# pva-2023

2023-01-15

## Analyse de viabilité

### Préparatifs

On calcule la viabilité de la population selon un modèle avec ou sans freinage. Ces modèles ont été développés par Guillaume pour l'expertise.

Tout se passe en bayésien. Si vous vous embêtez, vous pouvez m'écouter pendant 7 heures introduire tout ça par ici. Pour ce qui nous intéresse ici, il nous faudra un package spécifique pour implémenter les méthodes MCMC.

```
library(R2jags)
```

```
## Loading required package: rjags
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v ggplot2 3.4.0      v purrr   1.0.0
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

### Les modèles

On définit deux modèles, un modèle exponentiel, et un autre avec freinage.

Avec le modèle exponentiel, on stipule que les effectifs $N_t$ à l'année $t$ sont obtenus à partir des effectifs à l'année $t-1$ auxquels on a retranché les prélèvements $H_{t-1}$, le tout multiplié par le taux de croissance annuel $\lambda$ :

$$N_t = \lambda(N_{t-1} - H_{t-1}).$$

Cette relation est déterministe. Pour ajouter de la variabilité démographique, on suppose que les effectifs sont distribués selon une distribution log-normale, autrement dit que les effectifs sont normalement distribués sur l'échelle log :

$$\log(N_t) \sim \text{Normale}(\mu_t, \sigma_{\text{proc}})$$

avec $\mu_t = \log(N_t) = \log(\lambda(N_{t-1} - H_{t-1}))$ et $\sigma_{\text{proc}}$ l'erreur standard des effectifs sur l'échelle log. On aurait pu prendre une loi de Poisson à la place. La stochasticité environnementale est en général captée par le taux de croissance, mais pas ici puisqu'il est constant. C'est une hypothèse forte du modèle. Dans l'idéal, on pourrait coupler le modèle de capture-recapture, et le modèle qui décrit l'évolution des effectifs au cours du temps.

On ajoute une couche d'observation qui capture les erreurs sur les effectifs. **Ici il me semble qu'on peut faire mieux puisque les erreurs d'observation sont connues, et estimées par capture-recapture. Dommage de ne pas les utiliser.**

Si l'on note $y_t$ les effectifs observés, on suppose que ces comptages annuels (qui sont des estimations, je le redis, pas des observations entachées d'erreur comme dans le cas classique) sont distribués comme une loi de Poisson de moyenne les vrais effectifs $N_t$:

$$y_t \sim \text{Poisson}(N_t).$$

Souvent, les données de comptage sont surdispersés (la variance des $y_t$ est plus grande que son espérance,or l'égalité est le défaut dans une distribution de Poisson). Pour gérer ce problème, Guillaume spécifie le paramètre de la Poisson, le taux, comme une loi gamma. **Cela revient en fait à considérer que les effectifs sont distribués comme une distribution binomiale négative** qui relâche l'hypothèse d'égalité entre espérance et variance de la Poisson. Il écrit que :

$$y_t \sim \text{Poisson}(\psi_t)$$

avec $\psi_t \sim \gamma(\alpha_t, \beta_t)$, $\alpha_t = N_t^2/\sigma^2_{\text{obs}}$ et $\beta_t = N_t/\sigma^2_{\text{obs}}$.

Ce modèle exponentiel s'écrit de la façon suivante dans Jags.

```
exp_model <- function(){

  # Priors
  errorObs ~ dunif(0, 0.20)
  sigmaProc ~ dunif(0, 10)
  tauProc <- 1/sigmaProc^2
  lambda ~ dunif(0, 2)
  N[1] ~ dgamma(1.0E-6, 1.0E-6)

  # Process model
  for (t in 2:(nyears)) {
    # density dependence on lambda
    Nproc[t] <- log(max(1, lambda*N[t-1]))
    N[t] ~ dlnorm(Nproc[t], tauProc)
}
```

```
  # Observation model
  for (t in 1:nyears) {
    sigmaObs[t] <- errorObs*N[t]
    shapeObs[t] <- N[t]*N[t]/(sigmaObs[t]*sigmaObs[t])
    rateObs[t] <- N[t]/(sigmaObs[t]*sigmaObs[t])
    lambdaNobs[t] ~ dgamma(shapeObs[t], rateObs[t])
    Nobs[t] ~ dpois(lambdaNobs[t])
}

  # Projected population
  for (t in (nyears + 1):(nyears + 15)) {
    Nproc[t] <- log(max(1, lambda*(N[t-1])))
    N[t] ~ dlnorm(Nproc[t], tauProc)
}

}
```

Il y a un terme dit de densité-dépendance qui est introduit dans ce code, via `Nproc[t] <- log(max(1, lambda*N[t-1]))`. L'astuce `log(max(1, x))` permet de s'assurer que le log est toujours positif puisqu'on prend le log d'une quantité qui est toujours au moins égale à 1.

A noter que les projections sont faites sur 15 années à venir, en utilisant une distribution log-normale.

Des priors informatifs (il me semble) sont spécifiés. **Ca vaudrait le coup de faire des priors predictive check pour vérifier que les priors induits sur les vrais effectifs $N_t$ ont du sens.**

Vient ensuite le modèle avec freinage. Celui-ci est une simple variation du modèle exponentiel dans lequel on introduit un terme de freinage qui agit sur le taux de croissance $\lambda$ à l'année $t$ via $\lambda + \beta * X_t$ où $X_t = 0$ les années sans freinage, et $X_t = 1$ pour les années avec freinage. Cette variable $X_t$ est à spécifier par l'utilisateur. **Il serait important d'expliciter ce choix dans les analyses, et d'étudier peut-être plusieurs choix.**

```
frein_model <- function(){

  # Priors
  errorObs ~ dunif(0, 0.20)
  sigmaProc ~ dunif(0, 10)
  tauProc <- 1/sigmaProc^2
  lambda ~ dunif(0, 2)
  beta ~ dnorm(0, 1.0E-6)
  N[1] ~ dgamma(1.0E-6, 1.0E-6)

  # Process model
  for (t in 2:(nyears)) {
    Nproc[t] <- log(max(1, (lambda + beta*X[t-1])*N[t-1]))
    N[t] ~ dlnorm(Nproc[t], tauProc)
}

  # Observation model
  for (t in 1:nyears) {
    sigmaObs[t] <- errorObs*N[t]
    shapeObs[t] <- N[t]*N[t]/(sigmaObs[t]*sigmaObs[t])
    rateObs[t] <- N[t]/(sigmaObs[t]*sigmaObs[t])
    lambdaNobs[t] ~ dgamma(shapeObs[t], rateObs[t])
    Nobs[t] ~ dpois(lambdaNobs[t])
```

```
}

  # Projected population
  for (t in (nyears + 1):(nyears + 15)) {
    Nproc[t] <- log(max(1, (lambda + beta)*(N[t-1])))
    N[t] ~ dlnorm(Nproc[t], tauProc)
}
}
```

Rien de nouveau dans les projections, sauf que $X_t = 1$ pour $t = T + 1, \ldots, T + 1$.

Ici aussi on a un terme de densité-dépendance. Des priors informatifs sont spécifiés, et non-informatifs (sur $\beta$ par exemple) qui pourraient être faiblement informatifs. A nouveau, **ça vaudrait le coup de faire des priors predictive check**.

### Les données

Il nous faut en théorie aussi les nombres de loups tués par an. En théorie, car **en inspectant les modèles ci-dessus, et la liste des données ci-dessous, il s'avère que les modèles développés par Guillaume n'utilisent pas les nombres de loups tués!**. C'est un peu embêtant si l'on veut explorer la viabilité en fonction des stratégies de gestion (seuil de prélèvements). Guillaume devait avoir une raison pour ne pas les inclure. C'est aussi une raison pour laquelle je proposais de s'inspirer de la théorie écologique des prélèvements (voir à la fin du document pour plus).

```
harvest <- c(0,0,0,0,0,1,0,0,2,1,2,0,0,1,0,4,4,6,18,36,34,42,51,98,105,103,169)
```

Les estimations d'effectifs par CMR:

```
CMR <- c(17.1,35.4,
47.7,
25.1,
62.6,
47.9,
81.7,
110.5,
102.7,
135.9,
132.6,
101.7,
130.3,
141.4,
141.5,
175.5,
210.3,
174.5,
353.6,
280.2,
376.7,
561.2,
571.9,
682.4,
645.7,
```

```
783.8,
868)
```

### Ajustement du modèle exponentiel, sans freinage

On met ensemble les effectifs estimés par CMR ainsi que les nombres de loups tués.

```
thedata <- cbind(round(CMR), harvest)
colnames(thedata) <- c("N", "H")
thedata <- as.data.frame(thedata)
nyears <- nrow(thedata)
```

On construit une liste avec les données, et on précise les paramètres à estimer et le nombre de chaines de MCMC (j'en prends trois ici).

```
bugs.data <- list(
    nyears = nyears,
    Nobs = thedata$N
)
bugs.monitor <- c("lambda", "sigmaProc", "sigmaObs", "N", "tauProc")
bugs.chains <- 3
bugs.inits <- function(){
    list(
    )
}
```

Allez zooh, on lance la machine!

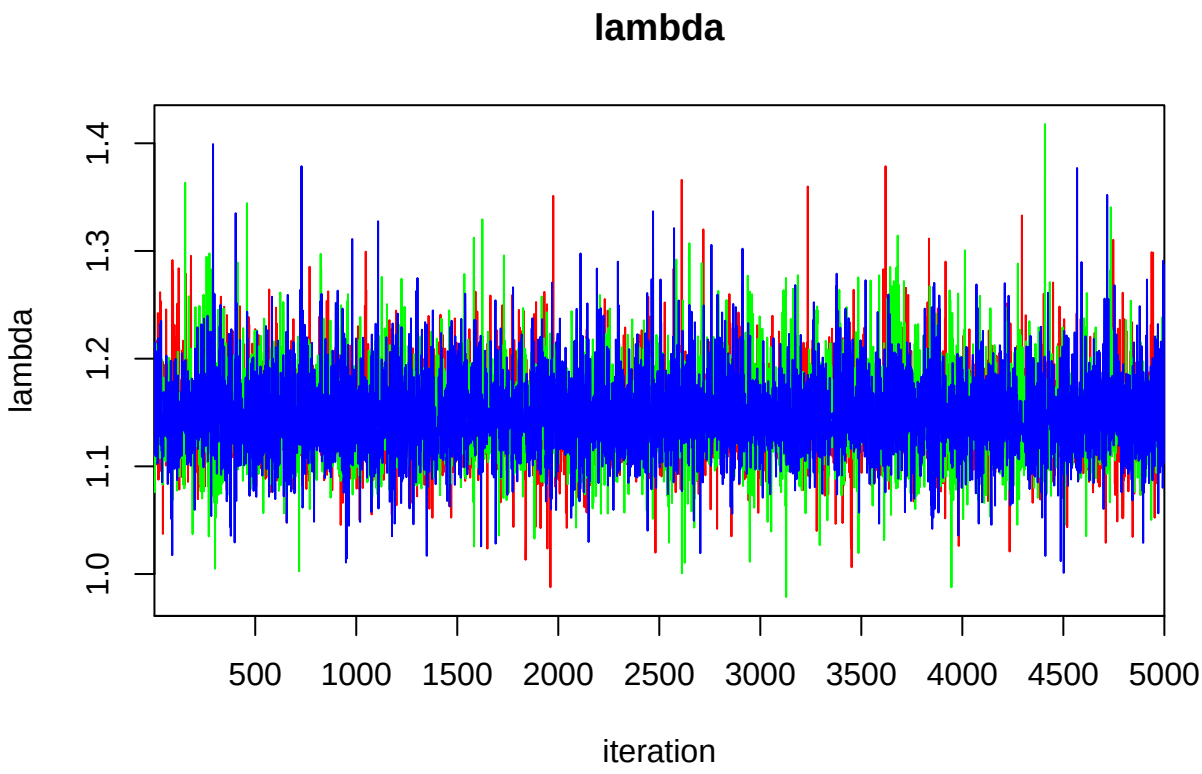
```
exp_res <- jags(data = bugs.data,
                inits = bugs.inits,
                parameters.to.save = bugs.monitor,
                model.file = exp_model,
                n.chains = bugs.chains,
                                                n.thin = 10,
                                                n.iter = 100000,
                                                n.burnin = 50000)
```

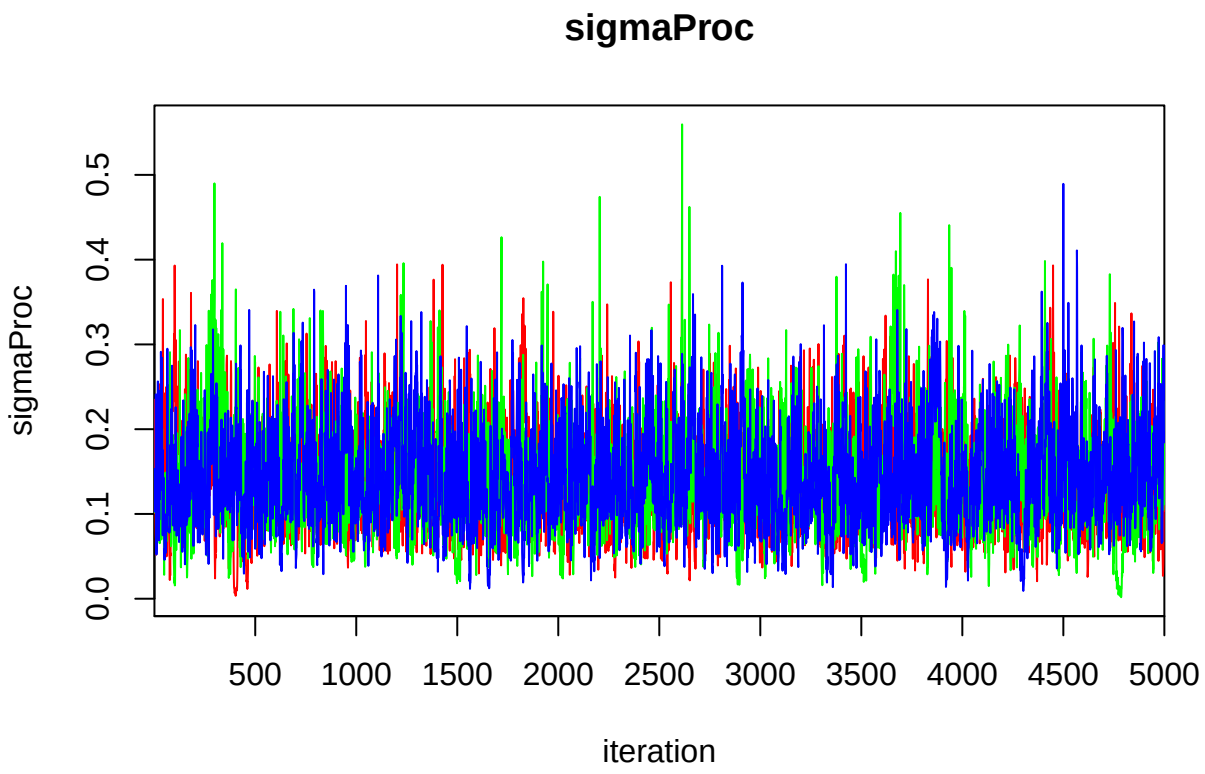
```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 27
##    Unobserved stochastic nodes: 72
##    Total graph size: 366
##
## Initializing model
```

On inspecte la convergence.

```
traceplot(exp_res, varname = c('lambda', "sigmaProc"), ask = FALSE)
```






Jetons un coup d'oeil aux estimations.

```
res <- print(exp_res, intervals = c(2.5/100, 50/100, 97.5/100))
```

```
## Inference for Bugs model at "/var/folders/ln/jf2twlj12snbq000z6qq5y7m0000gn/T//Rtmp4PbdS7/model1f647
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
##  n.sims = 15000 iterations saved
##                mu.vect   sd.vect     2.5%      50%     97.5%  Rhat n.eff
## N[1]            25.059     4.483   16.960   24.835    34.348 1.002  3100
## N[2]            31.835     4.258   24.106   31.627    40.808 1.001  5100
## N[3]            38.537     4.703   30.047   38.241    48.623 1.002  3000
## N[4]            40.773     5.438   30.320   40.819    51.314 1.002  2300
## N[5]            52.755     5.930   42.223   52.368    65.434 1.002  2400
## N[6]            59.693     6.822   46.678   59.483    73.892 1.001  6300
## N[7]            76.757     8.468   61.958   76.210    95.166 1.001  5100
## N[8]            94.646    11.576   74.820   93.783   119.536 1.001  4700
## N[9]           104.186    11.460   84.081  103.350   128.698 1.001 15000
## N[10]          119.419    13.891   95.247  118.490   148.803 1.001  6100
## N[11]          123.594    12.710  101.326  122.636   150.937 1.001 15000
## N[12]          120.297    13.051   95.341  120.060   146.881 1.001  3800
## N[13]          132.346    13.269  107.530  131.918   160.251 1.001  7800
## N[14]          144.149    14.610  117.910  143.671   174.353 1.001  6400
## N[15]          156.212    16.810  125.538  155.257   190.422 1.002  3300
## N[16]          179.809    18.187  146.510  178.909   217.721 1.001 15000
## N[17]          206.024    20.461  168.195  205.294   248.732 1.002  2200
## N[18]          223.418    27.815  172.510  223.047   279.072 1.002  3200
```

```
## N[19]          300.069    29.635  246.007  298.311   362.766 1.001  3900
## N[20]          321.646    32.658  262.896  320.055   389.693 1.001 15000
## N[21]          394.220    36.528  328.734  391.859   473.290 1.001 13000
## N[22]          503.926    50.231  414.564  500.666   608.031 1.001  4400
## N[23]          567.043    51.842  473.396  564.589   676.625 1.002  3000
## N[24]          647.102    59.875  537.394  644.411   775.010 1.001  7300
## N[25]          688.909    64.955  574.764  683.834   830.825 1.001 15000
## N[26]          788.480    74.053  651.790  784.385   954.202 1.001 15000
## N[27]          893.287    96.936  719.715  885.426  1107.135 1.001 15000
## N[28]         1040.123   206.158  693.506 1021.684  1503.358 1.001 15000
## N[29]         1210.186   319.544  702.727 1168.325  1962.171 1.001 15000
## N[30]         1410.633   469.681  734.846 1341.695  2518.096 1.001 15000
## N[31]         1644.543   667.287  755.299 1535.977  3195.241 1.001 15000
## N[32]         1919.668   929.277  791.985 1759.443  4035.644 1.001 15000
## N[33]         2240.784  1238.600  832.014 2006.297  4996.106 1.001 15000
## N[34]         2619.936  1609.223  887.176 2311.571  6163.322 1.001 15000
## N[35]         3078.517  2078.071  938.499 2657.989  7918.388 1.001 15000
## N[36]         3620.403  2855.066 1017.201 3040.254  9859.719 1.001 15000
## N[37]         4272.094  3910.501 1073.468 3483.730 12422.422 1.001 15000
## N[38]         5058.241  5569.892 1147.966 4004.172 15223.235 1.001 15000
## N[39]         5994.217  7552.849 1230.159 4573.175 19120.678 1.001 15000
## N[40]         7110.191  9643.113 1323.999 5240.991 23686.319 1.001 15000
## N[41]         8479.124 13214.744 1432.443 6030.287 30301.086 1.001 15000
## N[42]        10158.540 18130.047 1524.272 6944.207 37779.211 1.001 15000
## lambda           1.149     0.037    1.082    1.146     1.228 1.001 15000
## sigmaObs[1]      3.380     1.387    0.443    3.407     5.978 1.010  1100
## sigmaObs[2]      4.226     1.562    0.666    4.304     7.039 1.013  1000
## sigmaObs[3]      5.080     1.793    0.867    5.210     8.246 1.014  1100
## sigmaObs[4]      5.481     2.095    0.764    5.625     9.128 1.014  1100
## sigmaObs[5]      6.953     2.428    1.167    7.159    11.103 1.012  1400
## sigmaObs[6]      7.963     2.895    1.175    8.249    12.863 1.013  1100
## sigmaObs[7]     10.095     3.437    1.696   10.434    15.913 1.014  1100
## sigmaObs[8]     12.331     4.031    2.220   12.753    19.311 1.015  1100
## sigmaObs[9]     13.679     4.592    2.268   14.198    21.487 1.013  1100
## sigmaObs[10]    15.568     5.090    2.781   16.113    24.227 1.015  1100
## sigmaObs[11]    16.229     5.476    2.762   16.812    25.458 1.013  1200
## sigmaObs[12]    16.097     5.936    2.281   16.614    26.105 1.012  1100
## sigmaObs[13]    17.604     6.367    2.760   18.088    28.427 1.012  1300
## sigmaObs[14]    19.199     6.999    3.015   19.752    31.241 1.014  1100
## sigmaObs[15]    20.903     7.807    3.223   21.428    34.615 1.013  1200
## sigmaObs[16]    23.956     8.753    3.774   24.521    39.265 1.013  1200
## sigmaObs[17]    27.383     9.933    4.414   28.045    44.667 1.012  1100
## sigmaObs[18]    30.181    11.739    4.093   30.917    50.722 1.011  1100
## sigmaObs[19]    39.276    13.140    7.265   40.528    61.586 1.014  1200
## sigmaObs[20]    43.053    15.841    6.218   44.459    70.135 1.012  1200
## sigmaObs[21]    52.367    18.524    8.115   54.188    83.723 1.013  1100
## sigmaObs[22]    65.845    21.548   11.875   68.258   101.832 1.014  1100
## sigmaObs[23]    74.728    25.272   12.385   77.563   116.748 1.014  1100
## sigmaObs[24]    85.072    28.631   14.509   88.067   132.935 1.014  1000
## sigmaObs[25]    91.673    32.695   14.151   95.076   146.364 1.013  1100
## sigmaObs[26]   104.498    36.873   16.868  107.919   167.417 1.012  1200
## sigmaObs[27]   118.656    42.950   18.778  122.030   194.730 1.012  1200
## sigmaProc        0.140     0.060    0.043    0.132     0.280 1.004  1700
## tauProc        188.752  3192.444   12.800   57.122   528.615 1.004  1700
```

```
## deviance       216.924     7.883  203.327  216.268   234.020 1.001 15000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 31.1 and DIC = 248.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
res
```

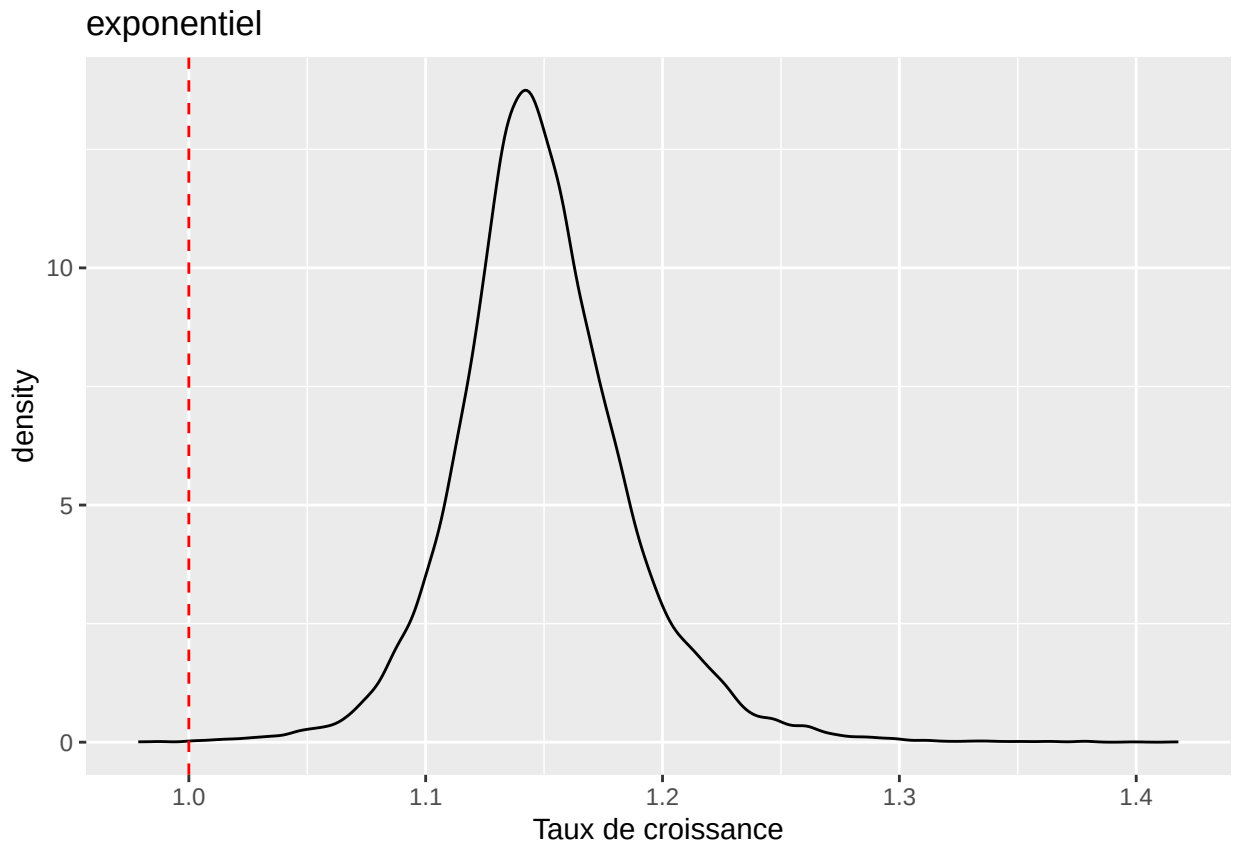
```
## Inference for Bugs model at "/var/folders/ln/jf2twlj12snbq000z6qq5y7m0000gn/T//Rtmp4PbdS7/model1f647
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
##  n.sims = 15000 iterations saved
##                mean      sd   2.5%    25%    50%     75%   97.5% Rhat n.eff
## N[1]           25.1     4.5   17.0   21.9   24.8    28.0    34.3    1  3100
## N[2]           31.8     4.3   24.1   28.9   31.6    34.5    40.8    1  5100
## N[3]           38.5     4.7   30.0   35.3   38.2    41.4    48.6    1  3000
## N[4]           40.8     5.4   30.3   37.1   40.8    44.3    51.3    1  2300
## N[5]           52.8     5.9   42.2   48.7   52.4    56.4    65.4    1  2400
## N[6]           59.7     6.8   46.7   55.1   59.5    64.0    73.9    1  6300
## N[7]           76.8     8.5   62.0   70.9   76.2    81.9    95.2    1  5100
## N[8]           94.6    11.6   74.8   86.3   93.8   102.1   119.5    1  4700
## N[9]          104.2    11.5   84.1   96.2  103.3   111.3   128.7    1 15000
## N[10]         119.4    13.9   95.2  109.4  118.5   128.3   148.8    1  6100
## N[11]         123.6    12.7  101.3  114.8  122.6   131.5   150.9    1 15000
## N[12]         120.3    13.1   95.3  111.5  120.1   128.6   146.9    1  3800
## N[13]         132.3    13.3  107.5  123.4  131.9   140.5   160.3    1  7800
## N[14]         144.1    14.6  117.9  133.9  143.7   153.6   174.4    1  6400
## N[15]         156.2    16.8  125.5  144.5  155.3   167.2   190.4    1  3300
## N[16]         179.8    18.2  146.5  167.3  178.9   191.3   217.7    1 15000
## N[17]         206.0    20.5  168.2  191.8  205.3   219.3   248.7    1  2200
## N[18]         223.4    27.8  172.5  203.5  223.0   242.2   279.1    1  3200
## N[19]         300.1    29.6  246.0  279.7  298.3   319.1   362.8    1  3900
## N[20]         321.6    32.7  262.9  298.8  320.1   342.9   389.7    1 15000
## N[21]         394.2    36.5  328.7  369.8  391.9   416.2   473.3    1 13000
## N[22]         503.9    50.2  414.6  468.3  500.7   536.7   608.0    1  4400
## N[23]         567.0    51.8  473.4  531.3  564.6   598.8   676.6    1  3000
## N[24]         647.1    59.9  537.4  605.9  644.4   684.3   775.0    1  7300
## N[25]         688.9    65.0  574.8  644.6  683.8   728.5   830.8    1 15000
## N[26]         788.5    74.1  651.8  739.0  784.4   832.5   954.2    1 15000
## N[27]         893.3    96.9  719.7  828.9  885.4   950.2  1107.1    1 15000
## N[28]        1040.1   206.2  693.5  905.8 1021.7  1147.4  1503.4    1 15000
## N[29]        1210.2   319.5  702.7 1009.2 1168.3  1358.8  1962.2    1 15000
## N[30]        1410.6   469.7  734.8 1122.6 1341.7  1600.7  2518.1    1 15000
## N[31]        1644.5   667.3  755.3 1257.9 1536.0  1884.5  3195.2    1 15000
## N[32]        1919.7   929.3  792.0 1407.8 1759.4  2223.3  4035.6    1 15000
## N[33]        2240.8  1238.6  832.0 1578.8 2006.3  2599.5  4996.1    1 15000
## N[34]        2619.9  1609.2  887.2 1772.0 2311.6  3054.3  6163.3    1 15000
## N[35]        3078.5  2078.1  938.5 1983.6 2658.0  3563.2  7918.4    1 15000
## N[36]        3620.4  2855.1 1017.2 2226.4 3040.3  4200.0  9859.7    1 15000
## N[37]        4272.1  3910.5 1073.5 2493.8 3483.7  4916.0 12422.4    1 15000
## N[38]        5058.2  5569.9 1148.0 2815.9 4004.2  5729.4 15223.2    1 15000
## N[39]        5994.2  7552.8 1230.2 3155.2 4573.2  6733.1 19120.7    1 15000
```

```
## N[40]         7110.2  9643.1 1324.0 3551.7 5241.0  7961.0 23686.3    1 15000
## N[41]         8479.1 13214.7 1432.4 3999.5 6030.3  9291.8 30301.1    1 15000
## N[42]        10158.5 18130.0 1524.3 4514.6 6944.2 10869.0 37779.2    1 15000
## deviance       216.9     7.9  203.3  211.2  216.3   222.0   234.0    1 15000
## lambda           1.1     0.0    1.1    1.1    1.1     1.2     1.2    1 15000
## sigmaObs[1]      3.4     1.4    0.4    2.5    3.4     4.3     6.0    1  1100
## sigmaObs[2]      4.2     1.6    0.7    3.3    4.3     5.3     7.0    1  1000
## sigmaObs[3]      5.1     1.8    0.9    4.0    5.2     6.3     8.2    1  1100
## sigmaObs[4]      5.5     2.1    0.8    4.2    5.6     7.0     9.1    1  1100
## sigmaObs[5]      7.0     2.4    1.2    5.5    7.2     8.7    11.1    1  1400
## sigmaObs[6]      8.0     2.9    1.2    6.2    8.2    10.0    12.9    1  1100
## sigmaObs[7]     10.1     3.4    1.7    8.2   10.4    12.5    15.9    1  1100
## sigmaObs[8]     12.3     4.0    2.2   10.2   12.8    15.0    19.3    1  1100
## sigmaObs[9]     13.7     4.6    2.3   11.2   14.2    16.7    21.5    1  1100
## sigmaObs[10]    15.6     5.1    2.8   12.8   16.1    19.0    24.2    1  1100
## sigmaObs[11]    16.2     5.5    2.8   13.1   16.8    20.1    25.5    1  1200
## sigmaObs[12]    16.1     5.9    2.3   12.4   16.6    20.4    26.1    1  1100
## sigmaObs[13]    17.6     6.4    2.8   13.7   18.1    22.2    28.4    1  1300
## sigmaObs[14]    19.2     7.0    3.0   14.9   19.8    24.1    31.2    1  1100
## sigmaObs[15]    20.9     7.8    3.2   16.0   21.4    26.4    34.6    1  1200
## sigmaObs[16]    24.0     8.8    3.8   18.6   24.5    30.1    39.3    1  1200
## sigmaObs[17]    27.4     9.9    4.4   21.3   28.0    34.4    44.7    1  1100
## sigmaObs[18]    30.2    11.7    4.1   22.7   30.9    38.6    50.7    1  1100
## sigmaObs[19]    39.3    13.1    7.3   31.7   40.5    48.4    61.6    1  1200
## sigmaObs[20]    43.1    15.8    6.2   33.5   44.5    54.3    70.1    1  1200
## sigmaObs[21]    52.4    18.5    8.1   41.4   54.2    65.4    83.7    1  1100
## sigmaObs[22]    65.8    21.5   11.9   54.0   68.3    80.6   101.8    1  1100
## sigmaObs[23]    74.7    25.3   12.4   60.7   77.6    92.2   116.7    1  1100
## sigmaObs[24]    85.1    28.6   14.5   68.8   88.1   105.1   132.9    1  1000
## sigmaObs[25]    91.7    32.7   14.2   72.2   95.1   115.0   146.4    1  1100
## sigmaObs[26]   104.5    36.9   16.9   82.8  107.9   130.3   167.4    1  1200
## sigmaObs[27]   118.7    43.0   18.8   93.0  122.0   148.0   194.7    1  1200
## sigmaProc        0.1     0.1    0.0    0.1    0.1     0.2     0.3    1  1700
## tauProc        188.8  3192.4   12.8   32.5   57.1   104.2   528.6    1  1700
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 31.1 and DIC = 248.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Le taux de croissance vaut 1.149145 avec son intervalle de crédibilité à 95% qui vaut (1.0821245, 1.2284741). Comme cet intervalle ne contient clairement pas 1, le taux de croissance est au-dessus de 1 sans ambiguïté. On le voit aussi sur la distribution a posteriori estimée dont la masse est concentrée sur les valeurs plus grandes que 1.

```
post_exp <- exp_res$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
#  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
#  filter(str_detect(parameter, "lambda")) %>%
  ggplot() +
  aes(x = lambda) +
  geom_density() +
```

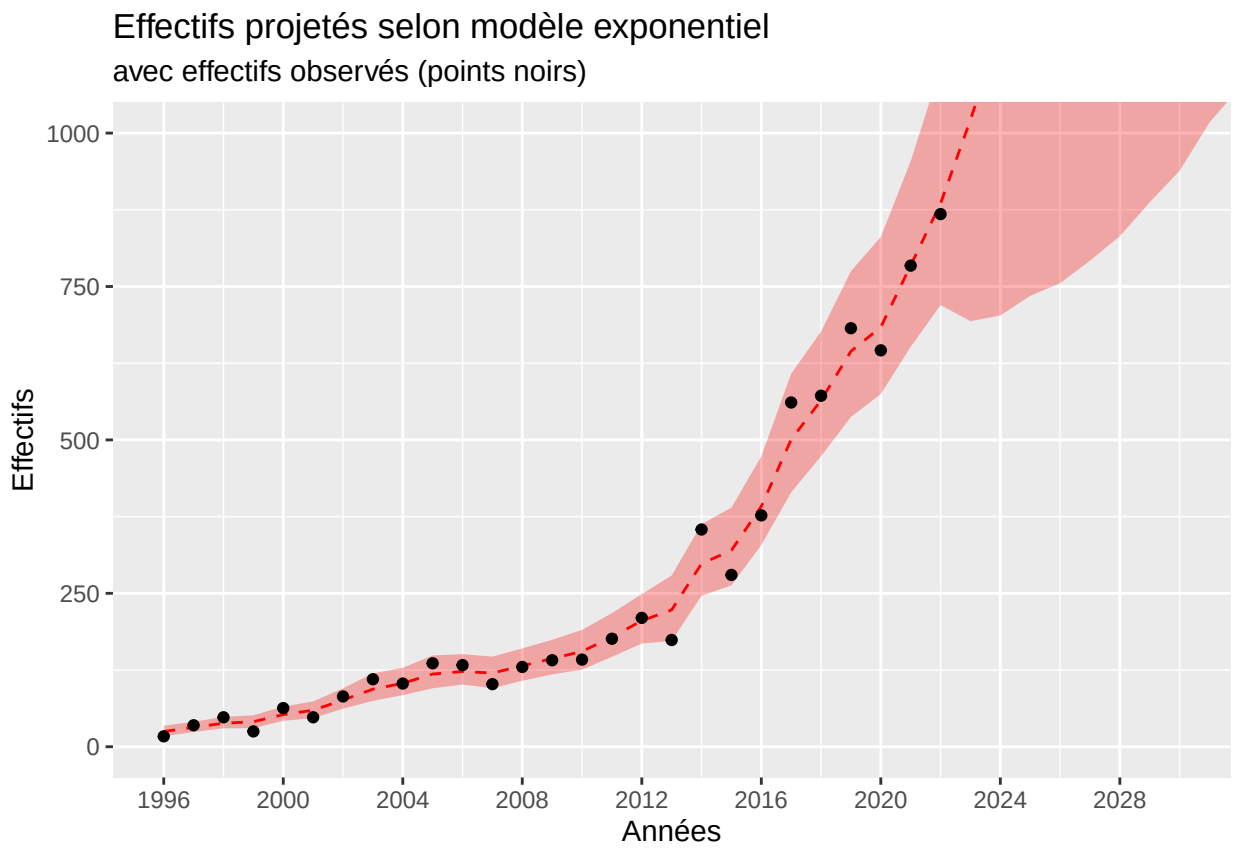
```
  geom_vline(xintercept = 1, lty = "dashed", color = "red") +
  labs(x = "Taux de croissance", title = "exponentiel")
post_exp
```




Ensuite les projections.

```
proj_exp <- exp_res$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
  filter(str_detect(parameter, "N")) %>%
  group_by(parameter) %>%
  summarize(medianN = median(value),
            lci = quantile(value, probs = 2.5/100),
            uci = quantile(value, probs = 97.5/100)) %>%
  mutate(an = parse_number(parameter)) %>%
  arrange(an) %>%
  ggplot() +
  geom_ribbon(aes(x = an, y = medianN, ymin = lci, ymax = uci), fill = "red", alpha = 0.3) +
  geom_line(aes(x = an, y = medianN), lty = "dashed", color = "red") +
#  geom_point(aes(x = an, y = medianN), color = "red") +
  geom_point(data = bugs.data %>% as_tibble, aes(x = 1:unique(nyears), y = Nobs)) +
  coord_cartesian(xlim = c(1, 35), ylim = c(0, 1000)) +
  labs(y = "Effectifs",
       x = "Années",
       title = "Effectifs projetés selon modèle exponentiel",
       subtitle = "avec effectifs observés (points noirs)") +
```

```
  scale_x_continuous(breaks = seq(1, 40, by = 4),
                     labels = seq(1996, 2035, by = 4))
proj_exp
```




**Ajustement du modèle avec freinage**

Les données, les paramètres à estimer, le nombre de chaînes MCMC et les valeurs initiales (qu'on laisse à Jags la liberté de choisir). Attention à $X$ dans les données. Je prendrai le temps aussi d'écrire le modèle formellement, en langage mathématique, pour être bien sûr de comprendre ce qu'il fait.

```
bugs.data <- list(
    nyears = nyears,
    Nobs = thedata$N,
    X = c(rep(0, nyears-8), rep(1, 7), NA)
)
bugs.monitor <- c("lambda", "beta", "sigmaProc", "sigmaObs", "N", "tauProc")
bugs.chains <- 3
bugs.inits <- function(){
    list(
    )
}
```

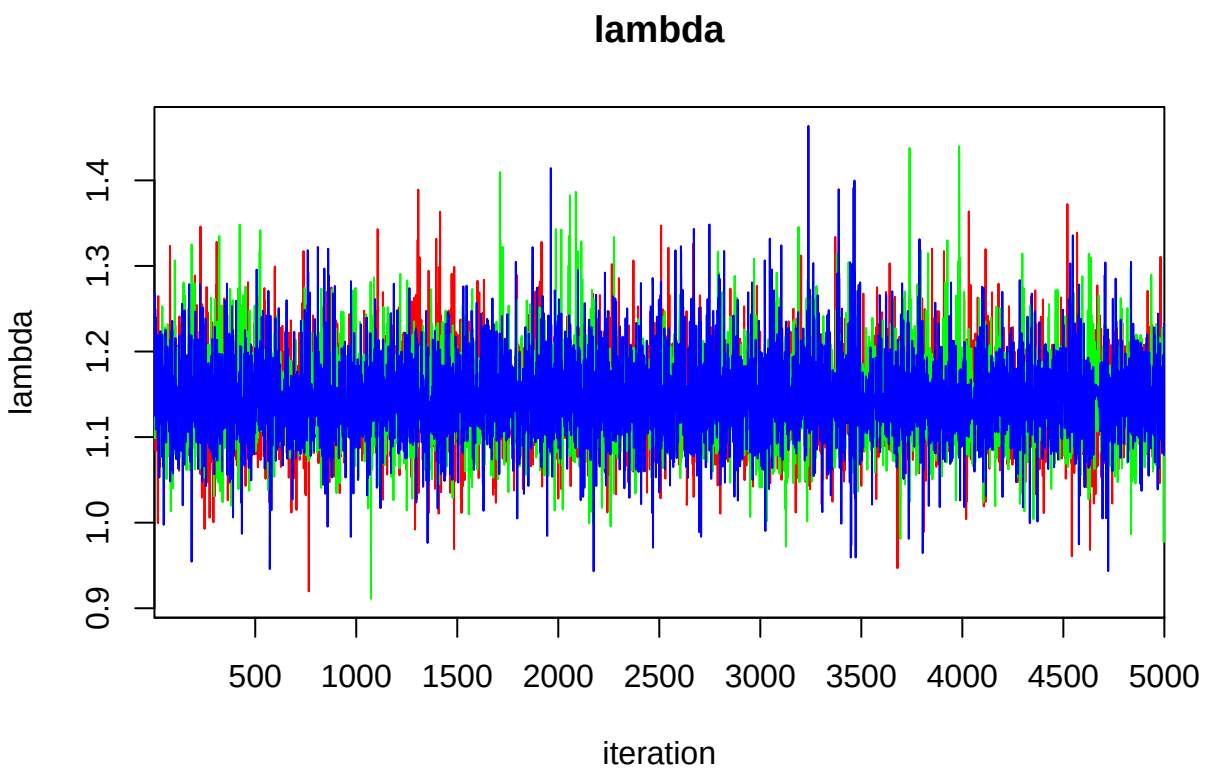
Et zooh, on lance le tout!

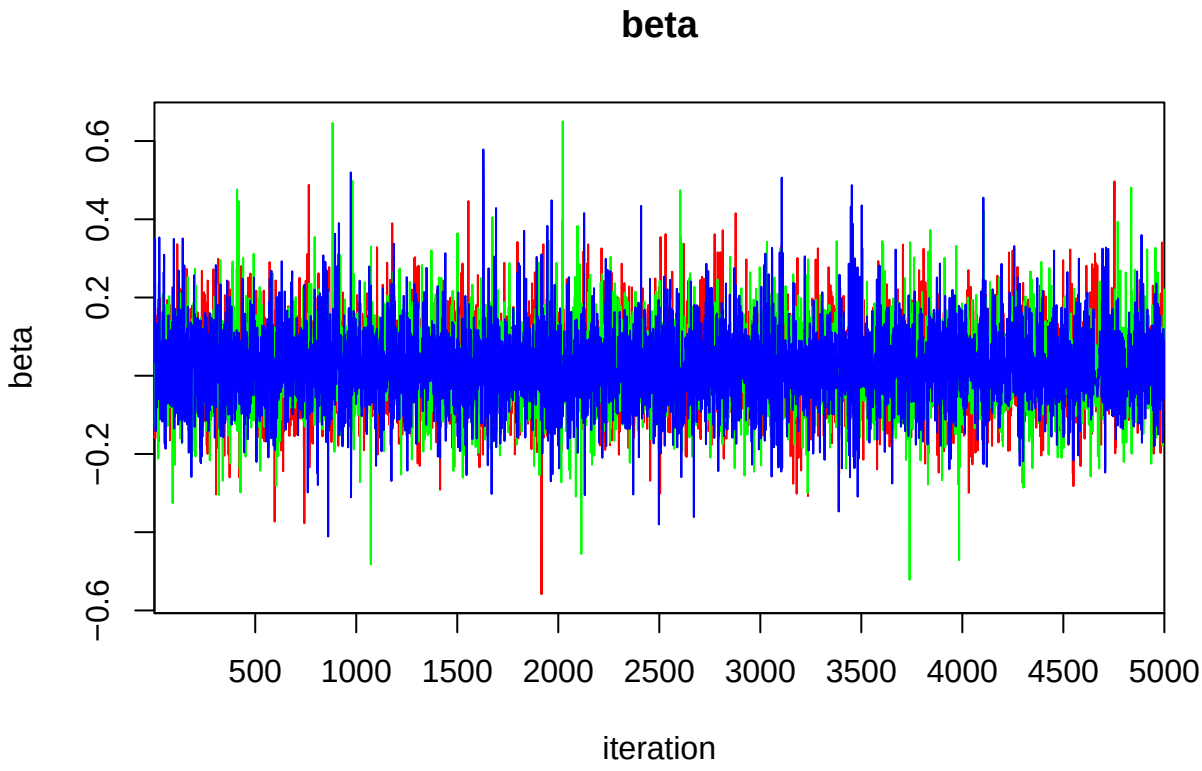
```
frein_mcmc <- jags(data = bugs.data,
                                                    inits = bugs.inits,
                                                    parameters.to.save = bugs.monitor,
                                                    model.file = frein_model,
                                                    n.chains = bugs.chains,
                                                    n.thin = 10,
                                                    n.iter = 100000,
                                                    n.burnin = 50000)
```

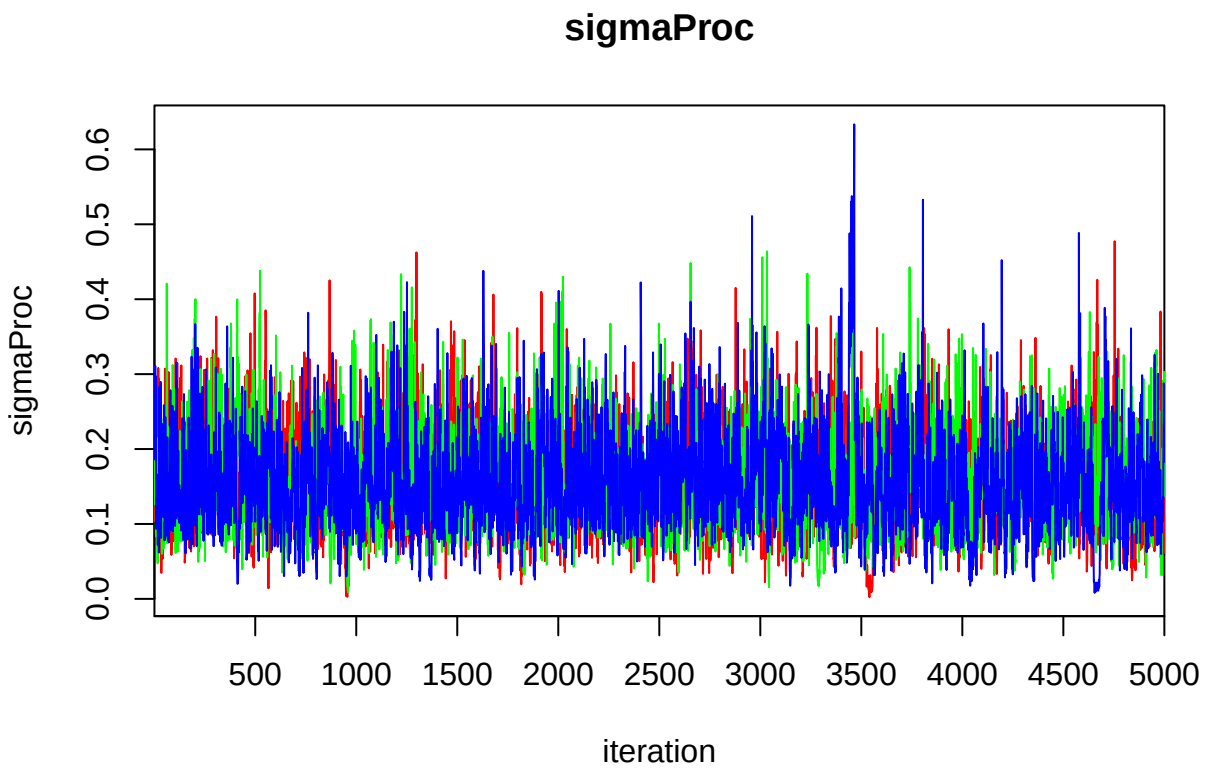
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 27
##    Unobserved stochastic nodes: 73
##    Total graph size: 398
##
## Initializing model
```

On inspecte la convergence.

```
traceplot(frein_mcmc, varname = c('lambda', "beta", "sigmaProc"), ask = FALSE)
```




beta
beta
−0.6
−0.2
0.2
0.4
0.6
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
iteration




Jetons un coup d'oeil aux estimations.

```
res <- print(frein_mcmc, intervals = c(2.5/100, 50/100, 97.5/100))
```

```
## Inference for Bugs model at "/var/folders/ln/jf2twlj12snbq000z6qq5y7m0000gn/T//Rtmp4PbdS7/model1f646
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
##  n.sims = 15000 iterations saved
##                mu.vect    sd.vect    2.5%      50%      97.5%  Rhat n.eff
## N[1]            24.747      4.708  16.192   24.460     34.569 1.001  5500
## N[2]            32.003      4.401  23.954   31.757     41.331 1.001  4600
## N[3]            38.974      4.947  30.197   38.651     49.775 1.001 15000
## N[4]            40.306      5.687  29.292   40.271     51.537 1.001  7700
## N[5]            53.019      6.163  42.048   52.657     66.203 1.001 15000
## N[6]            59.340      7.083  45.966   59.111     74.215 1.001 15000
## N[7]            77.378      8.723  62.073   76.755     96.082 1.001  4400
## N[8]            96.068     11.875  75.171   95.291    121.591 1.002  2500
## N[9]           105.025     11.832  84.207  104.219    130.593 1.001 15000
## N[10]          121.043     14.270  95.864  120.163    150.883 1.002  2600
## N[11]          124.314     13.094 101.315  123.567    152.779 1.001  4000
## N[12]          119.057     13.506  94.071  118.605    146.864 1.001 15000
## N[13]          131.514     13.666 106.430  130.862    160.188 1.001  6100
## N[14]          143.190     14.781 116.056  142.509    174.320 1.001  8800
## N[15]          154.539     16.836 124.272  153.632    190.115 1.001  4700
## N[16]          178.712     18.247 145.315  177.912    217.209 1.001  5800
## N[17]          204.919     20.500 166.879  203.869    247.941 1.001 15000
## N[18]          219.412     27.540 170.466  218.354    275.455 1.001  6100
```

```
## N[19]          301.575     31.679 244.987  299.642    367.520 1.001 15000
## N[20]          315.790     34.783 255.457  312.827    392.018 1.001 15000
## N[21]          391.197     37.682 323.676  388.364    472.501 1.001  4700
## N[22]          508.988     52.648 414.030  506.948    615.741 1.001 13000
## N[23]          569.835     53.964 471.761  566.966    686.329 1.001  7400
## N[24]          651.244     61.417 537.364  649.083    780.566 1.001 15000
## N[25]          689.332     67.090 571.348  683.441    834.176 1.001 15000
## N[26]          792.389     77.757 650.855  787.179    960.844 1.001 15000
## N[27]          901.448    105.138 715.229  890.056   1140.548 1.001 15000
## N[28]         1073.824    251.403 669.276 1047.325   1649.724 1.001 15000
## N[29]         1285.353    430.143 648.895 1224.220   2319.959 1.001 15000
## N[30]         1546.085    670.940 649.689 1428.744   3225.569 1.001 15000
## N[31]         1871.567   1031.657 643.858 1663.342   4429.449 1.001 15000
## N[32]         2274.961   1560.806 649.538 1932.886   6054.767 1.001 15000
## N[33]         2784.957   2306.823 659.608 2259.657   8202.160 1.001 15000
## N[34]         3456.997   4073.074 680.841 2639.288  11432.463 1.001 15000
## N[35]         4354.113  10799.316 701.892 3058.482  15491.625 1.001 15000
## N[36]         5485.161  16606.864 694.228 3547.983  21036.104 1.001 15000
## N[37]         7124.319  42236.657 706.986 4143.426  28290.007 1.001 15000
## N[38]         9137.536  57405.909 724.802 4829.618  38211.273 1.001 15000
## N[39]        11663.767  70382.288 737.726 5608.485  52037.607 1.001 15000
## N[40]        15779.630 127931.099 734.926 6572.110  69837.784 1.001 15000
## N[41]        22171.271 331852.772 747.285 7586.265  95132.295 1.001 15000
## N[42]        28796.647 418040.984 767.499 8815.168 125458.659 1.001 15000
## beta             0.023      0.095  -0.161    0.021      0.224 1.001 15000
## lambda           1.147      0.047   1.058    1.143      1.248 1.001 15000
## sigmaObs[1]      3.236      1.418   0.436    3.241      6.005 1.009  1100
## sigmaObs[2]      4.103      1.601   0.671    4.181      7.081 1.011   970
## sigmaObs[3]      4.963      1.852   0.860    5.112      8.234 1.012  1000
## sigmaObs[4]      5.254      2.153   0.735    5.407      9.077 1.009  1200
## sigmaObs[5]      6.767      2.516   1.161    6.989     11.148 1.011  1100
## sigmaObs[6]      7.678      3.004   1.159    7.967     12.839 1.008  1300
## sigmaObs[7]      9.836      3.553   1.724   10.248     15.907 1.010  1400
## sigmaObs[8]     12.104      4.221   2.242   12.575     19.397 1.013  1300
## sigmaObs[9]     13.355      4.821   2.318   13.868     21.525 1.012  1200
## sigmaObs[10]    15.266      5.314   2.862   15.846     24.306 1.013  1300
## sigmaObs[11]    15.805      5.687   2.734   16.406     25.515 1.011  1300
## sigmaObs[12]    15.442      6.100   2.314   15.960     25.799 1.009  1300
## sigmaObs[13]    16.934      6.509   2.738   17.405     28.188 1.010  1300
## sigmaObs[14]    18.448      7.119   3.018   18.950     30.888 1.010  1200
## sigmaObs[15]    20.023      7.942   3.041   20.517     34.015 1.009  1100
## sigmaObs[16]    23.011      8.881   3.750   23.610     38.671 1.010  1100
## sigmaObs[17]    26.331     10.059   4.385   27.107     44.032 1.010  1200
## sigmaObs[18]    28.728     11.861   4.005   29.552     49.550 1.009  1200
## sigmaObs[19]    38.078     13.438   7.201   39.187     61.184 1.012  1100
## sigmaObs[20]    40.952     16.190   6.095   42.174     69.279 1.010  1300
## sigmaObs[21]    50.309     19.056   8.050   52.172     83.020 1.010  1200
## sigmaObs[22]    64.305     22.473  11.684   66.613    102.995 1.012  1100
## sigmaObs[23]    72.691     26.360  12.262   75.744    117.475 1.011  1300
## sigmaObs[24]    82.877     29.886  14.499   86.090    133.482 1.011  1100
## sigmaObs[25]    88.998     34.291  13.891   91.995    148.714 1.010  1200
## sigmaObs[26]   101.838     38.841  16.667  104.626    170.042 1.010  1100
## sigmaObs[27]   116.221     45.792  18.687  118.355    201.083 1.010  1200
## sigmaProc        0.157      0.066   0.049    0.149      0.304 1.003  1500
```

```
## tauProc        157.689   2495.578  10.810   45.089    423.264 1.003  1500
## deviance       216.689      7.888 203.329  215.951    233.958 1.001 15000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 31.1 and DIC = 247.8
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
res
```

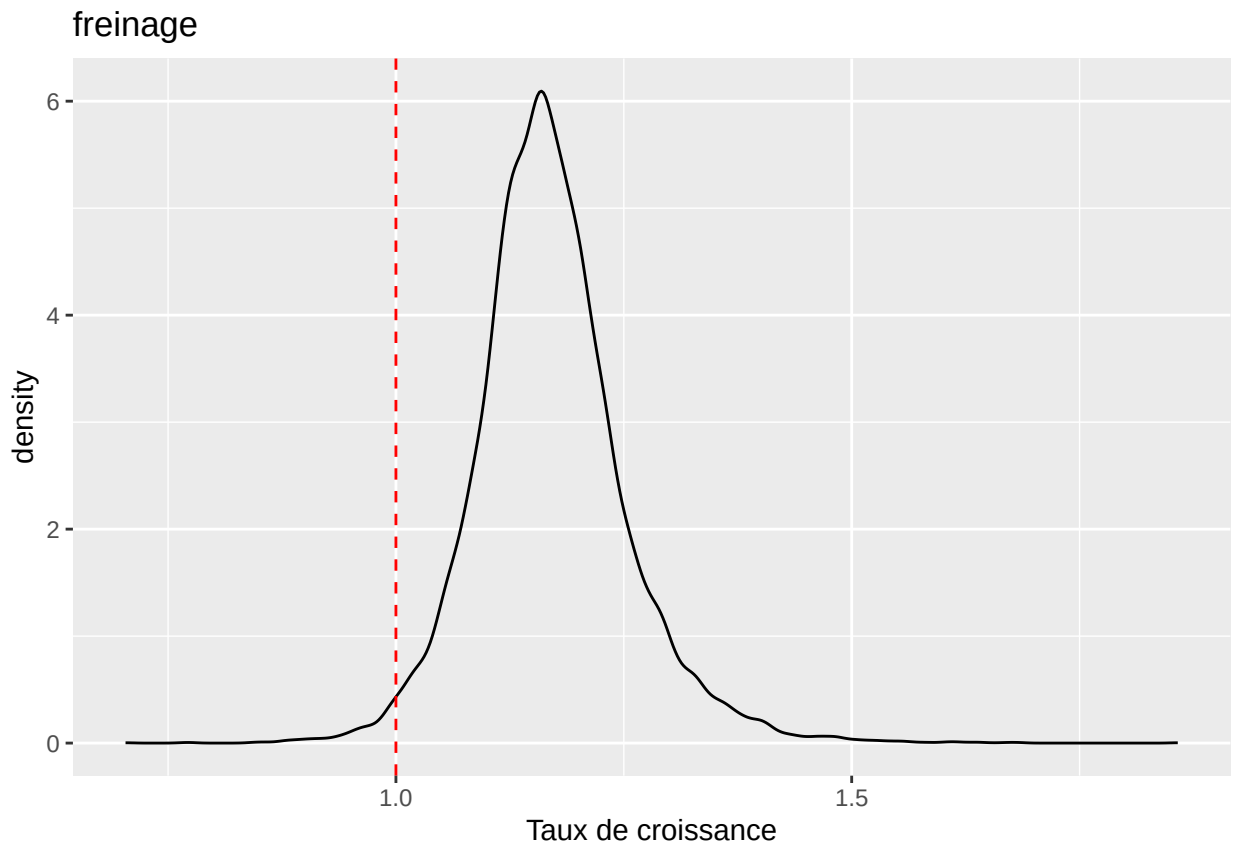
```
## Inference for Bugs model at "/var/folders/ln/jf2twlj12snbq000z6qq5y7m0000gn/T//Rtmp4PbdS7/model1f646
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
##  n.sims = 15000 iterations saved
##                mean       sd  2.5%    25%    50%     75%    97.5% Rhat n.eff
## N[1]           24.7      4.7  16.2   21.4   24.5    27.8     34.6    1  5500
## N[2]           32.0      4.4  24.0   29.0   31.8    34.8     41.3    1  4600
## N[3]           39.0      4.9  30.2   35.6   38.7    42.0     49.8    1 15000
## N[4]           40.3      5.7  29.3   36.5   40.3    44.0     51.5    1  7700
## N[5]           53.0      6.2  42.0   48.8   52.7    56.8     66.2    1 15000
## N[6]           59.3      7.1  46.0   54.5   59.1    63.8     74.2    1 15000
## N[7]           77.4      8.7  62.1   71.2   76.8    82.8     96.1    1  4400
## N[8]           96.1     11.9  75.2   87.6   95.3   103.6    121.6    1  2500
## N[9]          105.0     11.8  84.2   96.9  104.2   112.3    130.6    1 15000
## N[10]         121.0     14.3  95.9  110.9  120.2   130.2    150.9    1  2600
## N[11]         124.3     13.1 101.3  115.0  123.6   132.6    152.8    1  4000
## N[12]         119.1     13.5  94.1  109.9  118.6   127.6    146.9    1 15000
## N[13]         131.5     13.7 106.4  122.3  130.9   140.0    160.2    1  6100
## N[14]         143.2     14.8 116.1  133.3  142.5   152.6    174.3    1  8800
## N[15]         154.5     16.8 124.3  142.8  153.6   165.2    190.1    1  4700
## N[16]         178.7     18.2 145.3  166.3  177.9   190.1    217.2    1  5800
## N[17]         204.9     20.5 166.9  191.0  203.9   217.6    247.9    1 15000
## N[18]         219.4     27.5 170.5  199.4  218.4   237.9    275.5    1  6100
## N[19]         301.6     31.7 245.0  279.1  299.6   322.3    367.5    1 15000
## N[20]         315.8     34.8 255.5  291.1  312.8   337.6    392.0    1 15000
## N[21]         391.2     37.7 323.7  366.0  388.4   413.8    472.5    1  4700
## N[22]         509.0     52.6 414.0  471.6  506.9   544.2    615.7    1 13000
## N[23]         569.8     54.0 471.8  533.9  567.0   602.0    686.3    1  7400
## N[24]         651.2     61.4 537.4  610.4  649.1   688.8    780.6    1 15000
## N[25]         689.3     67.1 571.3  643.6  683.4   730.0    834.2    1 15000
## N[26]         792.4     77.8 650.9  741.5  787.2   837.8    960.8    1 15000
## N[27]         901.4    105.1 715.2  833.7  890.1   959.5   1140.5    1 15000
## N[28]        1073.8    251.4 669.3  906.4 1047.3  1203.8   1649.7    1 15000
## N[29]        1285.4    430.1 648.9 1009.5 1224.2  1476.7   2320.0    1 15000
## N[30]        1546.1    670.9 649.7 1128.6 1428.7  1804.6   3225.6    1 15000
## N[31]        1871.6   1031.7 643.9 1263.6 1663.3  2191.1   4429.4    1 15000
## N[32]        2275.0   1560.8 649.5 1415.6 1932.9  2671.8   6054.8    1 15000
## N[33]        2785.0   2306.8 659.6 1581.1 2259.7  3257.2   8202.2    1 15000
## N[34]        3457.0   4073.1 680.8 1764.5 2639.3  3916.2  11432.5    1 15000
## N[35]        4354.1  10799.3 701.9 1971.1 3058.5  4777.5  15491.6    1 15000
## N[36]        5485.2  16606.9 694.2 2178.8 3548.0  5806.4  21036.1    1 15000
## N[37]        7124.3  42236.7 707.0 2450.1 4143.4  7039.4  28290.0    1 15000
## N[38]        9137.5  57405.9 724.8 2740.5 4829.6  8460.6  38211.3    1 15000
```

```
## N[39]        11663.8  70382.3 737.7 3060.4 5608.5 10286.2  52037.6    1 15000
## N[40]        15779.6 127931.1 734.9 3433.9 6572.1 12516.3  69837.8    1 15000
## N[41]        22171.3 331852.8 747.3 3858.7 7586.3 15135.3  95132.3    1 15000
## N[42]        28796.6 418041.0 767.5 4349.9 8815.2 18206.3 125458.6    1 15000
## beta             0.0      0.1  -0.2    0.0    0.0     0.1      0.2    1 15000
## deviance       216.7      7.9 203.3  211.1  216.0   221.5    234.0    1 15000
## lambda           1.1      0.0   1.1    1.1    1.1     1.2      1.2    1 15000
## sigmaObs[1]      3.2      1.4   0.4    2.3    3.2     4.2      6.0    1  1100
## sigmaObs[2]      4.1      1.6   0.7    3.1    4.2     5.2      7.1    1   970
## sigmaObs[3]      5.0      1.9   0.9    3.8    5.1     6.2      8.2    1  1000
## sigmaObs[4]      5.3      2.2   0.7    3.8    5.4     6.8      9.1    1  1200
## sigmaObs[5]      6.8      2.5   1.2    5.2    7.0     8.6     11.1    1  1100
## sigmaObs[6]      7.7      3.0   1.2    5.7    8.0     9.9     12.8    1  1300
## sigmaObs[7]      9.8      3.6   1.7    7.7   10.2    12.3     15.9    1  1400
## sigmaObs[8]     12.1      4.2   2.2    9.7   12.6    15.0     19.4    1  1300
## sigmaObs[9]     13.4      4.8   2.3   10.5   13.9    16.7     21.5    1  1200
## sigmaObs[10]    15.3      5.3   2.9   12.2   15.8    18.9     24.3    1  1300
## sigmaObs[11]    15.8      5.7   2.7   12.4   16.4    19.9     25.5    1  1300
## sigmaObs[12]    15.4      6.1   2.3   11.5   16.0    19.9     25.8    1  1300
## sigmaObs[13]    16.9      6.5   2.7   12.7   17.4    21.7     28.2    1  1300
## sigmaObs[14]    18.4      7.1   3.0   13.9   19.0    23.6     30.9    1  1200
## sigmaObs[15]    20.0      7.9   3.0   14.8   20.5    25.8     34.0    1  1100
## sigmaObs[16]    23.0      8.9   3.8   17.2   23.6    29.4     38.7    1  1100
## sigmaObs[17]    26.3     10.1   4.4   19.8   27.1    33.5     44.0    1  1200
## sigmaObs[18]    28.7     11.9   4.0   20.7   29.6    37.5     49.6    1  1200
## sigmaObs[19]    38.1     13.4   7.2   29.9   39.2    47.5     61.2    1  1100
## sigmaObs[20]    41.0     16.2   6.1   30.5   42.2    52.8     69.3    1  1300
## sigmaObs[21]    50.3     19.1   8.0   38.2   52.2    64.0     83.0    1  1200
## sigmaObs[22]    64.3     22.5  11.7   51.2   66.6    79.9    103.0    1  1100
## sigmaObs[23]    72.7     26.4  12.3   56.9   75.7    91.2    117.5    1  1300
## sigmaObs[24]    82.9     29.9  14.5   64.6   86.1   104.1    133.5    1  1100
## sigmaObs[25]    89.0     34.3  13.9   66.9   92.0   113.8    148.7    1  1200
## sigmaObs[26]   101.8     38.8  16.7   77.1  104.6   129.5    170.0    1  1100
## sigmaObs[27]   116.2     45.8  18.7   86.6  118.4   148.0    201.1    1  1200
## sigmaProc        0.2      0.1   0.0    0.1    0.1     0.2      0.3    1  1500
## tauProc        157.7   2495.6  10.8   26.0   45.1    83.1    423.3    1  1500
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 31.1 and DIC = 247.8
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Le taux de croissance vaut 1.1696071 avec son intervalle de crédibilité à 95% qui vaut (1.0193808, 1.3521404). Cet intervalle ne contient pas 1, le taux de croissance est au-dessus de 1 et la distribution est plus recentrée vers la valeur 1 que pour le modèle sans freinage. On le voit aussi sur la distribution a posteriori estimée.

```
post_frein <- frein_mcmc$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
#  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
#  filter(str_detect(parameter, "lambda")) %>%
  ggplot() +
  aes(x = lambda + beta) +
```

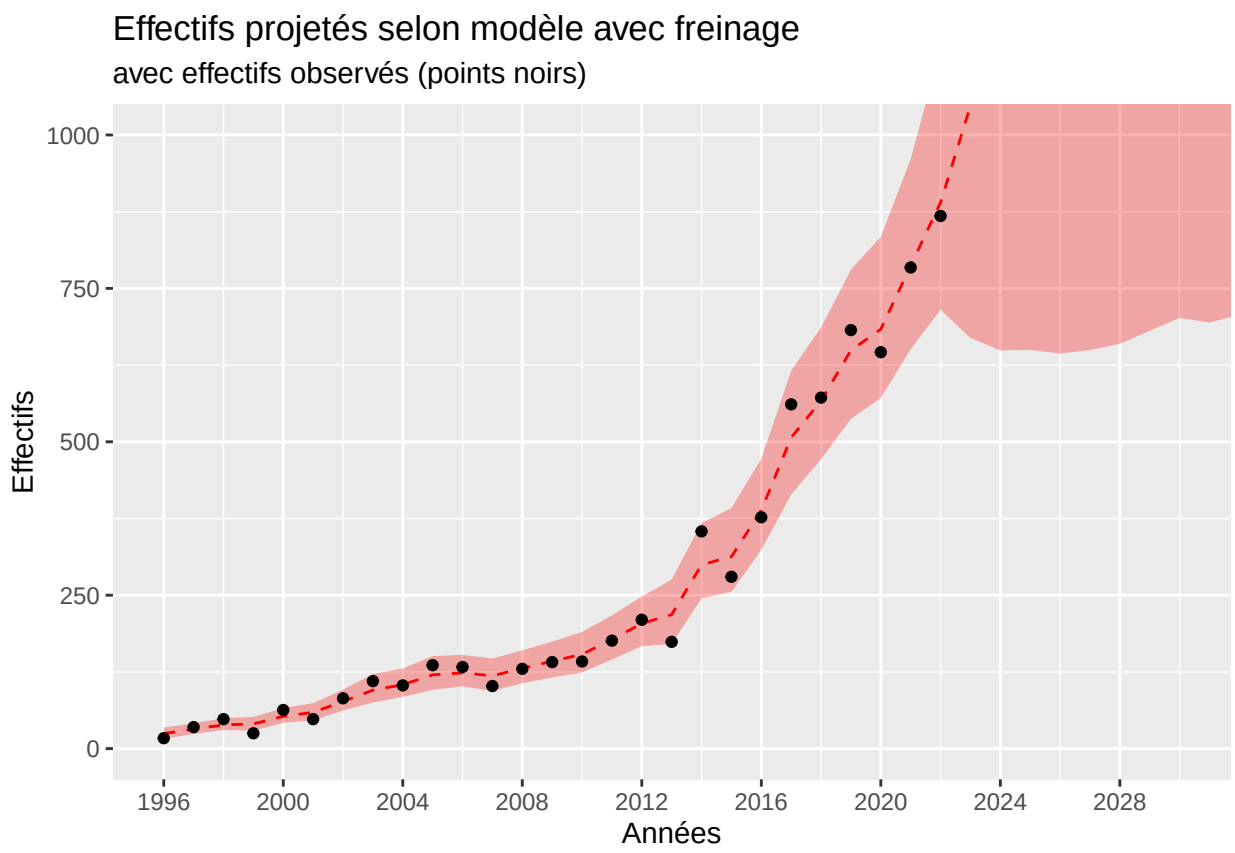
```
  geom_density() +
  geom_vline(xintercept = 1, lty = "dashed", color = "red") +
  labs(x = "Taux de croissance", title = "freinage")
post_frein
```




Ensuite les projections.

```
proj_frein <- frein_mcmc$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
  filter(str_detect(parameter, "N")) %>%
  group_by(parameter) %>%
  summarize(medianN = median(value),
            lci = quantile(value, probs = 2.5/100),
            uci = quantile(value, probs = 97.5/100)) %>%
  mutate(an = parse_number(parameter)) %>%
  arrange(an) %>%
  ggplot() +
  geom_ribbon(aes(x = an, y = medianN, ymin = lci, ymax = uci), fill = "red", alpha = 0.3) +
  geom_line(aes(x = an, y = medianN), lty = "dashed", color = "red") +
#  geom_point(aes(x = an, y = medianN), color = "red") +
  geom_point(data = bugs.data %>% as_tibble, aes(x = 1:unique(nyears), y = Nobs)) +
  coord_cartesian(xlim = c(1, 35), ylim = c(0, 1000)) +
  labs(y = "Effectifs",
       x = "Années",
       title = "Effectifs projetés selon modèle avec freinage",
```

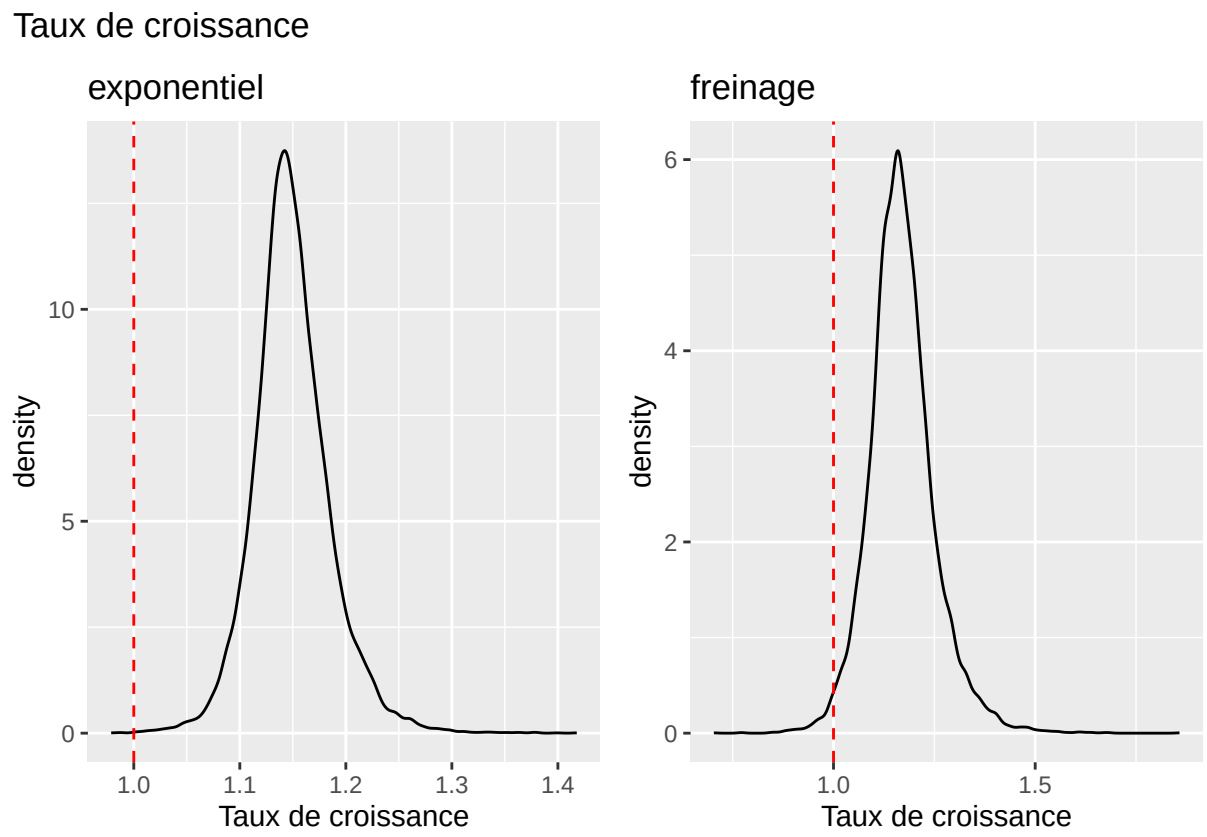
```
       subtitle = "avec effectifs observés (points noirs)") +
  scale_x_continuous(breaks = seq(1, 40, by = 4),
                     labels = seq(1996, 2035, by = 4))
proj_frein
```




## Conclusions

Pour résumer, on obtient pour le taux de croissance.

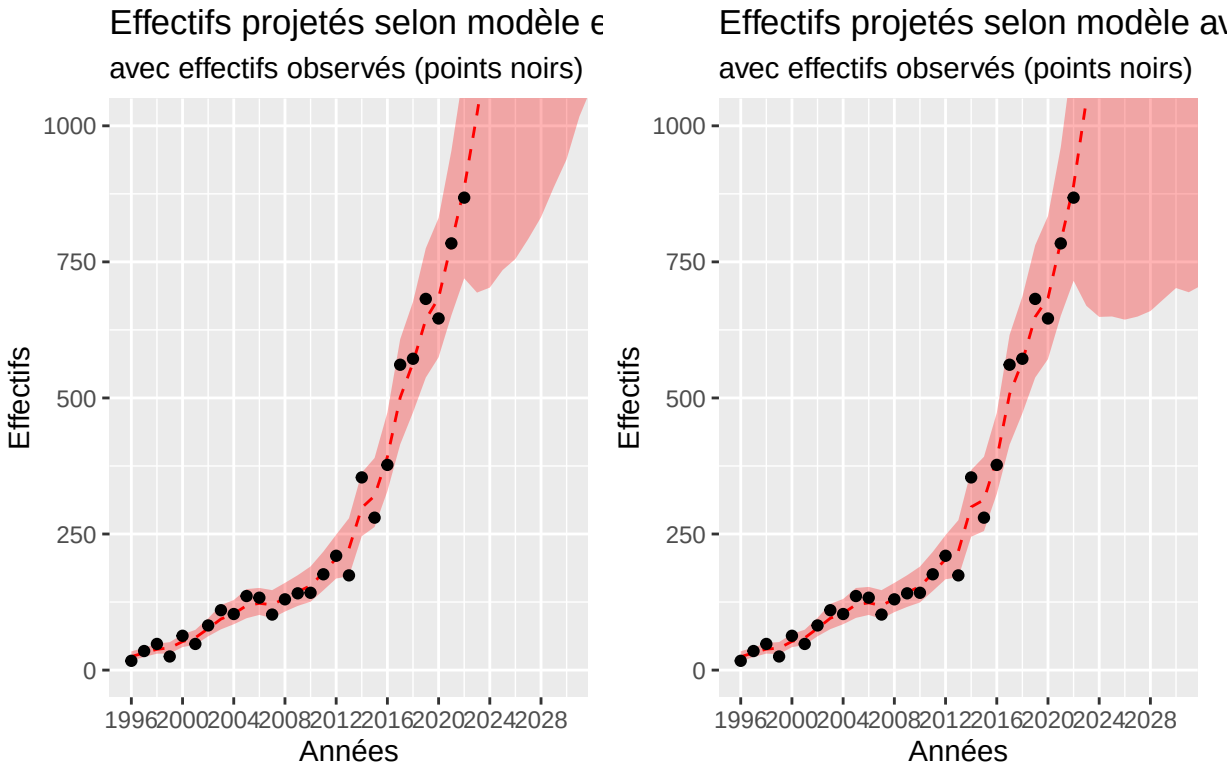
```
library(patchwork)
(post_exp | post_frein)  +
  plot_annotation(title = "Taux de croissance")
```




Et pour les projections.

```
(proj_exp | proj_frein) +
  plot_annotation("Effectifs projetés")
```

Effectifs projetés




- Avec le modèle exponentiel, le taux de croissance est de 15% avec un intervalle de crédibilité qui ne contient pas 0, donc significatif positif ; le modèle avec freinage suggère un taux de croissance similaire, avec un intervalle de crédibilité qui ne contient pas (mais pas de beaucoup) la valeur 0.
- Il est impossible de distinguer entre le modèle exponentiel vs celui avec freinage en utilisant un critère de sélection comme le DIC (un peu comme l'AIC en bayésien). Les valeurs sont très proches l'une de l'autre.

## Annexe: Modèle avec les prélèvements

On suit Andrén, H., Hobbs, N. T., Aronsson, M., Brøseth, H., Chapron, G., Linnell, J. D. C., Odden, J., Persson, J., and Nilsen, E. B.. 2020. Harvest models of small populations of a large carnivore using Bayesian forecasting. *Ecological Applications* 30(3):02063. 10.1002/eap.2063.

Dans leur papier, Henrik et les collègues construisent un modèle démographique structuré en classes d'âge. J'ai pas envie de me lancer dans un truc compliqué, l'idée est simplement de comprendre comment dérouler leur approche.

On part sur un modèle exponentiel. On stipule que les effectifs $N_t$ à l'année $t$ sont obtenus à partir des effectifs à l'année $t-1$ auxquels on a retranché les prélèvements $H_{t-1}$, le tout multiplié par le taux de croissance annuel $\lambda$ :

$$N_t = \lambda(N_{t-1} - H_{t-1}).$$

Cette relation est déterministe. Pour ajouter de la variabilité démographique, on suppose que les effectifs sont distribués selon une distribution log-normale, autrement dit que les effectifs sont normalement distribués sur l'échelle log :

$$\log(N_t) \sim \text{Normale}(\mu_t, \sigma_{\text{proc}})$$

avec $\mu_t = \log(N_t) = \log(\lambda(N_{t-1} - H_{t-1}))$ et $\sigma_{\text{proc}}$ l'erreur standard des effectifs sur l'échelle log. On aurait pu prendre une loi de Poisson à la place. La stochasticité environnementale est en général captée par le taux de croissance, mais pas ici puisqu'il est constant. C'est une hypothèse forte du modèle. Dans l'idéal, on pourrait coupler le modèle de capture-recapture, et le modèle qui décrit l'évolution des effectifs au cours du temps.

On ajoute une couche d'observation qui capture les erreurs sur les effectifs. Si l'on note $y_t$ les effectifs observés, on suppose que ces comptages annuels sont distribués comme une loi de Poisson de moyenne les vrais effectifs $N_t$:

$$y_t \sim \text{Poisson}(N_t).$$

```
model <- function(){

  # Priors
  sigmaProc ~ dunif(0, 10)
  tauProc <- 1/sigmaProc^2
  lambda ~ dunif(0, 5)

    N[1] ~ dgamma(1.0E-6, 1.0E-6)

    # Process model
    for (t in 2:(nyears)) {
      mu[t] <- lambda * (N[t-1] - harvest[t-1])
      Nproc[t] <- log(max(1, mu[t]))
      N[t] ~ dlnorm(Nproc[t], tauProc)
    }

  # Observation model
  for (t in 1:nyears) {
    y[t] ~ dpois(N[t])
  }

}
```

On prépare les données.

```
bugs.data <- list(
    nyears = nrow(thedata),
    y = round(thedata$N),
    harvest = thedata$H)
```

On précise les paramètres à estimer et le nombre de chaines de MCMC (j'en prends trois ici).

```
bugs.monitor <- c("lambda", "sigmaProc","N", "tauProc")
bugs.chains <- 3
bugs.inits <- function(){
    list(
    )
}
```

Allez zooh, on lance la machine!

```
library(R2jags)
wolf_mod <- jags(data = bugs.data,
                 inits = bugs.inits,
                 parameters.to.save = bugs.monitor,
                 model.file = model,
                 n.chains = bugs.chains,
                                                    n.thin = 10,
                                                    n.iter = 100000,
                                                    n.burnin = 50000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 27
##    Unobserved stochastic nodes: 29
##    Total graph size: 196
##
## Initializing model
```

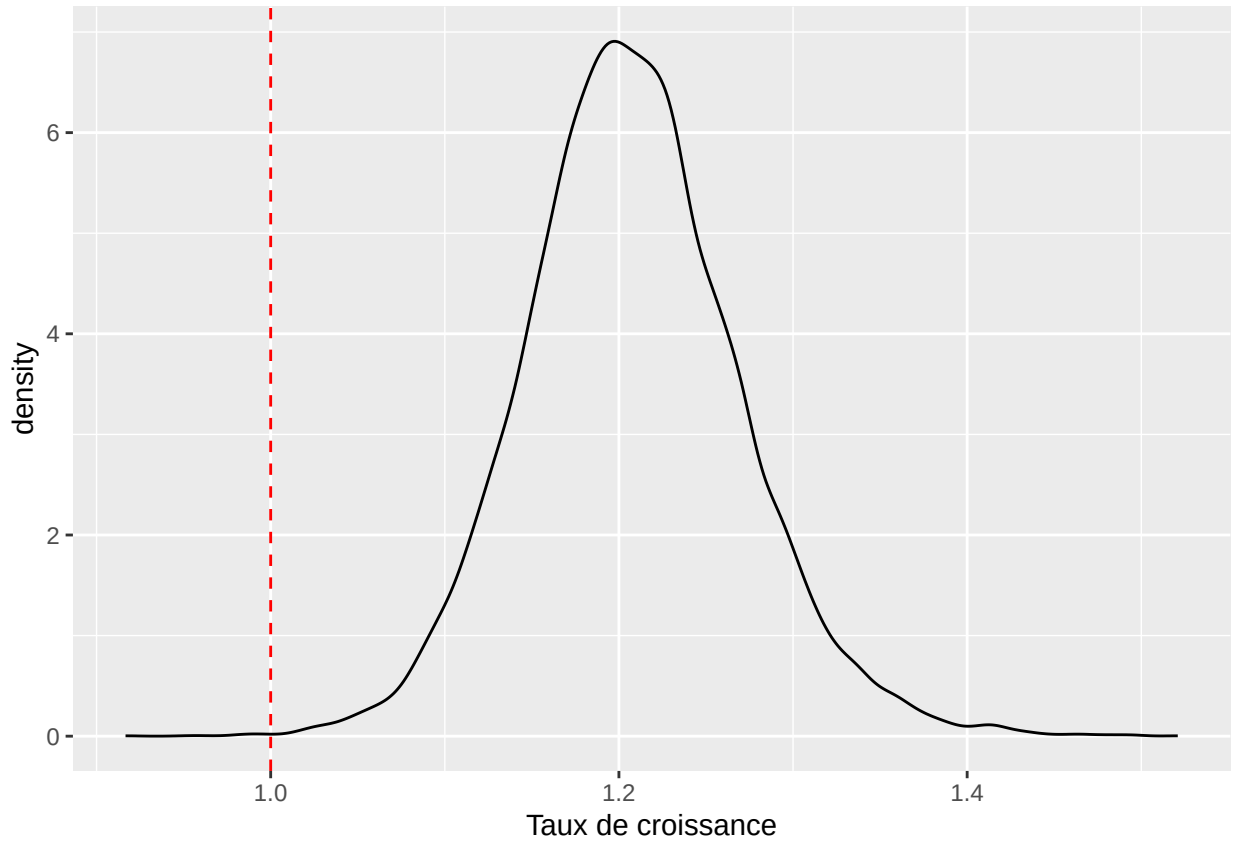
Jetons un coup d'oeil aux estimations.

```
print(wolf_mod, intervals = c(2.5/100, 50/100, 97.5/100))
```

```
## Inference for Bugs model at "/var/folders/ln/jf2twlj12snbq000z6qq5y7m0000gn/T//Rtmp4PbdS7/model1f646
##  3 chains, each with 1e+05 iterations (first 50000 discarded), n.thin = 10
##  n.sims = 15000 iterations saved
##           mu.vect sd.vect    2.5%     50%   97.5%  Rhat n.eff
## N[1]       21.067   3.772  14.244  20.832  29.067 1.001 15000
## N[2]       32.049   4.387  24.329  31.746  41.266 1.001 14000
## N[3]       41.271   5.247  31.900  40.910  52.481 1.001 15000
## N[4]       35.194   4.803  26.226  35.043  45.057 1.001 15000
## N[5]       55.345   6.283  43.878  54.953  68.841 1.001 15000
## N[6]       54.905   6.226  43.192  54.657  67.508 1.001 15000
## N[7]       80.001   7.739  65.747  79.601  96.373 1.001  9500
## N[8]      105.278   9.181  88.485 104.907 124.225 1.001 15000
## N[9]      106.668   9.141  89.403 106.396 125.283 1.001 15000
## N[10]     131.869  10.466 112.548 131.502 153.816 1.001 15000
## N[11]     130.055  10.294 110.689 129.771 150.890 1.001  9200
## N[12]     107.852   9.297  90.381 107.683 126.756 1.001 15000
## N[13]     128.536  10.168 109.426 128.260 149.174 1.001  7200
## N[14]     140.288  10.643 120.256 139.818 162.001 1.001 15000
## N[15]     144.792  10.882 124.479 144.406 166.911 1.001 15000
## N[16]     175.992  12.269 152.713 175.733 201.005 1.001 15000
## N[17]     205.502  13.400 179.999 205.218 232.447 1.001 15000
## N[18]     186.724  13.316 161.688 186.476 213.778 1.001 15000
## N[19]     340.466  18.132 306.025 340.144 376.880 1.001 15000
## N[20]     289.598  16.340 258.376 289.408 322.138 1.001 13000
## N[21]     378.814  18.564 343.603 378.745 415.871 1.001  5300
## N[22]     554.640  22.974 510.930 554.200 601.217 1.001  6500
## N[23]     574.670  23.313 529.900 574.434 621.320 1.001 15000
## N[24]     679.043  25.171 630.260 678.475 728.977 1.001 15000
```

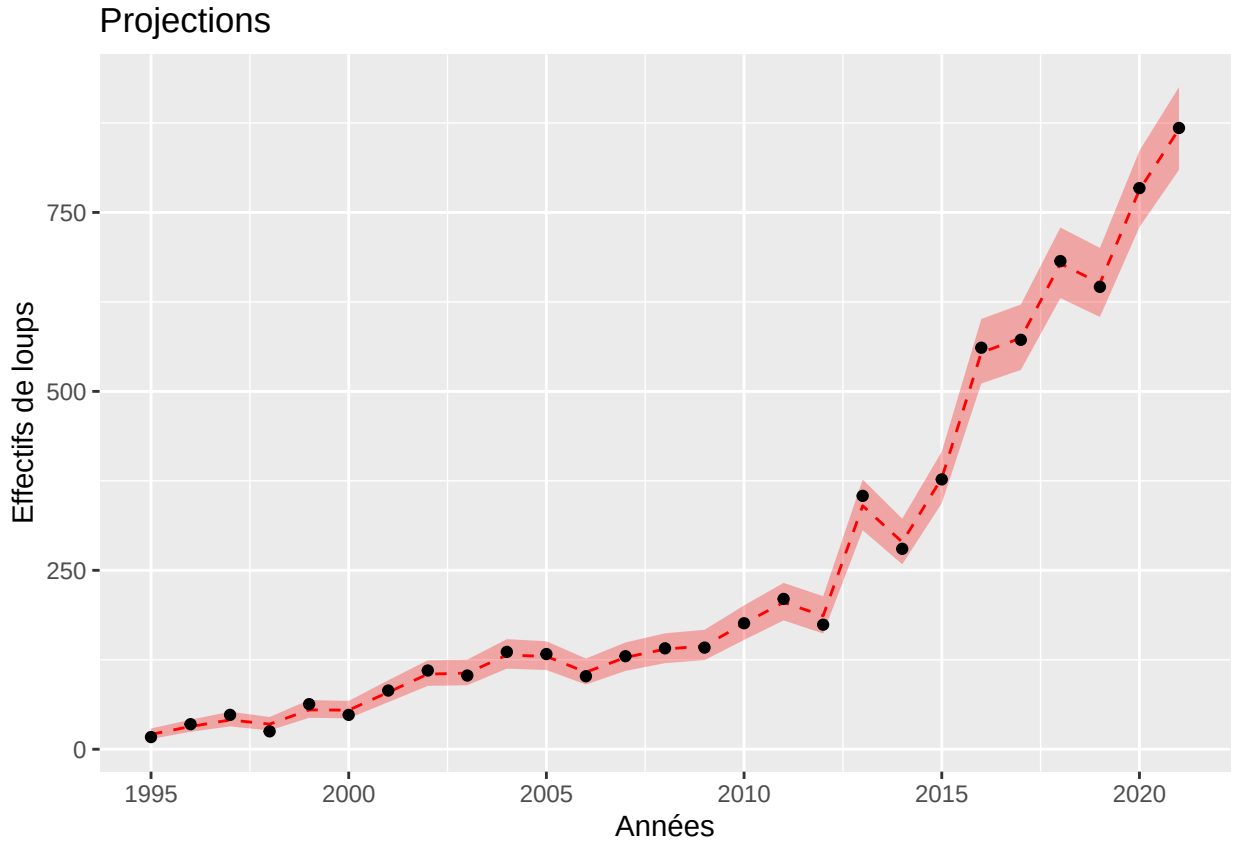
```
## N[25]     651.263  24.776 603.938 651.052 700.429 1.001  4500
## N[26]     781.808  27.432 729.465 781.600 836.358 1.001 15000
## N[27]     866.648  29.596 809.500 866.298 925.145 1.001 14000
## lambda      1.207   0.062   1.091   1.205   1.336 1.001 15000
## sigmaProc   0.250   0.054   0.163   0.244   0.373 1.001 15000
## tauProc    18.268   7.904   7.204  16.843  37.478 1.001 15000
## deviance  218.948   8.251 204.657 218.285 236.759 1.001  8900
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 34.0 and DIC = 253.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
wolf_mod$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
#  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
#  filter(str_detect(parameter, "lambda")) %>%
  ggplot() +
  aes(x = lambda) +
  geom_density() +
  geom_vline(xintercept = 1, lty = "dashed", color = "red") +
  labs(x = "Taux de croissance")
```




Ensuite les projections.

```
wolf_mod$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
  filter(str_detect(parameter, "N")) %>%
  group_by(parameter) %>%
  summarize(medianN = median(value),
            lci = quantile(value, probs = 2.5/100),
            uci = quantile(value, probs = 97.5/100)) %>%
  mutate(an = parse_number(parameter) + 1994) %>%
  arrange(an) %>%
  ggplot() +
  geom_ribbon(aes(x = an, y = medianN, ymin = lci, ymax = uci), fill = "red", alpha = 0.3) +
  geom_line(aes(x = an, y = medianN), lty = "dashed", color = "red") +
#  geom_point(aes(x = an, y = medianN), color = "red") +
  geom_point(data = bugs.data %>% as_tibble, aes(x = 1994 + 1:unique(nyears), y = y)) +
  labs(y = "Effectifs de loups",
       x = "Années",
       title = "Projections")
```




L'ajustement est top. Evidemment, si l'on veut faire des projections, les choses se compliquent puisqu'on n'a pas les prélèvements. On peut malgré tout faire des scénarios, comme dans le papier de Andren Harvest models of small populations of a large carnivore using Bayesian forecasting. J'ai exploré un peu plus en détails les analyses de ce papier, voir ici.

## Trucs à faire

Plusieurs petites choses.

- ~~Le fait de ne pas tenir compte des prélèvements me chagrine un peu aux entournures tout de même. Je ne me souviens plus des justifications apportées par Guillaume, j'ai sûrement raté des trucs. Je voudrais explorer l'idée d'avoir un seul modèle qui englobe les modèles exponentiel et avec freinage ($\theta$-logistique) et qui s'appuie sur la théorie du prélèvement en dynamique des populations. Jetez un coup d'oeil à l'annexe C qui commence à la page 97 du management plan de l'USGS pour l'ours polaire. Voir aussi le papier d'Henrik Andren sur le lynx : Andrén, H., Hobbs, N. T., Aronsson, M., Brøseth, H., Chapron, G., Linnell, J. D. C., Odden, J., Persson, J., and Nilsen, E. B.. 2020. Harvest models of small populations of a large carnivore using Bayesian forecasting. *Ecological Applications* 30(3):02063. 10.1002/eap.2063.~~
- ~~Je ne comprends pas bien le terme dit de densité-dépendance introduit via `Nproc[t] <- log(max(1, lambda * N[t-1]`~~ ~~Je simulerais bien des données selon ce modèle pour bien comprendre.~~
- Pourquoi utiliser une Poisson-Gamma un peu compliquée dans sa formulation plutôt que directement une Binomiale-négative?
- Il faut passer par une étape de prior predictive check pour être sûr que les priors induits sur les effectifs sont acceptables. Faire aussi une analyse de sensibilité.
- Explorer comment est choisi la variable $X$ de freinage. A minima, expliciter son choix, c'est-à-dire sur quelles années on suppose que le freinage a lieu. Pourquoi ne pas faire un modèle à seuil.
- Utiliser le wAIC plutôt que le DIC pour comparer le modèle exponentiel au modèle avec freinage. Ca se fait soit à la main dans Jags, soit via un passage en Nimble peu coûteux en général.
- Dommage de ne pas faire entrer les erreurs d'estimation obtenues par CMR dans les erreurs d'observation $\sigma^2_{\text{obs}}$ du modèle à espace d'états. Cela réduirait les incertitudes, en particulier sur les projections.