{j0}^{(t-1)}, \beta_j^{(t-1)}, \lambda_j^{(t-1)}, X_i, y_{ij}, n_i\right)}\right).$$

- Générer la **variance des effets site aléatoires** $V_\alpha^{(t)}$ selon :

$$V_\alpha^{(t)} \mid \alpha_1^{(t)}, \ldots, \alpha_I^{(t)} \sim \mathcal{IG}\left(\text{shape} = 0.5 + \frac{I}{2}, \text{rate} = 0.005 + \frac{1}{2}\sum_{i=1}^{I}\left(\alpha_i^{(t)}\right)^2\right)$$

- Générer les **prédicteurs non mesurés** (ou "variables latentes") $W_{il}^{(t)}$ pour $i = 1, \ldots, I$ et $l = 1, \ldots, q$ selon un algorithme de Metropolis adaptatif simulant $W_{il}^\star \sim \mathcal{N}(W_{il}^{(t-1)}, \sigma^2_{W_{il}})$ puis calculant le taux d'acceptation de la manière suivante :

$$\gamma = min\left(1,\ \frac{\pi\left(W_{il}^\star\right) \prod\limits_{1\le j\le J} f\left(W_{il}^\star, \alpha_i^{(t)}, \beta_{j0}^{(t-1)}, \beta_j^{(t-1)}, \lambda_j^{(t-1)}, X_i, y_{ij}, n_i\right)}{\pi\left(W_{il}^{(t-1)}\right) \prod\limits_{1\le j\le J} f\left(W_{il}^{(t-1)}, \alpha_i^{(t)}, \beta_{j0}^{(t-1)}, \beta_j^{(t-1)}, \lambda_j^{(t-1)}, X_i, y_{ij}, n_i\right)}\right).$$

- Générer les **effets espèces fixes** $\beta_{jk}^{(t)}$ pour $j = 1, \ldots, J$ et $k = 0, \ldots, p$ selon un algorithme de Metropolis adaptatif simulant $\beta_{jk}^\star \sim \mathcal{N}(\beta_{jk}^{(t-1)}, \sigma^2_{\beta_{jk}})$ puis calculant le taux d'acceptation de la manière suivante :

$$\gamma = min\left(1, \frac{\pi\left(\beta_{jk}^\star\right) \prod\limits_{1\le i\le I} f\left(\beta_{j0}^{(t)}, \ldots, \beta_{jk-1}^{(t)}, \beta_{jk}^\star, \beta_{jk+1}^{(t-1)}, \ldots, \beta_{jp}^{(t-1)}, \lambda_j^{(t-1)}, \alpha_1^{(t)}, W_1^{(t)}, \ldots, \alpha_I^{(t)}, W_I^{(t)}, X_i, y_{ij}, n_i\right)}{\pi\left(\beta_{jk}^{(t-1)}\right) \prod\limits_{1\le i\le I} f\left(\beta_{j0}^{(t)}, \ldots, \beta_{jk-1}^{(t)}, \beta_{jk}^{(t-1)}, \beta_{jk+1}^{(t-1)}, \ldots, \beta_{jp}^{(t-1)}, \lambda_j^{(t-1)}, \alpha_1^{(t)}, W_1^{(t)}, \ldots, \alpha_I^{(t)}, W_I^{(t)}, X_i, y_{ij}, n_i\right)}\right).$$

- Générer les **effets espèces fixes liés aux prédicteurs non mesurés** $\lambda_{jl}^{(t)}$ pour $j = 1, \ldots, J$ et $l = 1, \ldots, q$ selon un algorithme de Metropolis adaptatif pour $l \geq j$, simulant $\lambda_{jl}^\star \sim \mathcal{N}(\lambda_{jl}^{(t-1)}, \sigma^2_{\lambda_{jl}})$ puis calculant le taux d'acceptation de la manière suivante :

$$\gamma = min\left(1, \frac{\pi\left(\lambda_{jl}^\star\right) \prod\limits_{1\le i\le I} f\left(\lambda_{j1}^{(t)}, \ldots, \lambda_{jl-1}^{(t)}, \lambda_{jl}^\star, \lambda_{jl+1}^{(t-1)}, \ldots, \lambda_{jq}^{(t-1)}, \beta_{j0}^{(t)}, \beta_j^{(t)}, \alpha_1^{(t)}, W_1^{(t)}, \ldots, \alpha_I^{(t)}, W_I^{(t)}, X_i, y_{ij}, n_i\right)}{\pi\left(\lambda_{jl}^{(t-1)}\right) \prod\limits_{1\le i\le I} f\left(\lambda_{j1}^{(t)}, \ldots, \lambda_{jl-1}^{(t)}, \lambda_{jl}^{(t-1)}, \lambda_{jl+1}^{(t-1)}, \ldots, \lambda_{jq}^{(t-1)}, \beta_{j0}^{(t)}, \beta_j^{(t)}, \alpha_1^{(t)}, W_1^{(t)}, \ldots, \alpha_I^{(t)}, W_I^{(t)}, X_i, y_{ij}, n_i\right)}\right).$$

  Dans le cas $l > j$, on pose $\lambda_{jl}^{(t)} = 0$.

2.4 Evaluation de la fiabilité de ces méthodes sur des données simulées

2.4.1 Simulation des données

On simule pour chacun des sites i deux variables bioclimatiques selon une loi normale centrée réduite et on note $X_i = (1, X_{i1}, X_{i2})$ pour $i = 1, \dots, 500$. On considérera deux prédicteurs non mesurés.
On fixe les paramètres suivants afin de simuler des données de présence absence pour les 100 espèces et 500 sites considérés selon les deux modèles définis précédemment.
Les effets espèces fixes générés selon des lois uniformes :

$$\beta_{jk} \sim \mathcal{U}(-2, 2) \text{ pour } j = 1, \dots, 100 \text{ et } k = 0, 1, 2.$$

$$\lambda_{jl} \begin{cases} \sim \mathcal{U}(-2, 2) & \text{si } l < j, \\ \sim \mathcal{U}(0, 2) & \text{si } l = j, \\ = 0 & \text{si } l > j, \end{cases}$$

$$\text{pour } j = 1, \dots, 100 \text{ et } l = 1, 2.$$

Les prédicteurs non mesurés $W_{il} \sim \mathcal{N}(0, 1)$ pour $i = 1, \dots, 500$ et $l = 1, 2$.

Les effets sites aléatoires générés selon $\alpha_i \sim \mathcal{N}(0, V_{alpha})$ avec V_{alpha} fixé à 0.5.

Puis on simule les données de présence absence $Y = (y_{ij})_{i=1,\dots,500}^{j=1,\dots,100}$ de la manière apropriée pour chacun des modèles envisagés en fonction des paramètres établis :

Modèle probit :
Pour $i = 1, \dots, 500$ et $j = 1, \dots, 100$:
Générer $\epsilon_{ij} \sim \mathcal{N}(0, 1)$ *iid.*
Calculer $Z_{ij} = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j + \epsilon_{ij}$
Simuler $y_{ij} = \begin{cases} 1 & \text{si } Z_{ij} > 0 \\ 0 & \text{sinon.} \end{cases}$

Modèle logit :
Pour $i = 1, \dots, 500$ et $j = 1, \dots, 100$:
Calculer $\text{logit}(\theta_{ij}) = \alpha_i + \beta_{j0} + X_i\beta_j + W_i\lambda_j$.
Calculer $\theta_{ij} = \dfrac{1}{1 + \exp(-\text{logit}(\theta_{ij}))}$.
Simuler $y_{ij} \sim \mathcal{B}ernoulli(\theta_{ij})$.

2.4.2 Représentation des paramètres estimés

Modèle probit :

Figure 1 – *Représentation des $(\alpha_i)_{i=1,\dots,I}$ estimés en fonction de ceux simulés pour le modèle probit*

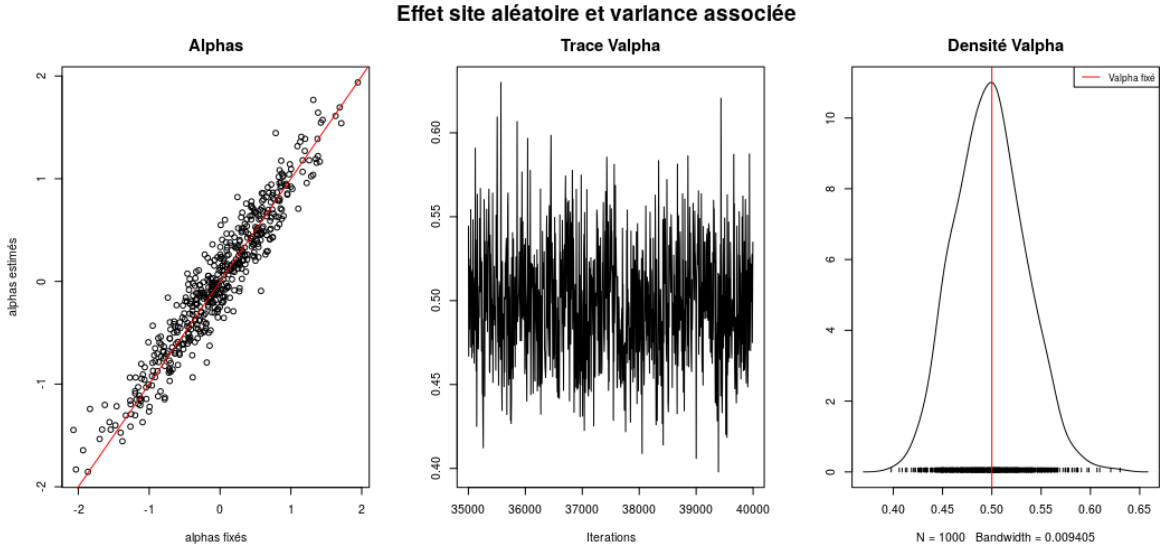


Figure 2 – *Représentation des* $(W_{il})_{i=1,\ldots,500}^{l=1,2}$ *estimés en fonction de ceux simulés pour le modèle probit*

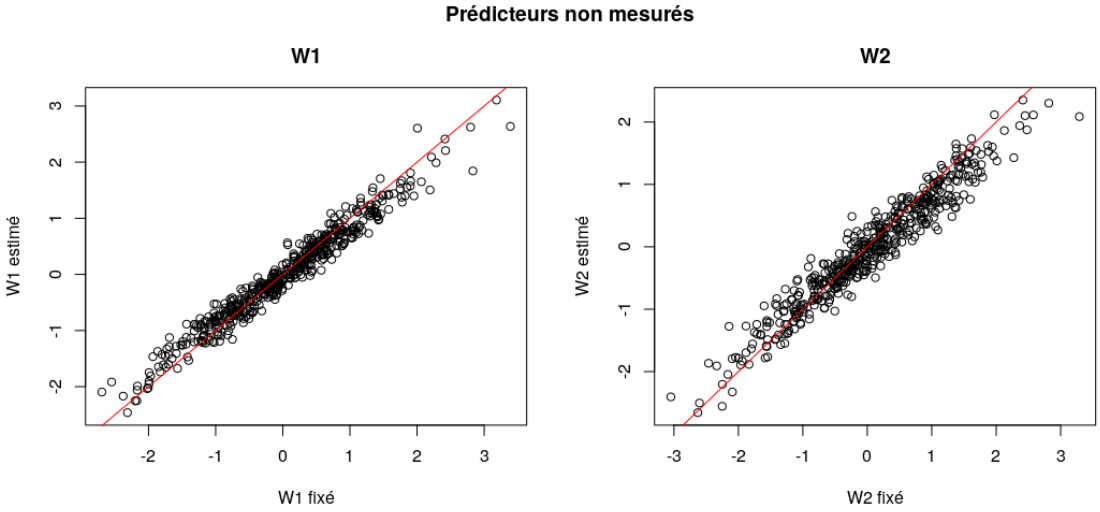


Figure 3 – *Représentation des* $(\beta_{jk})_{j=1,\ldots,100}^{k=1,2,3}$ *et* $(\lambda_{jl})_{j=1,\ldots,100}^{l=1,2}$ *estimés en fonction de ceux simulés pour le modèle probit*

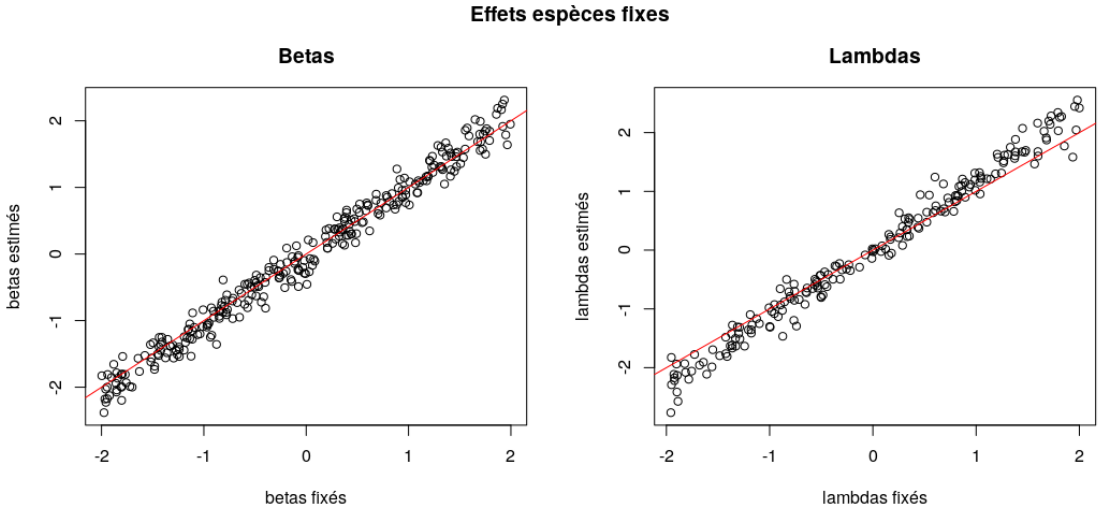
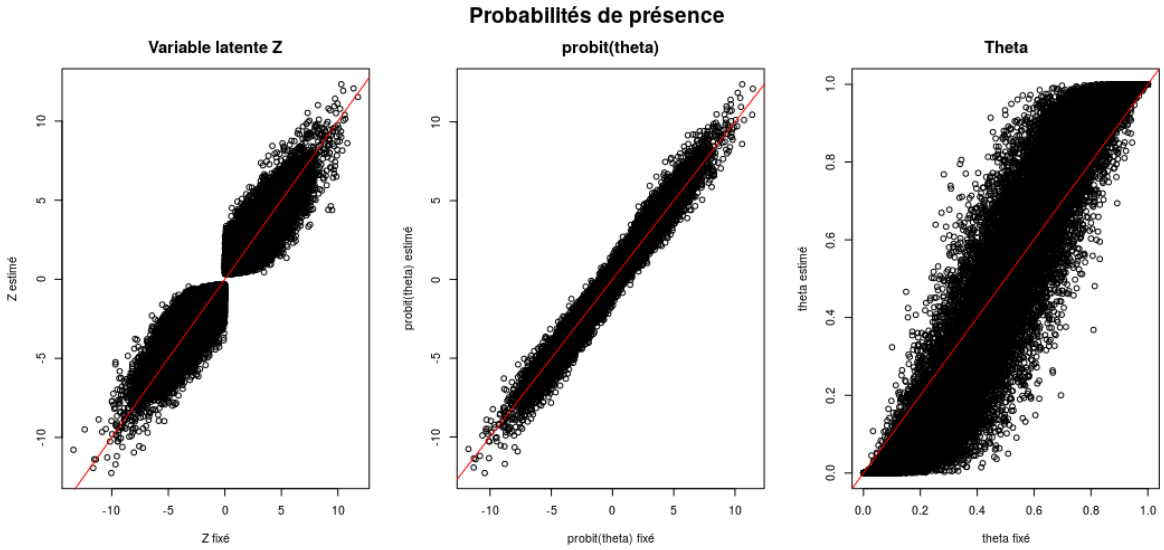


Figure 4 – *Représentation des* $(\text{probit}(\theta_{ij}))_{i=1,\dots,500}^{j=1,\dots,100}$, $(Z_{ij})_{i=1,\dots,500}^{j=1,\dots,100}$ *et* $(\theta_{ij})_{i=1,\dots,500}^{j=1,\dots,100}$ *estimés en fonction de ceux simulés pour le modèle probit*



Modèle logit :

Figure 5 – *Représentation des* $(\alpha_i)_{i=1,\dots,I}$ *estimés en fonction de ceux simulés pour le modèle logit*

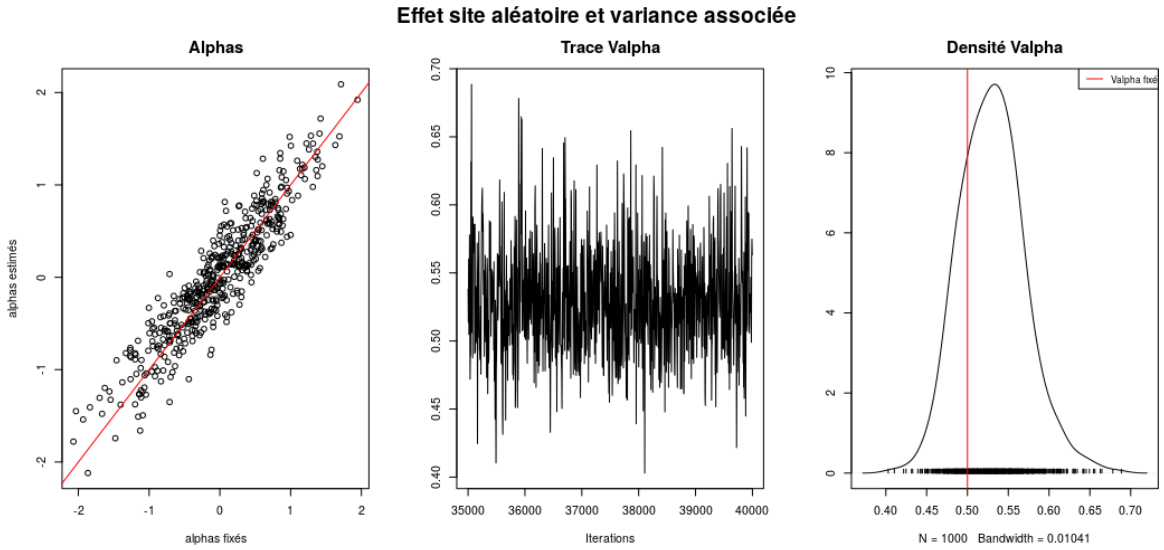


Figure 6 – *Représentation des* $(W_{il})_{i=1,\ldots,500}^{l=1,2}$ *estimés en fonction de ceux simulés pour le modèle logit*

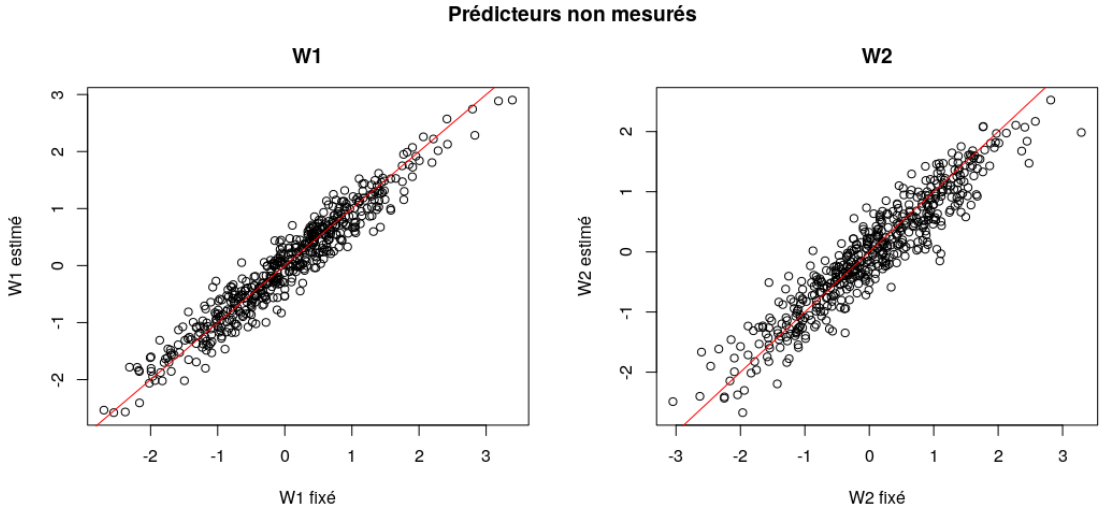


Figure 7 – *Représentation des* $(\beta_{jk})_{j=1,\ldots,100}^{k=1,2,3}$ *et* $(\lambda_{jl})_{j=1,\ldots,100}^{l=1,2}$ *estimés en fonction de ceux simulés pour le modèle logit*

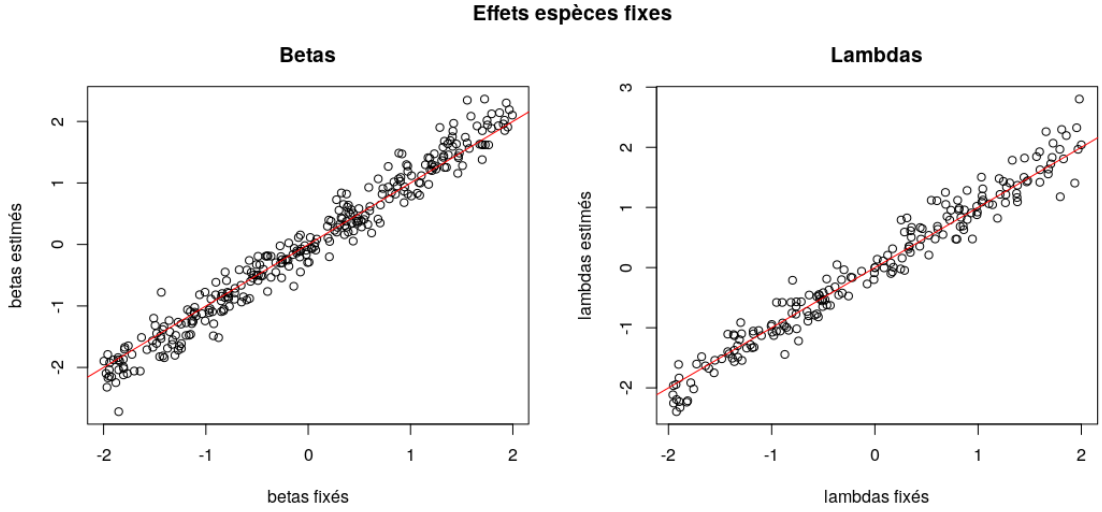
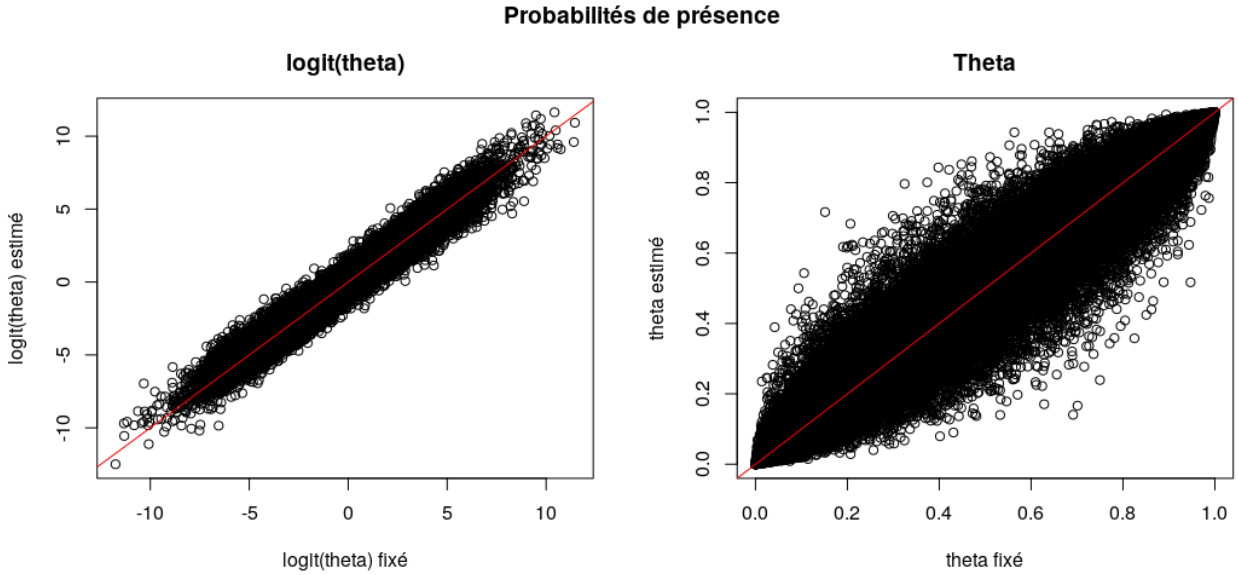


Figure 8 – *Représentation des* $(\text{logit}(\theta_{ij}))_{i=1,\dots,500}^{j=1,\dots,100}$ *et* $(\theta_{ij})_{i=1,\dots,500}^{j=1,\dots,100}$ *estimés en fonction de ceux simulés pour le modèle probit*



2.4.3 Evaluation du temps de calul et de la pertinence des résultats

tableau deviance RMSE temps de calcul + formules expliquer plus haut nombre de paramètres à estimer et moyenne des échantillons de NSAMP valeurs.

3 Application aux données collectées à Madagascar

3.1 Description des données

On dispose d'inventaires forestiers réalisés sur différents sites de l'île de Madagascar (plot sites).

3.2 Estimation des paramètres

3.3 Prédictions par interpolation

3.4 Prédictions avec auto-corrélation spatiale

3.5 Analyse des résultats et mise en évidence de lieux refuges de la biodiversité

Conclusion

Bibliographie

Albert, James H., and Chib Siddhartha. 1993. “Bayesian Analysis of Binary and Polychotomous Response Data.” *Journal of the American Statistical Association* 88 (422) : 669–79. doi :10.1080/01621459.1993.10476321.

Marine Guillerm, Pauline Givord et. 2016. “Les Modèles Multiniveaux.” *Méthodologie Statistique INSEE*.

Warton, David I., F. Guillaume Blanchet, Robert B. O’Hara, Otso Ovaskainen, Sara Taskinen, Steven C. Walker, and Francis K.C. Hui. 2015. “So Many Variables : Joint Modeling in Community Ecology.” *Trends in Ecology & Evolution* 30 (12). Elsevier : 766–79. doi :10.1016/j.tree.2015.09.007.

Annexes

Fonction utilisée pour le modèle probit

```
#include <RcppArmadillo.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_cdf.h>
#include <cmath>
#include "Rcpp_jSDM_useful.h"

// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(RcppGSL)]]

using namespace arma;
using namespace std;

/* ************************************************************ */
/* Gibbs sampler function */

// [[Rcpp::export]]
Rcpp::List Rcpp_jSDM_probit_block(const int ngibbs, int nthin, int nburn,
                                  arma::umat Y,
                                  arma::umat T,
                                  arma::mat X,
                                  arma::mat param_start,
                                  arma::mat Vparam,
                                  arma::vec muparam,
                                  arma::mat VW,
                                  arma::mat W_start,
                                  arma::vec alpha_start,
                                  double Valpha_start,
                                  double shape,
                                  double rate,
                                  const int seed,
                                  const int verbose) {

  ////////////////////////////////////
 // Defining and initializing objects//

  // Initialize random number generator
```

```
gsl_rng *s = gsl_rng_alloc(gsl_rng_mt19937);
gsl_rng_set(s, seed);

// Redefining constants
const int NGIBBS = ngibbs;
const int NTHIN = nthin;
const int NBURN = nburn;
const int NSAMP = (NGIBBS-NBURN)/NTHIN;
const int NSITE = Y.n_rows;
const int NP = X.n_cols;
const int NSP = Y.n_cols;
const int NL = W_start.n_cols;

////////////////////////////////////////////
// Declaring new objects to store results //
/* Parameters */
arma::Cube<double> param; param.zeros(NSAMP, NSP, NP+NL);
arma::Cube<double> W; W.zeros(NSAMP, NSITE, NL);
arma::mat alpha; alpha.zeros(NSAMP, NSITE);
arma::vec Valpha; Valpha.zeros(NSAMP);
/* Latent variable */
arma::mat probit_theta_pred; probit_theta_pred.zeros(NSITE, NSP);
arma::mat Z_latent; Z_latent.zeros(NSITE, NSP);
/* Deviance */
arma::vec Deviance; Deviance.zeros(NSAMP);

 ////////////////////////////////////
// Initializing running parameters //

//  mat of species effects parameters and coefficients for latent variables (nl+np,nsp)
arma::mat param_run = param_start;
// alpha vec of sites effects (nsite)
arma::vec alpha_run = alpha_start;
double Valpha_run = Valpha_start;
// w latent variables (nsite*nl)
arma::mat W_run = W_start;
// Z latent (nsite*nsp)
arma::mat Z_run; Z_run.zeros(NSITE,NSP);
// probit_theta_ij = X_i*beta_j + W_i*lambda_j + alpha_i
arma::mat probit_theta_run; probit_theta_run.zeros(NSITE,NSP);
```

```
// data
arma::mat data = arma::join_rows(X,W_run);

////////////
// Message//
Rprintf("\nRunning the Gibbs sampler. It may be long, please keep cool :)\n\n");
R_FlushConsole();

////////////////////////////////////////////////////////////////////////////////////////
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// Gibbs sampler

for (int g=0; g < NGIBBS; g++) {

  ////////////////////////
  // latent variable Z //

  for (int j=0; j<NSP; j++) {
    for (int i=0; i<NSITE; i++) {
      // Actualization
      if (Y(i,j) == 0) {
        Z_run(i,j) = rtnorm(s, R_NegInf, 0, probit_theta_run(i,j), 1);
      } else {
        Z_run(i,j) = rtnorm(s, 0, R_PosInf, probit_theta_run(i,j), 1);
      }
    } // loop on sites
  }// loop on species

  ///////////////////////////////////
  // mat param: Gibbs algorithm //

  // Loop on species
  for (int j=0; j<NSP; j++) {
    // small_v
    arma::vec small_v = inv(Vparam)*muparam + data.t()*(Z_run.col(j) - alpha_run);
    // big_V
    arma::mat big_V = inv(inv(Vparam)+data.t()*data);
    // Draw in the posterior distribution
    arma::vec param_prop = arma_mvgauss(s, big_V*small_v, chol_decomp(big_V));
```

```
  // constraints on lambda
  for (int l=0; l<NL; l++) {
    if (l > j) {
      param_prop(NP+l) = 0;
    }
    if ((l==j) & (param_prop(NP+l) < 0)) {
      param_prop(NP+l) = param_run(NP+l,j);
    }
  }
  param_run.col(j) = param_prop;
}// loop on species


////////////////////////////////////////////
// mat latent variable W: Gibbs algorithm //

// Loop on sites
for (int i=0; i<NSITE; i++) {
  arma::mat beta_run = param_run.submat(0,0,NP-1,NSP-1);
  arma::mat lambda_run = param_run.submat(NP,0,NP+NL-1,NSP-1);
  // big_V
  arma::mat big_V = inv(inv(VW)+lambda_run*lambda_run.t());

  // small_v
  arma::vec small_v =lambda_run*(Z_run.row(i)-X.row(i)*beta_run-alpha_run(i)).t();

  // Draw in the posterior distribution
  arma::vec W_i = arma_mvgauss(s, big_V*small_v, chol_decomp(big_V));
  W_run.row(i) = W_i.t();
}

data = arma::join_rows(X, W_run);

////////////////////////////////////
// vec alpha : Gibbs algorithm //

// Loop on sites
double sum = 0.0;
for (int i=0; i<NSITE; i++) {
  // small_v
```

```
    double small_v = arma::sum(Z_run.row(i)-data.row(i)*param_run);
    // big_V
    double big_V = 1/(1/Valpha_run + NSP);

    // Draw in the posterior distribution
    alpha_run(i) = big_V*small_v + gsl_ran_gaussian_ziggurat(s, std::sqrt(big_V));
    sum += alpha_run(i)*alpha_run(i);
  }

  ////////////////////////////////////////////////
  // Valpha
  double shape_posterior = shape + 0.5*NSITE;
  double rate_posterior = rate + 0.5*sum;

  Valpha_run = rate_posterior/gsl_ran_gamma_mt(s, shape_posterior, 1.0);

   //////////////
  // Deviance //

  // logLikelihood
  double logL = 0.0;
  for ( int i = 0; i < NSITE; i++ ) {
    for ( int j = 0; j < NSP; j++ ) {
      // probit(theta_ij) = X_i*beta_j + W_i*lambda_j + alpha_i
      probit_theta_run(i,j) = arma::as_scalar(data.row(i)*param_run.col(j) + alpha_run(i));
      // link function probit is the inverse of N(0,1) repartition function
      double theta = gsl_cdf_ugaussian_P(probit_theta_run(i,j));
      /* log Likelihood */
      logL += R::dbinom(Y(i,j), T(i,j), theta, 1);
    } // loop on species
  } // loop on sites

  // Deviance
  double Deviance_run = -2 * logL;

  ////////////////////////////////////////////////
  // Output
  if (((g+1)>NBURN) && (((g+1)%(NTHIN))==0)) {
    int isamp=((g+1)-NBURN)/(NTHIN);
    for ( int j=0; j<NSP; j++ ) {
```

```
      param.tube(isamp-1,j) = param_run.col(j);
      for ( int i=0; i<NSITE; i++ ) {
        W.tube(isamp-1,i) = W_run.row(i);
        Z_latent(i,j) += Z_run(i,j) / NSAMP; // We compute the mean of NSAMP values
        probit_theta_pred(i,j) += probit_theta_run(i,j)/NSAMP;
      }
    }
    alpha.row(isamp-1) = alpha_run.t();
    Valpha(isamp-1) = Valpha_run;
    Deviance(isamp-1) = Deviance_run;
  }

  ////////////////////////////////////////////////
  // Progress bar
  double Perc=100*(g+1)/(NGIBBS);
  if (((g+1)%(NGIBBS/100))==0 && (verbose==1)) {
    Rprintf("*");
    R_FlushConsole();
    //R_ProcessEvents(); for windows
    if (((g+1)%(NGIBBS/10))==0) {
      Rprintf(":%.1f%% \n",Perc);
      R_FlushConsole();
      //R_ProcessEvents(); for windows
    }
  }

  ////////////////////
// User interrupt //
  R_CheckUserInterrupt(); // allow user interrupts

} // Gibbs sampler

// Free memory
gsl_rng_free(s);

// Return results as a Rcpp::List
Rcpp::List results = Rcpp::List::create(Rcpp::Named("param") = param,
                                        Rcpp::Named("W") = W,
                                        Rcpp::Named("alpha") = alpha,
                                        Rcpp::Named("Valpha") = Valpha,
```

```
                                        Rcpp::Named("Deviance") = Deviance,
                                        Rcpp::Named("Z_latent") = Z_latent,
                                        Rcpp::Named("probit_theta_pred") = probit_theta_pred);
  return results;

} // end Rcpp_jSDM_probit_block
```

Fonction utilisée pour le modèle logit

```
#include <RcppArmadillo.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>

// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::depends(RcppGSL)]]

/* Function logit */
double logit (double x) {
  return std::log(x) - std::log(1-x);
}

/* Function invlogit */
double invlogit (double x) {
  if (x > 0) {
    return 1 / (1 + std::exp(-x));
  }
  else {
    return std::exp(x) / (1 + std::exp(x));
  }
}

/* Function mean */
double mean (arma::vec x){
  int n = x.size();
  double sum = 0.0;
  for ( int i = 0; i < n ; i++ ) {
    sum += x(i);
  }
  return sum/n ;
}
```

```
/* Function var */
double var (arma::vec x){
  int n = x.size();
  double sum = 0.0;
  for ( int i = 0; i < n ; i++ ) {
    sum += (x(i) - mean(x))*(x(i) - mean(x));
  }
  return sum/n ;
}

/* dens_par */
struct dens_par {
  // Data
  int NSITE;
  int NSP;
  arma::umat Y;
  arma::umat T;
  // Suitability
  // beta
  int NP;
  arma::mat X;
  int pos_beta;
  int sp_beta;
  arma::mat mubeta;
  arma::mat Vbeta;
  arma::mat beta_run;
  // lambda
  int NL;
  int pos_lambda;
  int sp_lambda;
  arma::mat mulambda;
  arma::mat Vlambda;
  arma::mat lambda_run;
  // W
  int site_W;
  int pos_W;
  arma::mat muW;
  arma::mat VW;
  arma::mat W_run;
```

```
  //alpha
  int site_alpha;
  double Valpha_run;
  double shape;
  double rate;
  arma::rowvec alpha_run;
};

/* betadens */
double betadens (double beta_jk, void *dens_data) {
  // Pointer to the structure: d
  dens_par *d;
  d = static_cast<dens_par *> (dens_data);
  // Indicating the rank and the species of the parameter of interest
  int k = d->pos_beta;
  int j = d->sp_beta;
  // logLikelihood
  double logL = 0.0;
  for ( int i = 0; i < d->NSITE; i++ ) {
    /* theta */
    double Xpart_theta = 0.0;
    for ( int p = 0; p < d->NP; p++ ) {
      if ( p != k ) {
        Xpart_theta += d->X(i,p) * d->beta_run(j,p);
      }
    }
    for ( int q = 0; q < d->NL; q++ ) {
      Xpart_theta += d->W_run(i,q) * d->lambda_run(j,q);
    }
    Xpart_theta += d->X(i,k) * beta_jk + d->alpha_run(i);
    double theta = invlogit(Xpart_theta);
    /* log Likelihood */
    logL += R::dbinom(d->Y(i,j), d->T(i,j), theta, 1);
  } // loop on sites

  // logPosterior = logL + logPrior
  double logP = logL + R::dnorm(beta_jk, d->mubeta(j,k), std::sqrt(d->Vbeta(j,k)), 1);
  return logP;
}
```

```
/* lambdadens */
double lambdadens (double lambda_jq, void *dens_data) {
  // Pointer to the structure: d
  dens_par *d;
  d = static_cast<dens_par *> (dens_data);
  // Indicating the rank and the species of the parameter of interest
  int q = d->pos_lambda;
  int j = d->sp_lambda;
  // logLikelihood
  double logL = 0.0;
  for ( int i = 0; i < d->NSITE; i++ ) {
    /* theta */
    double Xpart_theta = 0.0;
    for ( int p = 0;  p < d->NP; p++ ) {
      Xpart_theta += d->X(i,p) * d->beta_run(j,p);
    }
    for ( int l = 0;  l < d->NL; l++ ) {
      if(l != q) {
        Xpart_theta += d->W_run(i,l)* d->lambda_run(j,l);
      }
    }
    Xpart_theta += d->W_run(i,q) * lambda_jq +  d->alpha_run(i);
    double theta = invlogit(Xpart_theta);
    /* log Likelihood */
    logL += R::dbinom(d->Y(i,j), d->T(i,j), theta, 1);
  } // loop on sites

  // logPosterior = logL + logPrior
  double logP = logL + R::dnorm(lambda_jq, d->mulambda(j,q), std::sqrt(d->Vlambda(j,q)), 1);
  return logP;
}

double lambdaUdens (double lambda_jq, void *dens_data) {
  // Pointer to the structure: d
  dens_par *d;
  d = static_cast<dens_par *> (dens_data);
  // Indicating the rank and the species of the parameter of interest
  int q = d->pos_lambda;
  int j = d->sp_lambda;
  // logLikelihood
```

```
  double logL = 0.0;
  for ( int i = 0; i < d->NSITE; i++ ) {
    /* theta */
    double Xpart_theta = 0.0;
    for ( int p = 0;  p < d->NP; p++ ) {
      Xpart_theta += d->X(i,p) * d->beta_run(j,p);
    }
    for ( int l = 0;  l < d->NL; l++ ) {
      if(l != q) {
        Xpart_theta += d->W_run(i,l)* d->lambda_run(j,l);
      }
    }
    Xpart_theta += d->W_run(i,q) * lambda_jq + d->alpha_run(i);
    double theta = invlogit(Xpart_theta);
    /* log Likelihood */
    logL += R::dbinom(d->Y(i,j), d->T(i,j), theta, 1);
  } // loop on sites

  // logPosterior = logL + logPrior
  double logP = logL + R::dunif(lambda_jq, d->mulambda(j,q), d->Vlambda(j,q), 1);
  return logP;
}

/* Wdens */
double Wdens (double W_iq, void *dens_data) {
  // Pointer to the structure: d
  dens_par *d;
  d = static_cast<dens_par *> (dens_data);
  // Indicating the rank and the species of the parameter of interest
  int i = d->site_W;
  int q = d->pos_W;
  // logLikelihood
  double logL = 0.0;
  for ( int j = 0; j < d->NSP; j++ ) {
    /* theta */
    double Xpart_theta = 0.0;
    for ( int p = 0;  p < d->NP; p++ ) {
      Xpart_theta += d->X(i,p) * d->beta_run(j,p);
    }
    for ( int l = 0;  l < d->NL; l++ ) {
```

```
      if(l != q) {
        Xpart_theta += d->W_run(i,l)* d->lambda_run(j,l);
      }
    }
    Xpart_theta += W_iq * d->lambda_run(j,q) +  d->alpha_run(i);
    double theta = invlogit(Xpart_theta);
    /* log Likelihood */
    logL += R::dbinom(d->Y(i,j), d->T(i,j), theta, 1);
  } // loop on species

  // logPosterior = logL + logPrior
  double logP = logL + R::dnorm(W_iq, d->muW(i,q), std::sqrt(d->VW(i,q)), 1);
  return logP;
}

/* alphadens */

double alphadens (double alpha_i, void *dens_data) {
  // Pointer to the structure: d
  dens_par *d;
  d = static_cast<dens_par *> (dens_data);
  // Indicating the site of the parameter of interest
  int i = d->site_alpha;
  // logLikelihood
  double logL = 0.0;
  /* theta */
  for ( int j = 0; j < d->NSP; j++ ) {
    double Xpart_theta = 0.0;
    for ( int p = 0; p < d->NP; p++ ) {
      Xpart_theta += d->X(i,p) * d->beta_run(j,p);
    }
    for ( int q = 0; q < d->NL; q++ ) {
      Xpart_theta += d->W_run(i,q) * d->lambda_run(j,q);
    }
    Xpart_theta += alpha_i;
    double theta = invlogit(Xpart_theta);
    /* log Likelihood */
    logL += R::dbinom(d->Y(i,j), d->T(i,j), theta, 1);
  } // loop on species
```

```
  // logPosterior = logL + logPrior
  double logP = logL + R::dnorm(alpha_i, 0, std::sqrt(d->Valpha_run),1);
  return logP;
}

/* ************************ */
/* Gibbs sampler function */

// [[Rcpp::export]]
Rcpp::List  Rcpp_jSDM_gibbs_logit(
    const int ngibbs, int nthin, int nburn, // Number of iterations, burning and samples
    arma::umat Y, // Number of successes (presences)
    arma::umat T, // Number of trials
    arma::mat X, // Suitability covariates
    arma::mat beta_start,//beta
    arma::mat mubeta,
    arma::mat Vbeta,
    arma::mat mulambda,//lambda
    arma::mat Vlambda,
    arma::mat lambda_start,
    arma::mat muW,//W
    arma::mat VW,
    arma::mat W_start,
    arma::vec alpha_start,//alpha
    double Valpha_start,
    double shape,
    double rate,
    const int seed,
    const double ropt,
    const int verbose) {

  ////////////////////////////////////////
  // Defining and initializing objects //

  ////////////////////////////////////////
  // Initialize random number generator //
  gsl_rng *r = gsl_rng_alloc(gsl_rng_mt19937);
  gsl_rng_set(r, seed);

  ////////////////////////////
```

```
// Redefining constants //
const int NGIBBS = ngibbs;
const int NTHIN = nthin;
const int NBURN = nburn;
const int NSAMP = (NGIBBS-NBURN)/NTHIN;
const int NSITE = X.n_rows;
const int NP = X.n_cols;
const int NSP = Y.n_cols;
const int NL = lambda_start.n_cols;

//////////////////////////////////////////
// Declaring new objects to store results //
/* Parameters */
arma::Cube<double> beta; beta.zeros(NSAMP, NSP, NP);
arma::Cube<double> lambda; lambda.zeros(NSAMP, NSP, NL);
arma::Cube<double> W; W.zeros(NSAMP, NSITE, NL);
arma::mat alpha; alpha.zeros(NSAMP, NSITE);
arma::vec Valpha; Valpha.zeros(NSAMP);
/* Latent variable */
arma::mat theta_run; theta_run.zeros(NSITE, NSP);
arma::mat theta_latent; theta_latent.zeros(NSITE, NSP);
/* Deviance */
arma::vec Deviance; Deviance.zeros(NSAMP);

////////////////////////////////////////////////////////
// Set up and initialize structure for density function //
dens_par dens_data;
// Data
dens_data.NSITE = NSITE;
dens_data.NSP = NSP;
dens_data.NL = NL;
// Y
dens_data.Y = Y;
// T
dens_data.T = T;
// Suitability process
dens_data.NP = NP;
dens_data.X = X;
// beta
dens_data.pos_beta = 0;
```

```
dens_data.sp_beta = 0;
dens_data.mubeta = mubeta;
dens_data.Vbeta = Vbeta;
dens_data.beta_run = beta_start;
// lambda
dens_data.pos_lambda = 0;
dens_data.sp_lambda = 0;
dens_data.mulambda = mulambda;
dens_data.Vlambda = Vlambda;
dens_data.lambda_run = lambda_start;
// W
dens_data.site_W = 0;
dens_data.pos_W = 0;
dens_data.muW = muW;
dens_data.VW = VW;
dens_data.W_run = W_start;
// alpha
dens_data.site_alpha = 0;
dens_data.shape = shape;
dens_data.rate = rate;
dens_data.Valpha_run = Valpha_start;
dens_data.alpha_run = alpha_start.t();

////////////////////////////////////////////////////////////
// Proposal variance and acceptance for adaptive sampling //

// beta
arma::mat sigmap_beta; sigmap_beta.ones(NSP,NP);
arma::mat nA_beta; nA_beta.zeros(NSP,NP);
arma::mat Ar_beta; Ar_beta.zeros(NSP,NP); // Acceptance rate

// lambda
arma::mat sigmaq_lambda; sigmaq_lambda.ones(NSP,NL);
arma::mat nA_lambda; nA_lambda.zeros(NSP,NL);
arma::mat Ar_lambda; Ar_lambda.zeros(NSP,NL); // Acceptance rate

// W
arma::mat sigmaq_W; sigmaq_W.ones(NSITE,NL);
arma::mat nA_W; nA_W.zeros(NSITE,NL);
arma::mat Ar_W; Ar_W.zeros(NSITE,NL); // Acceptance rate
```

```
// alpha
arma::vec sigma_alpha; sigma_alpha.ones(NSITE);
arma::vec nA_alpha; nA_alpha.zeros(NSITE);
arma::vec Ar_alpha; Ar_alpha.zeros(NSITE); // Acceptance rate


////////////
// Message//
Rprintf("\nRunning the Gibbs sampler. It may be long, please keep cool :)\n\n");
R_FlushConsole();

////////////////////
// Gibbs sampler //

for ( int g = 0; g < NGIBBS; g++ ) {

  double sum = 0.0;
  for ( int i = 0; i < NSITE; i++ ) {
    // alpha
    dens_data.site_alpha = i; // Specifying the site
    double x_now = dens_data.alpha_run(i);
    double x_prop = x_now + gsl_ran_gaussian_ziggurat(r, sigma_alpha(i));
    double p_now = alphadens(x_now, &dens_data);
    double p_prop = alphadens(x_prop, &dens_data);
    double ratio = std::exp(p_prop - p_now); // ratio
    double z = gsl_rng_uniform(r);
    // Actualization
    if ( z < ratio ) {
      dens_data.alpha_run(i) = x_prop;
      nA_alpha(i)++;
    }
    sum += dens_data.alpha_run(i)*dens_data.alpha_run(i);

    // W
    dens_data.site_W = i; // Specifying the site
    for ( int q = 0; q < NL; q++ ) {
      dens_data.pos_W = q; // Specifying the rank of the latent variable of interest
      double x_now = dens_data.W_run(i,q);
      double x_prop = x_now + gsl_ran_gaussian_ziggurat(r,sigmaq_W(i,q));
```

```
      double p_now = Wdens(x_now, &dens_data);
      double p_prop = Wdens(x_prop, &dens_data);
      double ratio = std::exp(p_prop - p_now); // ratio
      double z = gsl_rng_uniform(r);
      // Actualization
      if ( z < ratio ) {
        dens_data.W_run(i,q) = x_prop;
        nA_W(i,q)++;
      }
    } // loop on rank of latent variable
  } // loop on sites

  // Valpha
  double shape_posterior = dens_data.shape + 0.5*NSITE;
  double rate_posterior = dens_data.rate + 0.5*sum;

  dens_data.Valpha_run = rate_posterior/gsl_ran_gamma_mt(r, shape_posterior, 1.0);

  // Centering and reducing W_i
  for ( int i = 0; i < NSITE; i++ ) {
    for ( int q = 0; q < NL; q++ ) {
      dens_data.W_run(i,q) = dens_data.W_run(i,q) - mean(dens_data.W_run.col(q));
      dens_data.W_run(i,q) = dens_data.W_run(i,q)/std::sqrt(var(dens_data.W_run.col(q)));
    }
  }

  for ( int j = 0; j < NSP; j++ ) {
    // beta
    dens_data.sp_beta = j; // Specifying the species
    for ( int p = 0; p < NP; p++ ) {
      dens_data.pos_beta = p; // Specifying the rank of the parameter of interest
      double x_now = dens_data.beta_run(j,p);
      double x_prop = x_now + gsl_ran_gaussian_ziggurat(r, sigmap_beta(j,p));
      double p_now = betadens(x_now, &dens_data);
      double p_prop = betadens(x_prop, &dens_data);
      double ratio = std::exp(p_prop - p_now); // ratio
      double z = gsl_rng_uniform(r);
      // Actualization
      if ( z < ratio ) {
        dens_data.beta_run(j,p) = x_prop;
```

```
        nA_beta(j,p)++;
      }
    } // loop on rank of parameters

    // lambda
    dens_data.sp_lambda = j; // Specifying the species
    for ( int q = 0; q < NL; q++ ) {
      dens_data.pos_lambda = q ; // Specifying the rank of the parameter of interest
      if (q < j ) {
        double x_now = dens_data.lambda_run(j,q);
        double x_prop = x_now + gsl_ran_gaussian_ziggurat(r, sigmaq_lambda(j,q));
        double p_now = lambdadens(x_now, &dens_data);
        double p_prop = lambdadens(x_prop, &dens_data);
        double ratio = std::exp(p_prop - p_now); // ratio
        double z = gsl_rng_uniform(r);
        // Actualization
        if ( z < ratio ) {
          dens_data.lambda_run(j,q) = x_prop;
          nA_lambda(j,q)++;
        }
      }
      if (q == j) {
        double x_now = dens_data.lambda_run(j,q);
        double x_prop = x_now + gsl_ran_gaussian_ziggurat(r,sigmaq_lambda(j,q));
        double p_now = lambdaUdens(x_now, &dens_data);
        double p_prop = lambdaUdens(x_prop, &dens_data);
        double ratio = std::exp(p_prop - p_now); // ratio
        double z = gsl_rng_uniform(r);
        // Actualization
        if ( z < ratio ) {
          dens_data.lambda_run(j,q) = x_prop;
          nA_lambda(j,q)++;
        }
      }
      if (q > j) {
        dens_data.lambda_run(j,q) = 0;
      }
    } // loop on rank of latent variable
  } // loop on species
```

```
  ///////////////
  // Deviance //

  // logLikelihood
  double logL = 0.0;
  for ( int i = 0; i < NSITE; i++ ) {
    for ( int j = 0; j < NSP; j++ ) {
      /* theta */
      double Xpart_theta = 0.0;
      for ( int p = 0; p < NP; p++ ) {
        Xpart_theta += dens_data.X(i,p) * dens_data.beta_run(j,p);
      }
      for ( int q = 0; q < NL; q++ ) {
        Xpart_theta += dens_data.W_run(i,q) * dens_data.lambda_run(j,q);
      }
      Xpart_theta += dens_data.alpha_run(i);
      theta_run(i,j) = invlogit(Xpart_theta);
      /* log Likelihood */
      logL += R::dbinom(dens_data.Y(i,j), dens_data.T(i,j), theta_run(i,j), 1);
    } // loop on species
  } // loop on sites

  // Deviance
  double Deviance_run = -2 * logL;



 //////////////
// Output //
  if (((g+1)>NBURN) && (((g+1)%(NTHIN))==0)) {
    int isamp=((g+1)-NBURN)/(NTHIN);
    for ( int j=0; j<NSP; j++ ) {
      beta.tube(isamp-1,j) = dens_data.beta_run.row(j);
      lambda.tube(isamp-1,j) = dens_data.lambda_run.row(j);
      for ( int i=0; i<NSITE; i++ ) {
        W.tube(isamp-1,i) = dens_data.W_run.row(i);
        theta_latent(i,j) += theta_run(i,j) / NSAMP; // We compute the mean of NSAMP values
      }//loop on sites
    }// loop on species
    alpha.row(isamp-1) = dens_data.alpha_run;
    Valpha(isamp-1) = dens_data.Valpha_run;
```

```
  Deviance(isamp-1) = Deviance_run;
}

////////////////////////////////////////////
// Adaptive sampling (on the burnin period) //
const double ROPT = ropt;
int DIV = 0;
if ( NGIBBS >= 1000 ) DIV=100;
else DIV = NGIBBS / 10;
/* During the burnin period */
if ( (g+1)%DIV== 0 && (g+1)<=NBURN ) {
  for (int j=0; j<NSP; j++) {
    for ( int p=0; p<NP; p++ ) {
      Ar_beta(j,p) = ((double) nA_beta(j,p)) / DIV;
      if ( Ar_beta(j,p) >= ROPT )
        sigmap_beta(j,p) = sigmap_beta(j,p)*(2-(1-Ar_beta(j,p)) / (1-ROPT));
      else sigmap_beta(j,p) = sigmap_beta(j,p) / (2-Ar_beta(j,p) / ROPT);
      nA_beta(j,p) = 0.0; // We reinitialize the number of acceptance to zero for beta
    } // loop on rank of parameters
    for ( int q=0; q<NL; q++ ) {
      Ar_lambda(j,q) = ((double) nA_lambda(j,q)) / DIV;
      if ( Ar_lambda(j,q) >= ROPT )
        sigmaq_lambda(j,q) = sigmaq_lambda(j,q)*(2-(1-Ar_lambda(j,q)) / (1-ROPT));
      else sigmaq_lambda(j,q) = sigmaq_lambda(j,q) / (2-Ar_lambda(j,q) / ROPT);
      nA_lambda(j,q) = 0.0; // We reinitialize the number of acceptance to zero for lambda
    } // loop on rank of latent variable
  } // loop on species
  for (int i=0; i<NSITE; i++) {
    Ar_alpha(i) = ((double) nA_alpha(i)) / DIV;
    if ( Ar_alpha(i) >= ROPT ) sigma_alpha(i) = sigma_alpha(i) * (2-(1-Ar_alpha(i)) / (1-ROPT));
    else sigma_alpha(i) = sigma_alpha(i) / (2-Ar_alpha(i) / ROPT);
    nA_alpha(i) = 0.0; // We reinitialize the number of acceptance for alpha to zero
    for ( int q=0; q<NL; q++ ) {
      Ar_W(i,q) = ((double) nA_W(i,q)) / DIV;
      if ( Ar_W(i,q) >= ROPT ) sigmaq_W(i,q) = sigmaq_W(i,q) * (2-(1-Ar_W(i,q)) / (1-ROPT));
      else sigmaq_W(i,q) = sigmaq_W(i,q) / (2-Ar_W(i,q) / ROPT);
      nA_W(i,q) = 0.0; // We reinitialize the number of acceptance to zero for z
    } // loop on rank of latent variable
  } // loop on sites
}
```

```
/* After the burnin period */
if ( (g+1) % DIV == 0 && (g+1) > NBURN ) {
  for (int j=0; j<NSP; j++) {
    for (int p=0; p<NP; p++) {
      Ar_beta(j,p) = ((double) nA_beta(j,p)) / DIV;
      nA_beta(j,p) = 0.0; // We reinitialize the number of acceptance to zero for beta
    } // loop on rank of parameters
    for (int q=0; q<NL; q++) {
      Ar_lambda(j,q) = ((double) nA_lambda(j,q)) / DIV;
      nA_lambda(j,q) = 0.0; // We reinitialize the number of acceptance to zero for lambda
    } // loop on rank of latent variable
  } // loop on species
  for (int i=0; i<NSITE; i++) {
    Ar_alpha(i) = ((double) nA_alpha(i)) / DIV;
    nA_alpha(i) = 0.0; // We reinitialize the number of acceptance for alpha to zero
    for (int q=0; q<NL; q++) {
      Ar_W(i,q) = ((double) nA_W(i,q)) / DIV;
      nA_W(i,q) = 0.0; // We reinitialize the number of acceptance to zero for z
    } // loop on rank of latent variable
  } // loop on sites
}

////////////////////////////////////////////////
// Progress bar
double Perc = 100 * (g+1) / (NGIBBS);
if ( (g+1) % (NGIBBS/100) == 0 && verbose == 1) {
  Rprintf("*");
  R_FlushConsole();
  if( (g+1) % (NGIBBS/10) == 0 ) {
    double mAr_beta=0; // Mean acceptance rate of beta
    double mAr_lambda=0; // Mean acceptance rate of lambda
    for ( int j = 0; j < NSP; j++ ) {
      for ( int p = 0; p < NP; p++ ) {
        mAr_beta += Ar_beta(j,p) / (NSP*NP);
      } // loop on rank of parameters
      for ( int q = 0; q < NL; q++ ) {
        mAr_lambda += Ar_lambda(j,q) / (NSP*NL-NL*(NL-1)*0.5);
      } // loop on rank of latent variable
    } // loop on species
```

```
        double mAr_W=0; // Mean acceptance rate of W
        double mAr_alpha=0; // Mean acceptance rate of alpha
        for ( int i = 0; i < NSITE; i++ ) {
          mAr_alpha += Ar_alpha(i) / NSITE;
          for ( int q = 0; q < NL; q++ ) {
            mAr_W += Ar_W(i,q) / (NSITE*NL);
          }// loop on rank of latent variable
        }// loop on sites
        Rprintf(":%.1f%%, mean accept. rates= beta:%.3f lambda:%.3f W:%.3f alpha:%4.3f\n",
                Perc, mAr_beta, mAr_lambda, mAr_W, mAr_alpha);
        R_FlushConsole();
      }
    }

    ////////////////////////////////////////////////
    // User interrupt
    R_CheckUserInterrupt(); // allow user interrupt

  } // Gibbs sampler

  // Free memory
  gsl_rng_free(r);

  // Return results as a Rcpp::List
  Rcpp::List w = Rcpp::List::create(Rcpp::Named("beta") = beta,
                                    Rcpp::Named("lambda") = lambda,
                                    Rcpp::Named("W") = W,
                                    Rcpp::Named("alpha") = alpha,
                                    Rcpp::Named("Valpha") = Valpha,
                                    Rcpp::Named("Deviance") = Deviance,
                                    Rcpp::Named("theta_latent") = theta_latent);

  return w;

}// end Rcpp_jSDM_gibbs_logit function
```