# Adapter les résultats de Harvest models of small populations of a large carnivore using Bayesian forecasting par Andrén et al. 2020 au modèle déjà existant de dynamique des populations de loups en France

Olivier Gimenez & Loïc Pages

12/01/2024

## Introduction

L'objectif de ce code est de modéliser la dynamique de population de loup en France et de prédir l'impact de la chasse sur celle-ci. Pour ce faire, on reprend ici le modèle de Andrén et al. 2020. Premièrement, on calcule la viabilité de la population entre les années 1995 à 2021 selon un modèle logistique. Puis on modélise le nombre optimal d'animaux à tuer pour maintenir la viabilité de la population.

### Préparatifs

On calcule la viabilité de la population selon un modèle exponentiel qui prend aussi en compte la chasse du loup.

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 4.3.0

## Loaded modules: basemod,bugs

##
## Attaching package: 'R2jags'

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

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

**Les données**

```r
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:

```r
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)
```

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

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

## 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 $N_t$:

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

```r
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])
  }

  # Projected population
  for (t in (nyears + 1):(nyears + 10)) {
    Nproc[t] <- log(max(1, lambda*(N[t-1])))
    N[t] ~ dlnorm(Nproc[t], tauProc)
  }
}
```

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)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 27
##    Unobserved stochastic nodes: 39
##    Total graph size: 236
##
## Initializing model
```

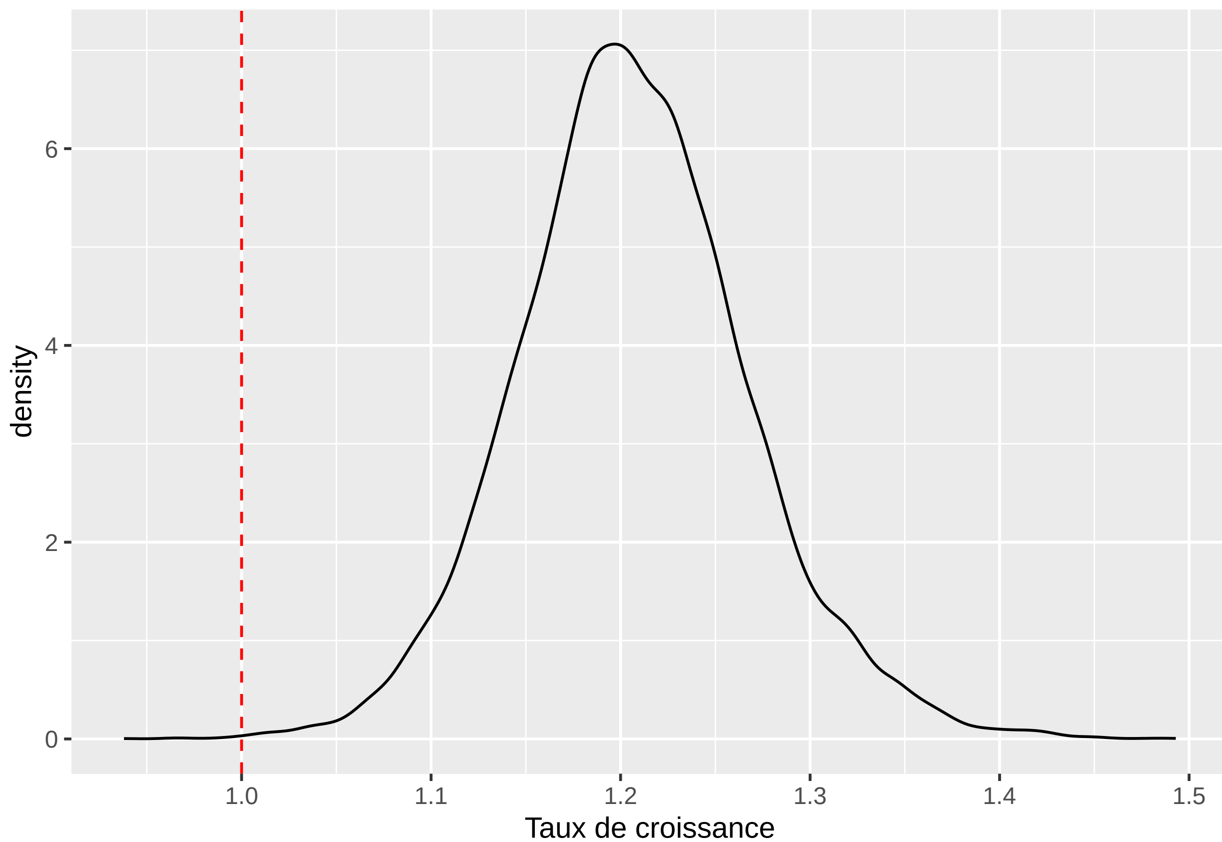
Jetons un coup d'oeil aux estimations.

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

```
## Inference for Bugs model at "/tmp/RtmpT28Sdp/model89ca5f5e06da.txt", fit using jags,
##  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]         21.051     3.780  14.247   20.841    28.945 1.001 15000
## N[2]         32.079     4.417  24.171   31.851    41.556 1.001 15000
## N[3]         41.124     5.286  31.698   40.740    52.430 1.001 15000
## N[4]         35.134     4.790  26.112   35.088    44.980 1.001 12000
## N[5]         55.300     6.214  44.079   54.960    68.185 1.001 15000
## N[6]         54.818     6.113  43.558   54.621    67.302 1.001 15000
## N[7]         80.050     7.745  65.807   79.729    96.260 1.001 14000
## N[8]        105.285     9.145  88.553  104.939   124.019 1.001 15000
## N[9]        106.733     9.182  89.566  106.536   125.404 1.001 15000
## N[10]       131.704    10.475 112.180  131.444   153.045 1.001 15000
## N[11]       130.128    10.213 110.895  129.824   151.111 1.001  5400
## N[12]       107.880     9.412  90.041  107.589   127.002 1.001 15000
## N[13]       128.515    10.265 109.487  128.116   149.506 1.001  5100
## N[14]       140.233    10.726 120.275  139.863   162.458 1.001 14000
## N[15]       144.717    10.914 124.305  144.414   166.960 1.001 15000
## N[16]       175.766    12.215 152.864  175.385   200.736 1.001 15000
## N[17]       205.460    13.364 180.271  205.080   232.847 1.001  7200
## N[18]       186.812    13.046 162.201  186.388   212.678 1.001 15000
## N[19]       340.606    18.140 305.881  340.321   377.261 1.001 15000
## N[20]       289.461    16.338 258.490  289.168   322.629 1.001 15000
## N[21]       378.656    18.665 343.121  378.165   416.289 1.001 15000
## N[22]       554.709    22.910 510.496  554.279   600.668 1.001 15000
## N[23]       574.831    23.229 530.281  574.590   621.108 1.001 15000
## N[24]       679.186    25.440 630.502  679.144   729.106 1.002  2800
## N[25]       650.614    24.671 602.415  650.351   700.436 1.001  7300
## N[26]       782.200    27.296 729.534  781.749   836.319 1.001 15000
## N[27]       867.397    29.114 811.589  866.883   926.639 1.001  4400
## N[28]      1082.217   291.643 613.815 1048.040  1754.638 1.001  8000
## N[29]      1357.632   546.399 592.706 1261.783  2674.521 1.001 15000
## N[30]      1713.301   903.832 603.629 1523.688  3959.802 1.001 15000
## N[31]      2148.029  1388.150 622.309 1830.200  5547.372 1.001 13000
## N[32]      2709.829  2027.744 657.280 2206.860  7963.417 1.001 11000
## N[33]      3427.372  3024.450 682.192 2665.226 10664.286 1.001 15000
## N[34]      4353.157  4626.880 715.700 3193.180 14898.126 1.001  9800
## N[35]      5576.229  6922.509 754.029 3846.329 20218.454 1.001  6200
## N[36]      7200.518 11184.243 809.213 4650.871 28155.485 1.001  6500
## N[37]      9195.155 15118.965 866.124 5602.346 38386.845 1.001  6900
## lambda        1.207     0.061   1.090    1.205     1.335 1.001  3900
## sigmaProc     0.250     0.054   0.164    0.243     0.372 1.001 15000
## tauProc      18.301     7.883   7.225   16.894    37.384 1.001 15000
## deviance    218.880     8.187 204.570  218.315   236.406 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 = 33.5 and DIC = 252.4
```

```
## DIC is an estimate of expected predictive error (lower deviance is better).
```

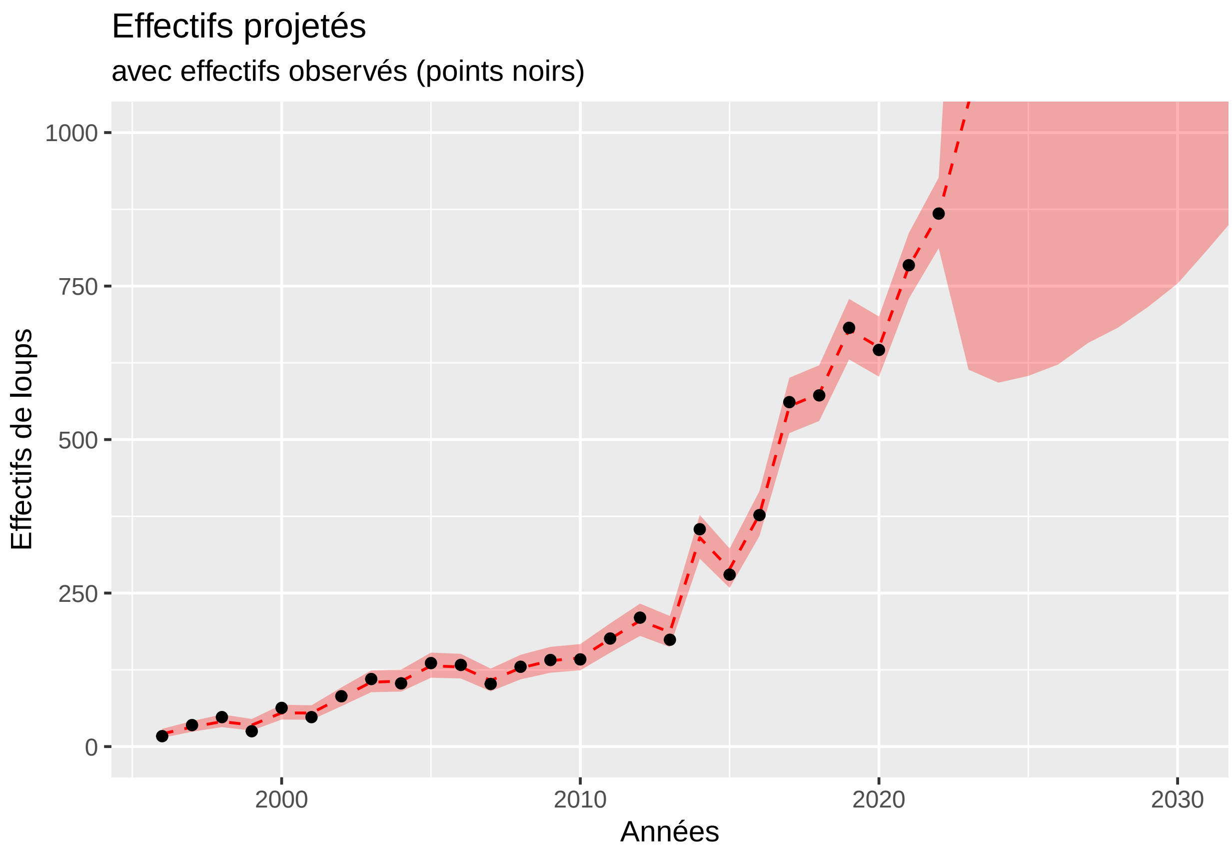
```
wolf_mod$BUGSoutput$sims.matrix %>%
  as_tibble() %>%
#  pivot_longer(cols = everything(),  values_to = "value", names_to = "parameter") %>%
#  filter(str_detect(parameter, "lambda")) %>%
  ggplot() +
  aes(x = lambda) +
  geom_density() +
  geom_vline(xintercept = 1, lty = "dashed", color = "red") +
  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) + 1995) %>%
  arrange(an) %>%
  ggplot() +
  geom_ribbon(aes(x = an, y = medianN, ymin = lci, ymax = uci), fill = "red", alpha = 0.3) +
```

```r
  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 = 1995 + 1:unique(nyears), y = y)) +
  coord_cartesian(xlim=c(1996,2030),ylim=c(0,1000))+
  labs(y = "Effectifs de loups",
       x = "Années",
       title = "Effectifs projetés",
       subtitle = "avec effectifs observés (points noirs)")
```




### Forecasting

Le modèle décrit l'évolution des effectifs à $t+1$ en fonction des effectifs à $t$ et permet donc de projeter les effectifs en 2021 en connaissant les effectifs de 2020 la dernière année du suivi, puis ceux de 2023 en utilisant les effectifs prédits pour 2022, et ainsi de suite. A chaque étape, il y a des erreurs qui s'accumulent. L'approche bayésienne a l'avantage de permettre de faire ces prédictions en reportant les incertitudes d'une année à l'autre. C'est ce qui fait des modèles à espace d'états en bayésien un outil très utile pour faire des projections.

Bien. Maintenant dans le modèle utilisé, la variable effectifs prélevés est supposée connue. Il s'agit d'une donnée, et par définition on ne la connait pas dans le futur. Il nous faut donc un modèle sur les effectifs prélevés, comme on en a un sur les effectifs comptés.

Andrén et al. proposent le modèle à espace d'états suivant :

$$H_t \sim \text{log-Normale}\left(\max(0, \log(b_0 + b_1 y_{t-1})), \sigma_q^2\right)$$

et

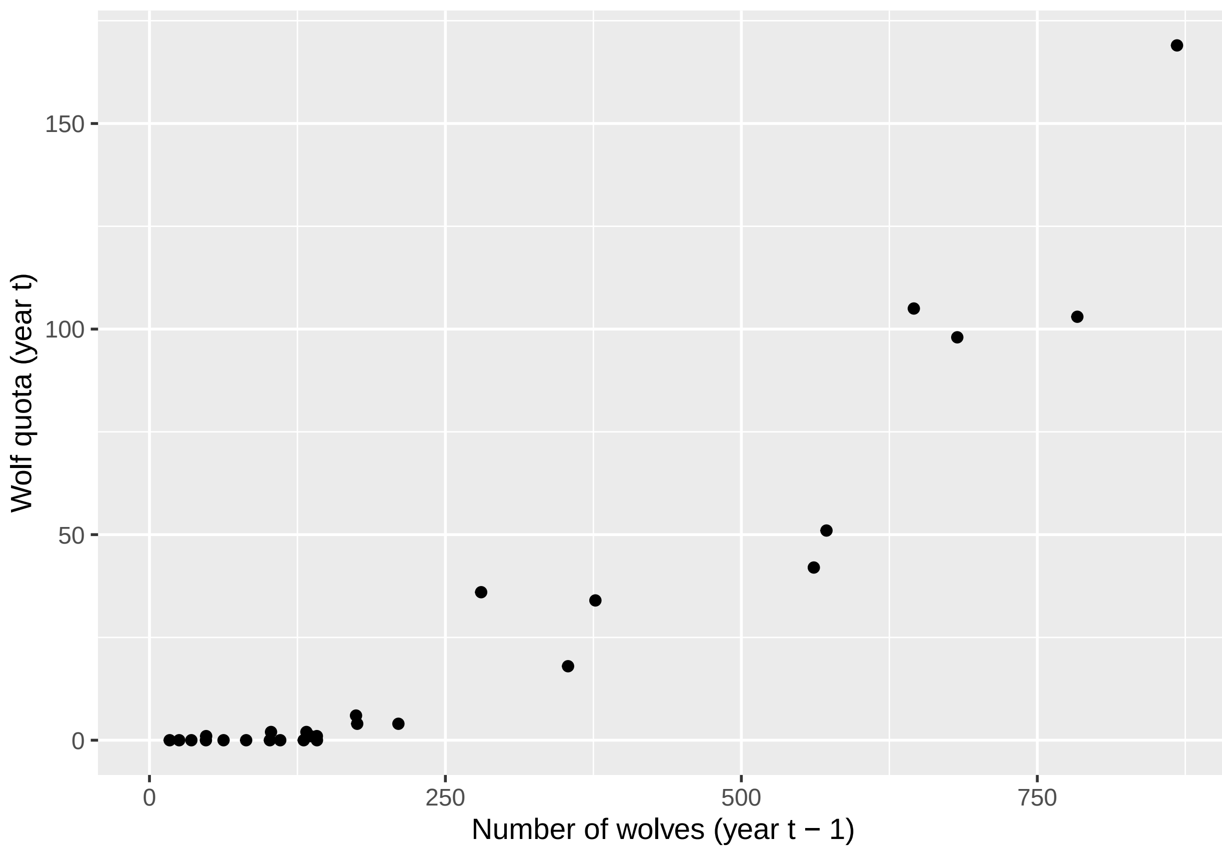
$$q_t \sim \text{Poisson}(H_t)$$

où $q_t$ est le quota observé au temps $t$ et $H_t$ l'effectif réel d'animaux prélevés. La prédiction du modèle est $H_t$ avec une erreur de processus $\sigma_q^2$.

On retrouve l'astuce utilisée par Guillaume pour forcer la moyenne de la normale à être supérieure ou égale à 0 avec le $\max(0, \log)$.

On a deux scénarios, ou bien un quota proportionnel aux effectifs comptés avec $b_0 = 0$ (modèle 1 : proportional quota setting strategy), ou bien des prélèvements qui augmentent proportionnellement, avec un quota nul en-dessous d'un seuil (modèle 2 : threshold quota setting strategy). Ce seuil $X$ se calcule en fixant $0 = b_0 + b_1 X$ soit $X = -b_0/b_1$. J'ai pas tout bien compris encore à ce scénario. Ca deviendra plus clair en essayant d'ajuster les modèles je suppose.

```
ggplot() +
  geom_point(aes(x = CMR, y = harvest), color = "black")  +
  expand_limits(x = 0, y = 0) +
  labs(x = "Number of wolves (year t - 1)",
       y = "Wolf quota (year t)")
```




### Modèle 1

Commençons par le modèle 1.

```
model1 <- function(){

  # Priors
  sigmaProc ~ dunif(0, 4)
  tauProc <- 1/sigmaProc^2
  b[1] ~ dnorm(0, 3)

  # Process model
  for (t in 1:(nyears)) {
    mu[t] <- log(b[1] * y[t])
    Hproc[t] <- max(0, mu[t])
    H[t] ~ dlnorm(Hproc[t], tauProc)
    }

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

}
```

On prépare les données.

```
bugs.data <- list(
    nyears = 27,
    y = CMR,
    q = harvest)
```

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

```
bugs.monitor <- c("b", "sigmaProc","H")
bugs.chains <- 3
bugs.inits <- function(){
    list(
    )
}
```

Allez zooh, on lance la machine!

```
mod1 <- jags(data = bugs.data,
                  inits = bugs.inits,
                  parameters.to.save = bugs.monitor,
                  model.file = model1,
                  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: 172
##
## Initializing model
```

Jetons un coup d'oeil aux estimations.

```r
print(mod1, intervals = c(2.5/100, 50/100, 97.5/100))
```

```
## Inference for Bugs model at "/tmp/RtmpT28Sdp/model89ca5371e5ff.txt", fit using jags,
##  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
## H[1]        0.487   0.557   0.015   0.303   1.993 1.001  8700
## H[2]        0.493   0.561   0.015   0.313   2.005 1.001 15000
## H[3]        0.539   0.597   0.018   0.338   2.210 1.001 13000
## H[4]        0.488   0.546   0.014   0.312   1.973 1.001 15000
## H[5]        0.584   0.633   0.020   0.383   2.312 1.001  4200
## H[6]        1.158   0.945   0.117   0.902   3.642 1.001 15000
## H[7]        0.638   0.676   0.020   0.427   2.491 1.001 15000
## H[8]        0.701   0.712   0.025   0.478   2.675 1.001 10000
## H[9]        2.160   1.370   0.393   1.861   5.628 1.001  4400
## H[10]       1.448   1.095   0.157   1.174   4.282 1.001 11000
## H[11]       2.229   1.389   0.410   1.939   5.682 1.001  3800
## H[12]       0.684   0.696   0.024   0.465   2.582 1.001 15000
## H[13]       0.746   0.750   0.027   0.513   2.763 1.001  7800
## H[14]       1.439   1.085   0.162   1.169   4.204 1.001  9900
## H[15]       0.771   0.769   0.028   0.541   2.846 1.001  7100
## H[16]       4.082   1.918   1.230   3.779   8.626 1.001  5100
## H[17]       4.148   1.940   1.257   3.833   8.760 1.001 15000
## H[18]       5.897   2.342   2.265   5.584  11.352 1.001 15000
## H[19]      17.743   4.168  10.606  17.450  26.922 1.001  9800
## H[20]      35.318   5.902  24.754  34.993  48.001 1.001 13000
## H[21]      33.516   5.743  23.309  33.223  45.684 1.001 15000
## H[22]      41.539   6.376  30.013  41.159  55.086 1.001  7700
## H[23]      50.538   7.116  37.625  50.217  65.273 1.001 14000
## H[24]      97.437   9.841  78.900  97.185 117.779 1.001 15000
## H[25]     104.244  10.084  85.546 103.887 125.171 1.001  9900
## H[26]     102.275  10.010  83.779 101.896 123.091 1.001 15000
## H[27]     168.280  13.050 143.535 167.950 195.092 1.001 15000
## b           0.026   0.009   0.012   0.025   0.048 1.001 15000
## sigmaProc   1.704   0.408   1.087   1.642   2.668 1.001 15000
## deviance  102.621   7.293  90.165 102.088 118.690 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 = 26.6 and DIC = 129.2
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Le paramètre $b_1$ est estimé être :

```
mod1$BUGSoutput$mean$b
```

```
## [1] 0.02613501
```

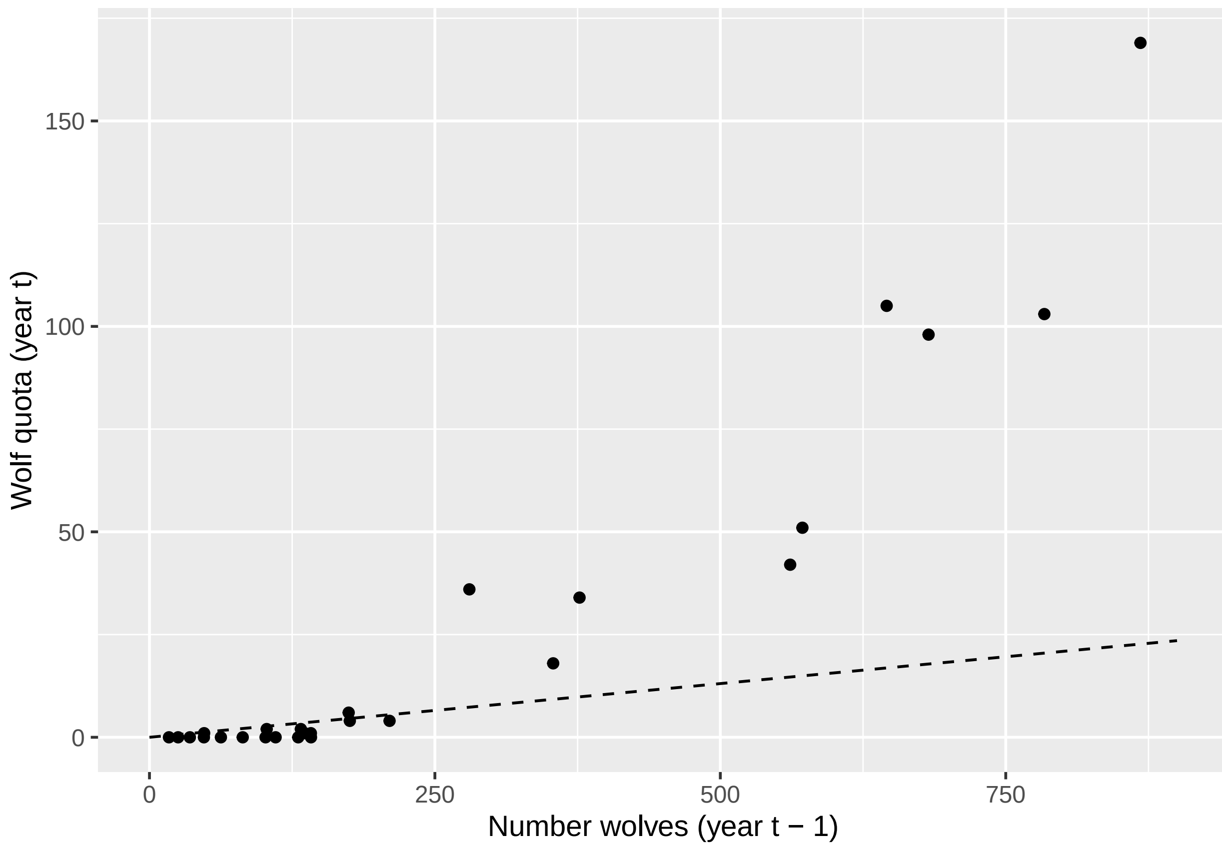
Graphiquement, on obtient.

```
grid <- seq(0, 900, length.out = length(CMR))

ggplot() +
  geom_point(aes(x = CMR, y = harvest), color = "black") +
  geom_line(aes(x = grid, y = mod1$BUGSoutput$mean$b * grid), color = "black", lty = "dashed") +

  expand_limits(x = 0, y = 0) +
  labs(x = "Number wolves (year t - 1)",
       y = "Wolf quota (year t)")
```

```
## Warning in mod1$BUGSoutput$mean$b * grid: Recycling array of length 1 in array-vector arithmetic is d
##   Use c() or as.vector() instead.
```




**Modèle 2**

On écrit le modèle. La différence avec le modèe 1 est qu'on estime une ordonnée à l'origine.

```r
model2 <- function(){

  # Priors
  sigmaProc ~ dunif(0, 4)
  tauProc <- 1/sigmaProc^2
  b[1] ~ dnorm(0, 1/3000)
  b[2] ~ dnorm(0, 1/3000)
  # Process model
  for (t in 1:(nyears)) {
    mu[t] <- log(b[1] + b[2] * y[t])
#    mu[t] <- log(b[1] + b[2] * y[t]) * index[t]
#    index[t] <- - 1000 * step(y[t] + b[1] / b[2]) # step(x) = 1 if x >= 0
#    index[t] <- step(q[t]) # step(x) = 1 if x >= 0
#    mu[t] <- log(b[1] + b[2] * y[t])
    Hproc[t] <- max(0, mu[t])
    H[t] ~ dlnorm(Hproc[t], tauProc)

# les lignes de code suivantes donnent un ajustement pas mal, mais
# sauf qu'à l'approche de census == 0 on a harvest == 0
#    Hproc[t] <- log(b[1] + b[2] * y[t])
#    H[t] ~ dlnorm(Hproc[t], tauProc)
    }

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

}
```

On prépare les données pour la Suède.

```r
bugs.data <- list(
    nyears = 27,
    y = CMR,
    q = harvest)
```

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

```r
bugs.monitor <- c("b", "sigmaProc")
bugs.chains <- 3
bugs.inits <- function(){
    list(
    )
}
```

Allez zooh, on lance la machine!

```r
mod2 <- jags(data = bugs.data,
                  inits = bugs.inits,
                  parameters.to.save = bugs.monitor,
                  model.file = model2,
```

```
                  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: 30
##    Total graph size: 201
##
## Initializing model
```

Jetons un coup d'oeil aux estimations.

```
print(mod2, intervals = c(2.5/100, 50/100, 97.5/100))
```

```
## Inference for Bugs model at "/tmp/RtmpT28Sdp/model89ca4000e8a0.txt", fit using jags,
##  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
## b[1]      -24.314   5.099 -35.698 -23.761 -16.349 1.011  1200
## b[2]        0.163   0.025   0.120   0.161   0.218 1.002  2100
## sigmaProc   0.396   0.127   0.209   0.376   0.706 1.001  5500
## deviance  107.806   5.868  97.840 107.318 120.845 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 = 17.2 and DIC = 125.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Les paramètres $b$ sont estimés comme suit.

```
mod2$BUGSoutput$mean$b
```

```
## [1] -24.3144165   0.1627461
```

Le ratio se calcule comme suit.

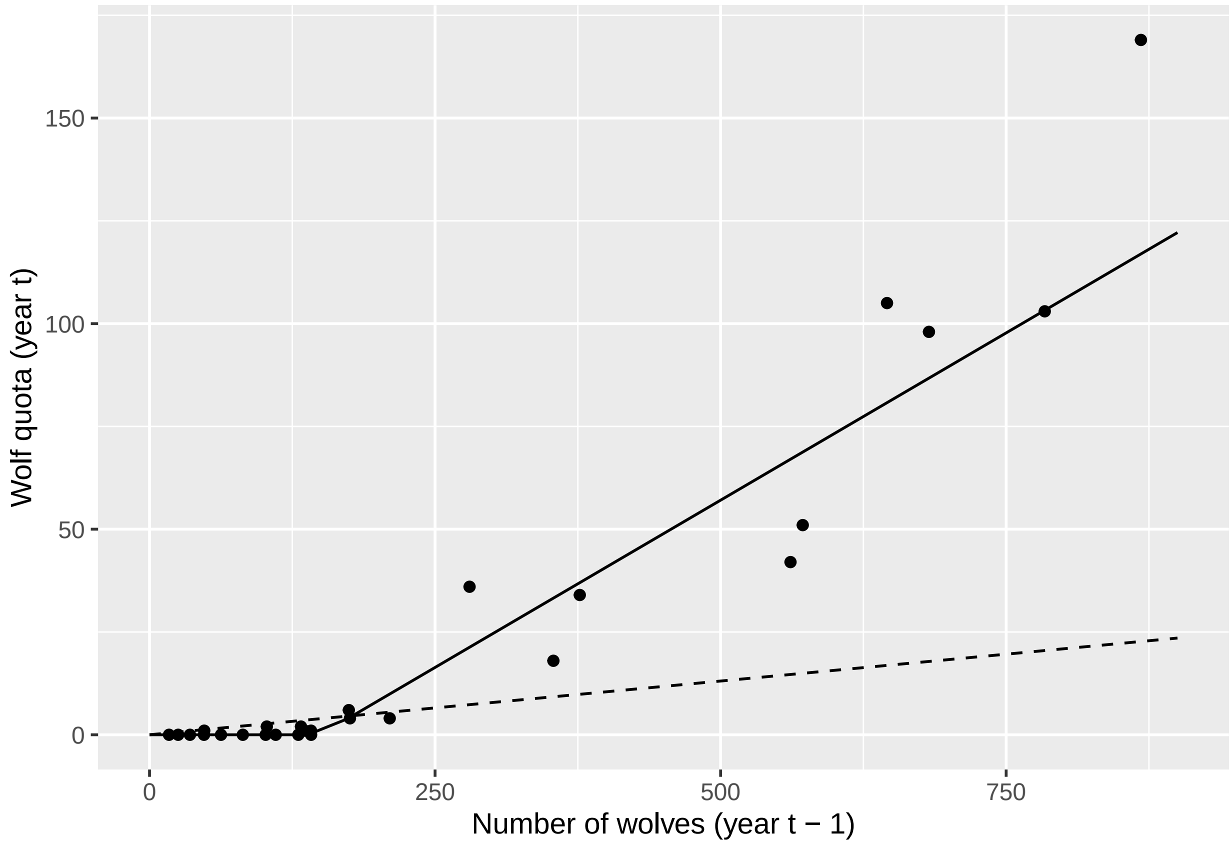
```
- mod2$BUGSoutput$mean$b[1] / mod2$BUGSoutput$mean$b[2]
```

```
## [1] 149.4009
```

Graphiquement, on obtient.

```
grid <- seq(0, 900, length.out = length(CMR))
treshold = - mod2$BUGSoutput$mean$b[1] / mod2$BUGSoutput$mean$b[2]
ggplot() +
  geom_point(aes(x = CMR, y = harvest), color = "black") +
  geom_line(aes(x = grid, y = mod1$BUGSoutput$mean$b * grid), color = "black", lty = "dashed") +
  geom_line(aes(x = grid, y = if_else(grid < treshold, 0, (mod2$BUGSoutput$mean$b[1] +  mod2$BUGSoutput$
  expand_limits(x = 0, y = 0) +
  labs(x = "Number of wolves (year t - 1)",
       y = "Wolf quota (year t)")
```

```
## Warning in mod1$BUGSoutput$mean$b * grid: Recycling array of length 1 in array-vector arithmetic is d
##   Use c() or as.vector() instead.
```




**Conclusion**

On constate que le meilleur scénario pour déterminer le quota du nombre de loups à tuer est le modèle à seuil (ici à 149).

Exemple de nombre de loup à tuer pour une population de 868 individus.

```
HWolf=function(n){
  if (n>149){
  return(as.integer(-24+0.1618981*n))}
  else {return(0)}}
```

```r
HWolf(868)
```

```
## [1] 116
```

# Amélioration de la projection

A l'aide de l'estimation du nombre de loups à tuer selon la taille de la population, on modélise la projection des effecitfs en tenant compte des prélèvements.

```r
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])
  }

  # Projected population
  for (t in (nyears + 1):(nyears + 10)) {
    mu[t] <- lambda * (N[t-1] - -24+0.1618981*N[t-1])
    Nproc[t] <- log(max(1, mu[t]))
    N[t] ~ dlnorm(Nproc[t], tauProc)
  }
}
```

On prépare les données.

```r
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(
    )
}
```

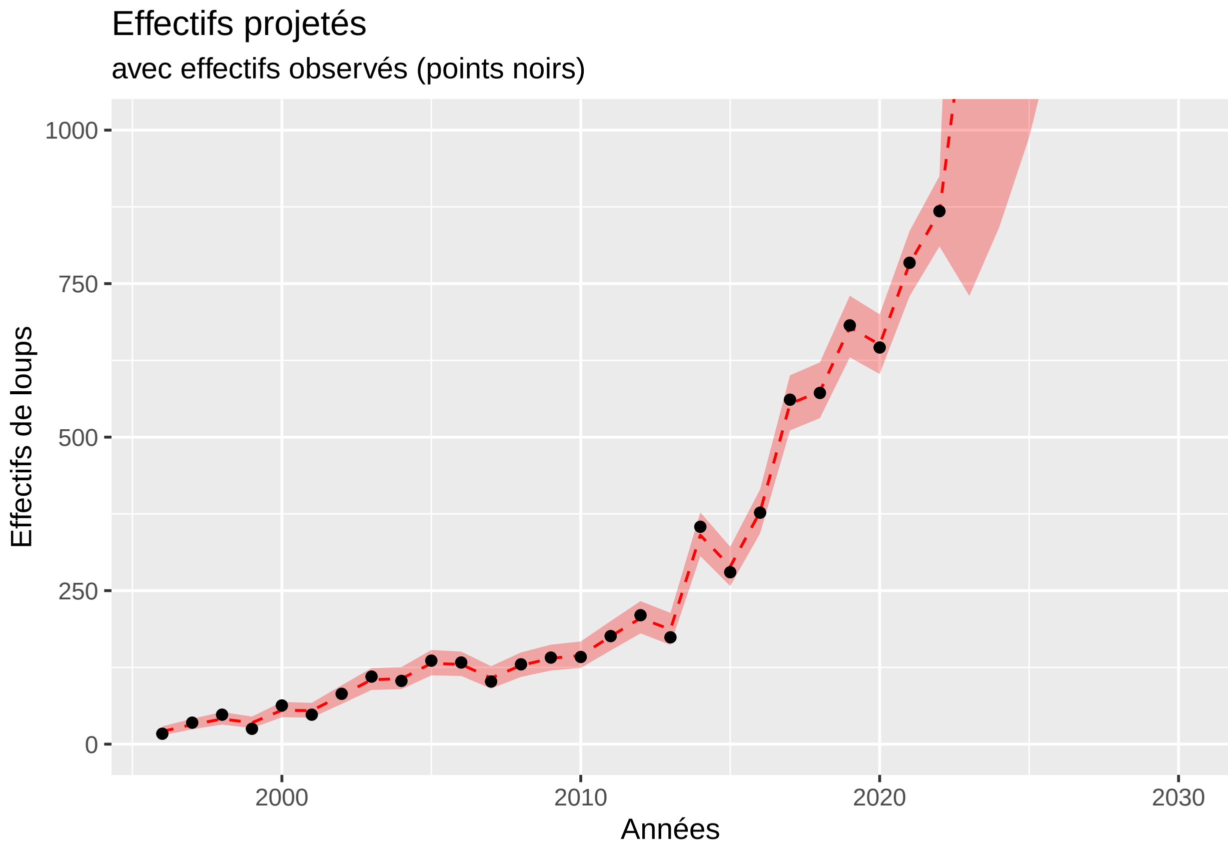
Allez zooh, on lance la machine!

```
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: 39
##    Total graph size: 269
##
## Initializing model
```

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) + 1995) %>%
  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 = 1995 + 1:unique(nyears), y = y)) +
  coord_cartesian(xlim=c(1996,2030),ylim=c(0,1000))+
  labs(y = "Effectifs de loups",
       x = "Années",
       title = "Effectifs projetés",
       subtitle = "avec effectifs observés (points noirs)")
```



Effectifs projetés
avec effectifs observés (points noirs)
Effectifs de loups
1000
750
500
250
0
2000
2010
2020
2030
Années