TD inférence bayésienne

Julien Papaïx (avec l'aide de Samuel Soubeyrand et d'Emily Walker)

26 mars 2018

Contents

0) Charger les packages et les données

```
library(R2OpenBUGS)
library(coda)
library(HDInterval)
don <- read.table("donnees.txt",header=T)
```

1) Ecrire un MCMC à la main

Nous allons considérer une fonction *dynamics()* qui simule une dynamique latente, observée uniquement à travers du processus d'observation *obsprocess()*. L'objectif est d'écire un algorithme de Metropolis-Hastings pour estimer les paramètres de la dynamique sous-jacente.

1.1) Probabilité d'acceptation

A partir des trois blocs qui suivent et de la formule du cours écrire la probabilité d'acceptation de l'algorithme MCMC - MH.

```
## loglikelihood function
if(length(obsprocess$parfunction)==1){
    loglikelihood=function(u){ sum(obsprocess$ddistrib(Y,obsprocess$parfunction(u),log=TRUE)) }
} else {
    if(length(obsprocess$parfunction)==2){
        loglikelihood=function(u){
            sum(obsprocess$ddistrib(Y,obsprocess$parfunction[[1]](u),
                obsprocess$parfunction[[2]](u),log=TRUE))
        }
    }
}

## logprior function
if(length(prior$parfunction)==1){
    logprior=function(u){ sum(prior$ddistrib(u,prior$parfunction(),log=TRUE)) }
} else {
    if(length(prior$par)==2){
        logprior=function(u){
            sum(prior$ddistrib(u,prior$parfunction[[1]](),prior$parfunction[[2]](),log=TRUE))
        }
    }
}

## logproposal function and generator under the proposal distribution
if(length(proposal$parfunction)==1){
    logproposal=function(par1,par2){
        sum(proposal$ddistrib(par2,proposal$parfunction(par1),log=TRUE))
    }
    rproposal=function(par){
        proposal$rdistrib(length(par),proposal$parfunction(par))
    }
} else {
    if(length(proposal$parfunction)==2){
        logproposal=function(par1,par2){
            sum(proposal$ddistrib(par2,proposal$parfunction[[1]](par1),
                proposal$parfunction[[2]](par1), log=TRUE))
        }
        rproposal=function(par){
            proposal$rdistrib(length(par),proposal$parfunction[[1]](par),
                proposal$parfunction[[2]](par))
        }
    }
}
```

1.2) MCMC

Ecrire la boucle décrivant les itérations MCMC.

A partir des étapes précédentes, écrire la fonction prenant comme arguments le nombre d'itérations, les données, le modèle de dynamique, le processus d'observation, les *a priori* et les lois de proposition et écrivant un fichier avec les valeurs des paramètres retenus à chaque itération.

1.3) On teste !

Simuler des données issues d'observations poissoniennes d'une dynamique exponentielle de type $exp(\alpha + \beta * t)$.

Utiliser la fonction pour estimer les paramètres des données simulées. On pourra prendre des *a priori* et des lois de proposition normales.

Jouer avec les paramètres du MCMC (par exemple la variance des lois de proposition), la loi des observations...

2) Estimation des paramètres d'un modèle proie-prédateur avec OpenBUGS

On va travailler ici sur des données de récolte de fourrures de Lynx et Lièvre au Canada par la *Hudson's Bay Company* dans les années 1900-1920.

2.1) Les données

Tracer la dynamique du Lynx et du lièvre sur le même graphique. Commenter.

2.2) Le modèle dynamique

Bien que plusieurs facteurs peuvent expliquer la dynamique observée, nous allons simplifier les choses en considérant le modèle de Lokta-Voltera suivant:

$$\begin{cases} \frac{du}{dt} = & \alpha u - \beta uv \\ \frac{dv}{dt} = & -\gamma v + \delta uv \end{cases}$$

La fonction *ode()* de OpenBUGS permet de simuler des systèmes EDO. Elle prend comme arguments les valeurs initiales, les temps pour lesquels la simulation est souhaitée, le système d'équations, le temps aux valeurs initiales et une tolérance.

Ecrire le modèle dynamique et la fonction le simulant.

2.3) Les observations

Nous allons dans un premier temps considérer le modèle log-normal suivant:

$$u_t^{obs} = u_t exp(\epsilon_t^u) \qquad v_t^{obs} = v_t exp(\epsilon_t^v),$$

avec

$$\epsilon_t^u \sim N(0, \sigma^u) \qquad \epsilon_t^v \sim N(0, \sigma^v).$$

Utiliser la distribution *dlnorm()* pour écrire le modèle d'observation. Attention à ne pas oublier les valeurs initiales...

2.4) Les *a priori*

Définir les lois *a priori* en prenant bien en compte les contraintes sur les paramètres.

2.5) Ajustement

Répéter les données, et introduire une mesure d'ajustement.

2.6) On lance la machine !

Mettre en forme les données puis lancer l'estimation à l'aide de la fonction *bugs()*.

2.7) Diagnostics

Utiliser la fonction *gelman.diag()* pour vérifier la convergence des chaînes.

Récupérer les mesures d'ajustement et conclure.

Regarder les corrélations entre paramètres et les auto-corrélations au sein des chaînes.

2.8) Estimations

Tracer les densités *a posteriori* des paramètres.

Donner les estimations ponctuelles des paramètres ainsi que l'intervalle de crédibilité et l'intervalle de plus grande densité à 95%.

Tracer les prédictions du modèle dynamique et celles des données répétées.

2.9) Et si on change le modèle d'observation...

Changer le modèle log-normal par un modèle de poisson. Refaire l'estimation, comparer avec les résultats précédents et discuter.

2.10) Echantillonner autrement

OpenBUGS est basé sur un échantillonneur de Gibbs et met à jour les paramètres les uns après les autres. Il peut être intéressant de faire une mise à jour par blocs. Pour ceci, il est nécessaire de passer par une loi multivariée. Changer les *a priori* dans le modèle log-normal pour mettre à jour les paramètres du modèle de Lokta-Voltera en un bloc.