Présentation du contexte

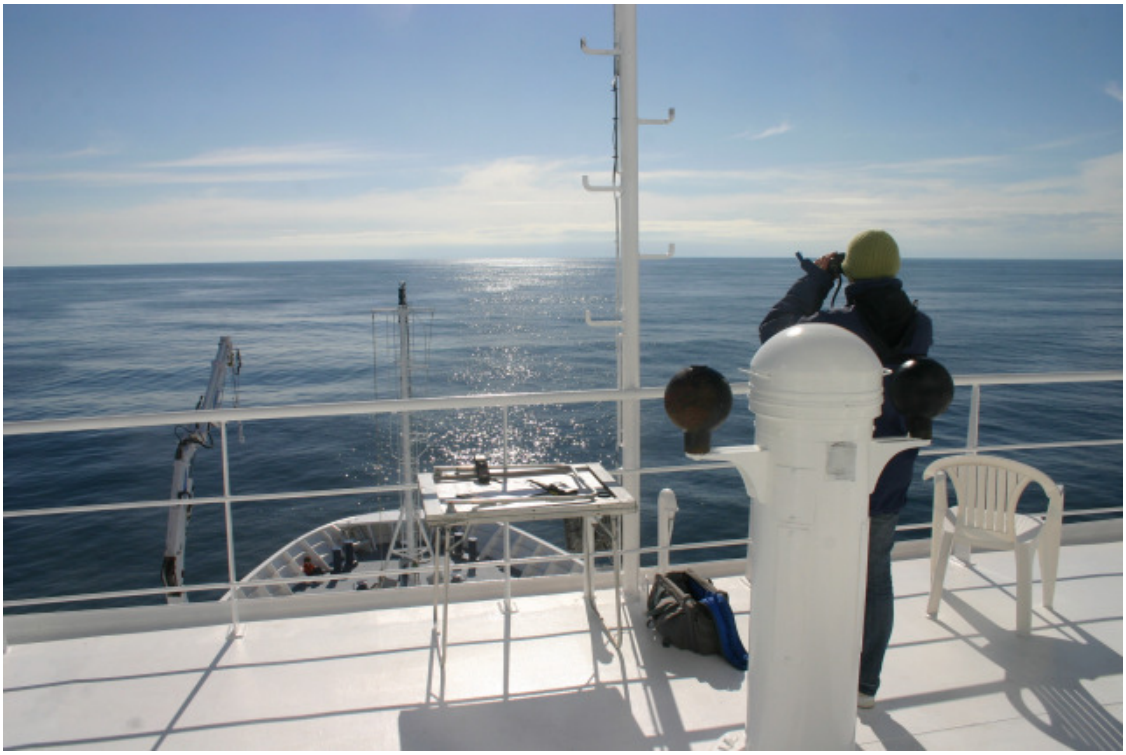
Objectifs

- Ajuster un modèle linéaire généralisé sur des données de comptages
- Vérifier l'ajustement du modèle grâce à la technique du "posterior predictive check"
- Changer la vraisemblance des données pour tenir compte de la surdispersion
- Ajouter des effets aléatoires
- Ajouter un effet spatial

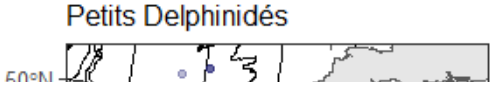
Estimer la taille de groupe du dauphin commun dans le golfe de Gascogne

Le dauphin commun ($Delphinus\ delphis$) est l'espèce la plus abondante de mammifères marins dans le Golfe de Gascogne (Laran et al, 2017).

Depuis 2004, les campagnes océanographiques de l'Ifremer sur le navire Thalassa permettent de recenser et dénombrer les dauphins communs.

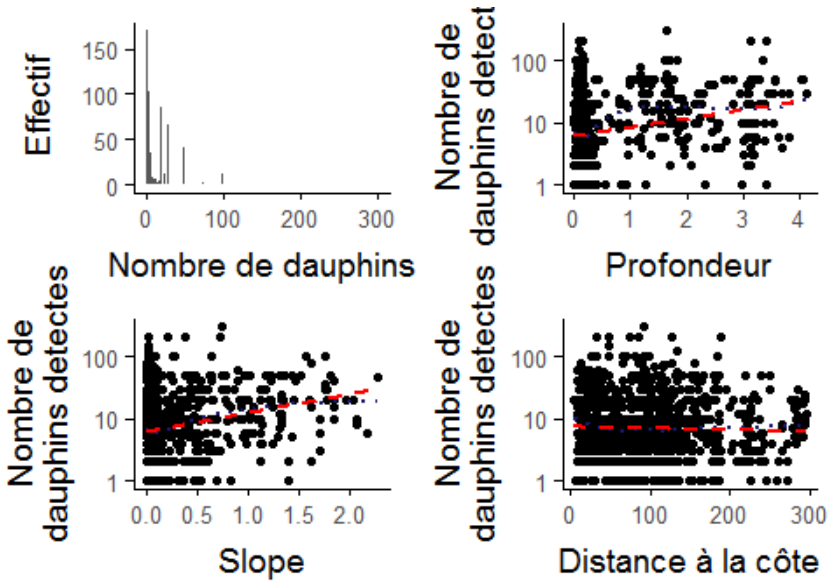


Les données collectées représentent 1269 observations de petits delphinidés (principalement des dauphins communs mais aussi quelques dauphins bleu et blanc $Stenella\ coeruleoalba$).



Question: Quels sont les facteurs déterminants de l'abondance de dauphins dans le Golfe de Gascogne?

Le but de l'analyse est ici de corréler le nombre de dauphins (la variable réponse) à un ensemble de covariables environnementales (par exemple la profondeur/bathymétrie, $etc.$) afin d'expliquer les variations observées.



Les données y_i sont des entiers naturels et la première approche est de considérer un modèle poissonien

$$y_i^{\text{obs}} \sim \mathcal{P}(\lambda_i)$$

où $\log(\lambda_i) = \beta_0 + \beta_1 \times \text{Bathy}_i + \dots$

Début de l'analyse

Charger les données et les bibliothèques de fonctions nécessaires à l'analyse

In [ ]:
```
lapply(c("ggplot2", "ggthemes", "sp", "sf", "loo"), library, character.only=TRUE
)

### theme for graphics
theme_set(theme_bw(base_size = 20))

### clean up
rm(list = ls())
getwd()
WorkDir <- paste(getwd(), "dauphin_v2", sep = "/")
DataDir <- paste(WorkDir, "data", sep = "/")
OutDir <- paste(WorkDir, "output", sep = "/")

### load the data
load(paste(DataDir, "Filtered_data.RData", sep = "/"))
```

In [ ]:
```
### view the different objects in your environment
ls()
```

In [ ]:
```
### dataframe of observations
head(obs)
```

In [ ]:
```
### Exploratory data analysis
ggplot(data = obs,
       aes(x = nombre)
       ) +
  geom_histogram(breaks = 0:300) +
  ylab("Effectif") + xlab("Nombre de dauphins") +
  theme(plot.title = element_text(lineheight = 0.8, face = "bold"),
        axis.text = element_text(size = 10)
        )
```

Travaux préliminaires

L'expérience nous pousse à utiliser une loi de Poisson, commune pour représenter les comptages. Les nombres observés étant toujours positifs, on peut décaler d'une unité. Il est d'usage de centrer et de réduire les variables explicatives. L'approche fréquentiste consisterait à *caler* un glm, ce qu'on fait.

Il nous faut écrire le modèle suivant:

$$y_i^{\text{obs}} \sim \mathcal{P}(\lambda_i)$$

avec $\log(\lambda_i) = \beta_0 + \beta_1 \times \text{Bathy}_i + \beta_2 \times \text{Pente}_i + \beta_3 \times \text{DistanceCote}_i$

In [ ]:
```
obs$nombreShift <- obs$nombre - 1
dauphin2.glm = glm(nombreShift ~ scale(DepthGebco) + scale(Slope) + scale(DistCo
t),
                   family = "poisson", data = obs
                   )
summary(dauphin2.glm)
dauphin2.glm$coefficients
```

Codage du modèle en bayésien avec `Jags`

Écrivons dans un premier temps le modèle avec le language spécifique à `Jags`. Ce language est structuré et la spécification d'un modèle se fait grâce à un ensemble de blocs de type BUGS:

Priors

Dans le paradigme bayésien, les inconnues sont munies de lois a priori qu'il nous faut spécifier. On prendra

$\beta_0 \sim \mathcal{N}(0.0, 1.0)$

$\beta_1 \sim \mathcal{N}(0.0, \frac{\log 2}{2})$

$\beta_2 \sim \mathcal{N}(0.0, \frac{\log 2}{2})$

$\beta_3 \sim \mathcal{N}(0.0, \frac{\log 2}{2})$

In [ ]:
```
library(R2jags)

model.jags <- '
model{
  # Prior
  intercept ~ dnorm(0.0, hyperparam_inter)
  slope[1] ~ dnorm(0.0, hyperparam_slope[1])
  slope[2] ~ dnorm(0.0, hyperparam_slope[2])
  slope[3] ~ dnorm(0.0, hyperparam_slope[3])

  # Likelihood
  for (i in 1:n_obs) {
    lambda[i] <- exp(intercept + slope[1] * X1[i] + slope[2] * X2[i] + slope[3]
* X3[i])
    Y[i] ~ dpois(lambda[i])
  }
}
'
```

Préparer les *données* et initialiser les chaînes (en trichant un peu)

In [ ]:
```
data.jags <- list(n_obs = nrow(obs),
                  Y = obs$nombreShift,
                  X1 = as.vector(scale(obs$DepthGebco)),
                  X2 = as.vector(scale(obs$Slope)),
                  X3 = as.vector(scale(obs$DistCot)),
                  hyperparam_inter = 1.0,
                  hyperparam_slope = 1/rep(log(2)/2, 3)^2
                  )
# Inits function
# On triche un peu: on va prendre les estimations de glm
inits.jags <- function(idchain, glm) {
  list(intercept = rnorm(1, glm$coefficients[1], sqrt(10 * diag(vcov(glm)))[1]),
       slope = rnorm(3, glm$coefficients[2:4], sqrt(10 * diag(vcov(glm)))[2:4])
       )
}
```

On nomme les inconnues, on va spécifier le nombre d'itérations puis lancer la machinerie d'inférence. On met les sorties au format de sortie de BUGS avant de les regarder sommairement.

In [ ]:
```
# parameter of inferential interest
params.jags <- c("intercept", "slope")

# MCMC settings
ni <- 2000 # TOTAL Number of iterations including Burn in
nb <- 1000 #Number of iterations for Burn in
nt <- 1
nc <- 3

# Load/check model, load data and inits
# Start Gibbs sampler
temp <- jags(data = data.jags,
             inits = lapply(1:nc, inits.jags, glm = dauphin2.glm),
             param = params.jags,
             model.file = textConnection(model.jags),
             n.chains = nc,
             n.burnin = nb,
             n.iter = ni,
             n.thin = nt
             )

out = temp$BUGSoutput
# Inferences
# After Burn in period let's work inference assuming ergodic regime is reached
print(out$summary, dig = 2)
```

Avant toute chose

On doit vérifier (rapidement) la convergence du modèle, par exemple en regardant le diagnostic de Gelman-Rubin.

Prendre un moment pour comprendre la construction de cette statistique d'analyse de variance pour les séries dans la partie théorie de l'aide sous R du programme: *?coda::gelman.diag*

In [ ]:
```
library(coda)
gelman.diag(x=out, confidence = 0.95, transform=FALSE, autoburnin=TRUE)
gelman.plot(x=out)
```

Visualisons la trace des paramètres : on peut voir que les paramètres convergent rapidement (au bout de seulement quelques itérations).

In [ ]:
```
traceplot(out)
```

Visualisation des lois a posteriori

On peut faire appel aux potentialités graphiques de R pour représenter les lois a posteriori. Voici un exemple avec ggplot

In [ ]:
```
library(tidyverse)

other =
    tibble(
      param = paste0("slope[",1:3,"]"),
      class = "slope",
      value = dauphin2.glm$coefficients[2:4],
      type = "mle"
    ) %>%
      rbind(list("intercept", "intercept", dauphin2.glm$coefficients[1], "mle"))

out$sims.matrix[,-1] %>% as_tibble() %>%
  tidyr::gather(param, value) %>%
  mutate(class = stringr::str_replace(param, "\\[\\d+\\]","")) %>%
  ggplot(aes_string(x="value", fill="class")) +
  geom_density(alpha=0.5) +
  facet_grid(param ~ .)  +
  geom_vline(data = other, aes_string(xintercept="value", color="type", linetype
="type"), size=0.8) +
  scale_colour_manual(values=c("red","purple")) +
  guides(fill=FALSE)+labs(x="valeur de l'inconnue", y="densité")

#essayer   facet_grid(~param, scales = "free")
```

Exercice

- dans la matrice des sorties jags , les 3 chaines sont empilées à la suite. Créer une nouvelle variable catégorielle donnant le numéro de la chaine
- refaire sous *ggplot2* l'équivalent de *traceplot*

Critère d'information

Pour calculer le critère d'information de Watanabee, on a besoin de calculer la log vraisemblance sous le modèle. On fait appel au package loo.

In [ ]:
```
### calcul du WAIC
## besoin de calculer la log vraisemblance sous le modèle
library(loo)
y_rep_1 <- log_lik1 <- array(NA, dim = c(nc * (ni - nb)/nt, nrow(obs)))

for(i in 1:nrow(obs)) {
  linpred <- out$sims.list$intercept +
    out$sims.list$slope[, 1] * data.jags$X1[i] +
    out$sims.list$slope[, 2] * data.jags$X2[i] +
    out$sims.list$slope[, 3] * data.jags$X3[i]
  log_lik1[, i] <- dpois(data.jags$Y[i], lambda = exp(linpred), log = TRUE)
  y_rep_1[, i] <- rpois(nrow(y_rep_1), lambda = exp(linpred)) + 1
}; rm(i, linpred)

loo::waic(log_lik1)
loo::loo(log_lik1)
```

Posterior Predictive Checks

On va extraire directement les valeurs prédites y^{rep_i}

In [ ]:
```
y_rep_poisson_1 <- y_rep_1
dim(y_rep_poisson_1)
```

Cet objet est une matrice de 3000 lignes et 1269 colonnes, soit une colonne pour chaque donnée et une ligne pour chaque itération.

In [ ]:
```
hist(y_rep_poisson_1[, 10], las = 1, xlab = quote(y[10]^rep), main = "PPC for th
e tenth datum")
abline(v = obs$nombreShift[10], lty = 2, col = "red", lwd = 2)
```

On peut par exemple calculer la probabilité $Pr(y_{10}^{\text{rep}} > y_{10}^{\text{obs}})$

In [ ]:
```
round(mean(ifelse(y_rep_poisson_1[, 10] > (obs$nombreShift[10]), 1, 0)),dig=3)
```

On peut aussi comparer les histogrammes de l'ensemble des données (observées et prédites) au lieu de regarder donnée par donnée.

In [ ]:
```
hist(y_rep_poisson_1[1, ], las = 1, main = "Iteration 1", breaks = c(0:100))
```

On va moyenner l'histogramme sur l'ensemble des itérations en faisant appel à une fonction

In [ ]:
```
# rootograms
rootogram <- function(countdata, y_rep, min_obs = 1) {
  f_histogram <- function(x, max_obs) { table(factor(x, levels = min_obs:max_obs
)) }
  max_obs <- max(countdata, as.numeric(y_rep))
  dd <- ggplot(data.frame(mids = (min_obs:max_obs + (min_obs + 1):(max_obs + 1))
/2,
                          y_obs = as.numeric(f_histogram(x = countdata, max_obs
= max_obs)),
                          y_rep = apply(apply(y_rep, 1, f_histogram, max_obs = m
ax_obs), 1, mean)
                          ),
               aes(x = mids, y = y_rep)
               ) +
    geom_rect(aes(xmin = mids - 0.5, xmax = mids + 0.5,
                  ymax = y_obs, ymin = 0),
              fill = "lightgrey", color = grey(0.6), alpha = 0.5) +
    geom_line(aes(x = mids, y = y_rep), color = "black", size = 1.0) +
    scale_y_continuous(name = "Count") +
    scale_x_continuous(name = "Nb of dolphins", breaks = c(0, seq(1, max_obs, ma
x_obs/50))) +
    guides(size = "none") +
    theme(plot.title = element_text(lineheight = 0.8, face = "bold"),
          axis.text = element_text(size = 12),
          strip.text = element_text(size = 12),
          strip.background = element_rect(fill = grey(0.95))
          )
  return(dd)
}

rootogram_model_1 <- rootogram(countdata = obs$nombre, y_rep = y_rep_1+1)
rootogram_model_1 +
  coord_cartesian(xlim = c(1, 50))
```

Un ajustement très (trop?) optimiste quant aux incertitudes

Nous allons reconstruire les effets des prédicteurs avec les gammes d'incertitude issues de la connaissance a posteriori des paramètres. *Trop beau pour être honnête*?

In [ ]:
```
plot_relationships <- function(data_df, jagsfit, cov_name) {
  X <- data_df[, cov_name]
  x_scale <- apply(X, 2, function(x) {c(mean(x), sd(x))})
  x_range <- apply(apply(X, 2, scale), 2, range)
  stdX <- cbind(rep(1, 1e4),
                apply(x_range, 2, function(vec) {seq(vec[1], vec[2], length.out
= 1e4)})
                )
  beta <- cbind(jagsfit$sims.list$intercept,
                jagsfit$sims.list$slope
                )
  if(ncol(stdX) != ncol(beta)) { stop("Please check model and covariable names:\
n\tdimension mismatch between slope parameters and covariables") }
  else {
    dd <- data.frame(x = numeric(0),
                     y = numeric(0),
                     lower = numeric(0),
                     upper = numeric(0),
                     what = character(0),
                     covariable = character(0)
                     )
    for(j in 1:length(cov_name)) {
      Xmat <- cbind(rep(1, 1e4),
                    matrix(0, nrow = 1e4, ncol = length(cov_name))
                    )
      Xmat[, j+1] <- stdX[, j+1]
      linpred <- beta %*% t(Xmat)
      dd <- rbind(dd,
                  data.frame(x = Xmat[, j+1] * x_scale[2, cov_name[j]] + x_scale
[1, cov_name[j]],
                             y = apply(linpred, 2, mean),
                             lower = apply(linpred, 2, stats::quantile, prob = 0
.025),
                             upper = apply(linpred, 2, stats::quantile, prob = 0
.975),
                             what = rep("linear predictor", 1e4),
                             covariable = rep(cov_name[j], 1e4)
                             ),
                  data.frame(x = Xmat[, j+1] * x_scale[2, cov_name[j]] + x_scale
[1, cov_name[j]],
                             y = apply(exp(linpred), 2, mean),
                             lower = apply(exp(linpred), 2, stats::quantile, pro
b = 0.025),
                             upper = apply(exp(linpred), 2, stats::quantile, pro
b = 0.975),
                             what = rep("response", 1e4),
                             covariable = rep(cov_name[j], 1e4)
                             )
                  )
    }
    dd$what <- factor(as.character(dd$what), levels = c("response", "linear pred
ictor"))
    dd$covariable <- factor(as.character(dd$covariable), levels = cov_name)

    the_plot <- ggplot(data = dd,
                       aes(x = x, y = y)
                       ) +
      geom_ribbon(aes(x = x, ymin = lower, ymax = upper),
                  fill = "lightgrey", color = grey(0.6), alpha = 0.5
                  ) +
      geom_line(aes(x = x, y = y), color = "black", size = 1.0) +
      scale_y_continuous(name = "y") +
      scale_x_continuous(name = "Covariable") +
      facet_grid(what~covariable, scales = "free") +
      theme(plot.title = element_text(lineheight = 0.8, face = "bold"),
            axis.text = element_text(size = 12),
            strip.text = element_text(size = 12),
            strip.background = element_rect(fill = grey(0.95))
            )
```

Travaux dirigés R

- Retour à la définition: Construire une fonction qui calcule le WAIC à la main à partir d'une matrice comme log_lik1. Soit θ le vecteur des paramètres inconnus d'un modèle d'observations $f(.|\theta)$. Soit $(y_1, \ldots, y_n)$ un échantillon de réalisations de $f(.|\theta)$. Soit $(\theta^1, \ldots, \theta^S)$ un échantillon de valeurs issues de la loi a posteriori jointe de θ et générées à l'aide d'un algorithme MCMC. Le critère d'information WAIC (Watanabe-Akaike ou Widely Applicable Information Criterion) du modèle peut alors être approximé (voir Bayesian Data Analysis, Gelman et al., 2016) en remplaçant les espérances *a posteriori* par des sommes sur les tirages dans le posterior:

$$\hat{WAIC} = -2\hat{llpd} + 2p_{\hat{WAIC}}$$

avec

$$\hat{llpd} = \sum_{i=1}^{n} log\left(\frac{1}{S}\sum_{s=1}^{S} f(y_i|\theta^s)\right)$$

et

$$p_{\hat{WAIC}} = \sum_{i=1}^{n} V_{s=1}^{S}\left(logf(y_i|\theta^s)\right)$$

$V_{s=1}^{S}(a_s) = \frac{1}{S-1}\sum_{s=1}^{S}(a_s - \bar{a})^2, \bar{a} = \frac{1}{S}\sum_{s=1}^{S} a_s.$

- On pourra également y ajouter le DIC comme sortie (qu'on comparera aux résultats de la fonction *dic.samples* de *rjags*).
- Faire un modèle à 2 covariables sous Jags (par exemple, supprimer la distance à la côte).
- Est-ce que ce nouveau modèle est meilleur que le précédent? On pourra utiliser le critère WAIC pour faire une comparaison de modèle.
- Les critères DIC et WAIC mènent-ils aux mêmes conclusions? Quel(s) avantage(s) voyez-vous à l'utilisation du critère WAIC par rapport au critère DIC?

Pour aller plus loin: Les modèles fonctionnent mais l'ajustement est atroce, que faire?

Ici, nous avons un problème d'ajustement sérieux des modèles envisagés aux données. Le problème provient en fait d'une hypothèse implicite lorsque l'on utilise un modèle poissonien: l'équidispersion. Cela signifie que moyenne et variance sont égale. Cependant, le cas général est plutôt d'avoir des données surdispersées, c'est-à-dire des données dont la variance est supérieure à la moyenne. Cette surdispersion peut être le reflet de différent processus, par exemple une structure spatiale non prise en compte par les covariables, etc.

Ici, nous allons modifier le modèle pour ne plus supposer une vraisemblance poissonienne pour les données, mais une vraisemblance Negative Binomiale:

$$y_i^{\text{obs}} \sim \mathcal{NB}(\omega, \lambda_i)$$

où $\omega > 1$ est le paramètre de surdipersion. Il permet de prendre en compte une variance supérieure à la moyenne dans des données de comptages:

$\mathbb{V}(y) = \omega\,\mathbb{E}(y)$

Nous allons simuler quelques données avec $\mathtt{R}$ pour regarder la différence entre une distribution Poisson et une distribution Negative Binomiale.

In [ ]:
```
n <- 1e3 # nombre de données à simuler
lambda <- 5 # moyenne du processus
omega <- 3 # paramètre de surdispersion
ggplot(data = data.frame(x = c(rpois(n, lambda),
                               MASS::rnegbin(n, mu = lambda, theta = lambda/(ome
ga - 1))
                               ),
                         distribution = rep(c("Poisson", "NegBin"), each = n)
                         ),
       aes(x = x, group = distribution)
       ) +
geom_histogram(breaks = seq(0:50)) +
facet_wrap(~distribution, ncol = 1)
```

Negative Binomiale et Poisson

La fonction de densité de probabilité d'une variable aléatoire y qui suit une loi de Poisson d'intensité $\lambda\ (>0)$ est donnée par :

$$p(y|\lambda) = \frac{e^{\lambda}\lambda^{y}}{y!}$$

La moyenne de y est égale à la variance : $\mathbb{V}(y) = \mathbb{E}(y) = \lambda$

La fonction de densité de probabilité d'une variable aléatoire y qui suit une loi Negative Binomiale d'intensité $\lambda\ (>0)$ et de paramètre de surdispersion $\omega\ (>0)$ est donnée par :

$$p(y|\lambda,\omega) = \binom{y+\frac{\lambda}{\omega-1}-1}{y}(\frac{1}{\omega})^{\frac{\lambda}{\omega-1}}(1-\frac{1}{\omega})^{y}$$

La distribution Negative binomiale peut aussi être vue comme une généralisation de la Poisson avec une intensité qui est elle-même une variable aléatoire suivant une loi Gamma de paramètre de forme $\frac{\lambda}{\omega-1}$ et de paramètre de taux $\frac{1}{\omega-1}$. Cette caractérisation permet d'écrire facilement un modèle sous `Jags` ou `Bugs`.

Voir aussi https://www.johndcook.com/negative_binomial.pdf (https://www.johndcook.com/negative_binomial.pdf) pour les différentes paramétrisations possible.

Pour modifier la distribution dans le code `Jags` il va falloir faire quelques changements. En particulier, il faut spécifier un prior pour ω.

La paramètre de surdispersion est compris entre 1 et $+\infty$. On va ici mettre un prior uniforme entre 0 et 1 sur l'inverse du paramètre de surdispersion :

$$p(\frac{1}{\omega}) \propto 1$$

Il faut également modifier le calcul de la vraisemblance. La parametrisation la plus prisée par les écologistes utilise un paramètre $\phi=\lambda\omega-1$.

Nota:Ce modèle, plus complexe, nécessite plus de temps pour tourner

In [ ]:
```
model.jags <- '
 model{
  # Prior
  intercept ~ dnorm(0.0, hyperparam_inter)
  slope[1] ~ dnorm(0.0, hyperparam_slope[1])
  slope[2] ~ dnorm(0.0, hyperparam_slope[2])
  slope[3] ~ dnorm(0.0, hyperparam_slope[3])
  inv_overdispersion ~ dunif(0, 1)
  overdispersion <- 1/inv_overdispersion

  # Likelihood
  for (i in 1:n_obs) {
    lambda[i] <- exp(intercept + slope[1] * X1[i] + slope[2] * X2[i] + slope[3]
* X3[i])
    r[i] <- lambda[i] / (overdispersion - 1)
    Y[i] ~ dnegbin(inv_overdispersion, r[i])
  }
}
'

data.jags <- list(n_obs = nrow(obs),
                  Y = obs$nombreShift,
                  X1 = as.vector(scale(obs$DepthGebco)),
                  X2 = as.vector(scale(obs$Slope)),
                  X3 = as.vector(scale(obs$DistCot)),
                  hyperparam_inter = 1.0,
                  hyperparam_slope = 1/rep(log(2)/2, 3)^2
                  )
# Inits function
inits.jags <- function(idchain, glm) {
  list(intercept = rnorm(1, glm$coefficients[1], sqrt(10 * diag(vcov(glm)))[1]),
       slope = rnorm(3, glm$coefficients[2:4], sqrt(10 * diag(vcov(glm)))[2:4]),
       inv_overdispersion = runif(1, 0, 1)
  )
}

params.jags <- c("intercept", "slope", "overdispersion")

# Start Gibbs sampler
temp <- jags(data = data.jags,
             inits = lapply(1:nc, inits.jags, glm = dauphin2.glm),
             param = params.jags,
             model.file = textConnection(model.jags),
             n.chains = nc,
             n.burnin = nb,
             n.iter = ni,
             n.thin = nt
             )

out3 = temp$BUGSoutput
# Inferences
# After Burn in period let's work inference assuming ergodic regime is reached
print(out3$summary, dig = 2)
```

On peut vérifier graphiquement la convergence du modèle.

In [ ]:
```
library(coda)
gelman.diag(x=out3, confidence = 0.95, transform=FALSE, autoburnin=TRUE)
gelman.plot(x=out3)
```

Et la trace des paramètres du modèle :

In [ ]:
```
traceplot(out3)
```

Calcul du critère d'information

Ce résultat est à comparer au modèle poisson sans surdispersion. Se souvenir que pour ces critères : *the lower, the better*

In [ ]:
```
y_rep_3 <- log_lik3 <- array(NA, dim = c(nc * (ni - nb)/nt, nrow(obs)))

for(i in 1:nrow(obs)) {
  overdispersion <- drop(out3$sims.list$overdispersion)
  linpred <- out3$sims.list$intercept +
    out3$sims.list$slope[, 1] * data.jags$X1[i] +
    out3$sims.list$slope[, 2] * data.jags$X2[i] +
    out3$sims.list$slope[, 3] * data.jags$X3[i]
  log_lik3[, i] <- dnbinom(data.jags$Y[i],
                           prob = 1 / overdispersion,
                           size = exp(linpred) / (overdispersion - 1),
                           log = TRUE)
  y_rep_3[, i] <- rnbinom(nrow(y_rep_3),
                          prob = 1 / overdispersion,
                          size = exp(linpred) / (overdispersion - 1)
                          ) + 1
}; rm(i, linpred, overdispersion)

loo::waic(log_lik3)
loo::loo(log_lik3)
```

Que dire de l'ajustement global du modèle aux données?

In [ ]:
```
rootogram_model_3 <- rootogram(countdata = obs$nombre, y_rep = y_rep_3)
rootogram_model_3 +
  coord_cartesian(xlim = c(1, 50))

plot_relationships(data_df = obs, jagsfit = out3, cov_name = c("DepthGebco", "Slope", "DistCot"))
```

Une première satisfaction

Comme nous le disait le WAIC, l'ajustement est bien meilleur et les zones de crédibilité nous semblent plus réalistes.

- exercice: Est-ce que cela se reflète dans le DIC?
- exercice: tracer les lois a posteriori des inconnues

Qu'en est il du paramètres de surdispersion?

In [ ]:
```
hist(out3$sims.list$overdispersion, las = 1, bty = 'n',
     xlab = quote(omega), ylab = "frequency", main = "Posterior")
```

On peut comparer cette distribution a posteriori à la distribution a priori :

In [ ]:
```
ggplot(data = data.frame(x = c(1 / runif(length(out3$sims.list$overdispersion),
0.0, 1.0),
                                out3$sims.list$overdispersion
                                ),
                         distribution = rep(c("prior", "posterior"), each = leng
th(out3$sims.list$overdispersion))
                         ),
       aes(x = x, group = distribution)
       ) +
geom_histogram(breaks = c(0:100)) +
facet_wrap(~distribution, ncol = 1) +
coord_cartesian(xlim = c(0, 30))
```

Modéliser la surdispersion de manière plus explicite

Il existe plusieurs processus écologiques susceptibles de générer de la surdispersion dans les données de comptage ici, par exemple des structures d'aggrégation des proies dont dépendent les dauphins et qui ne seraient que partiellement prises en compte par les covariables, $etc.$ On a jusqu'à présent ignorer certaines structures dans les données, notamment le fait que celles-ci correspondent à des années différentes. Or, il est possible que certaines années, pour des raisons que l'on comprend mal, on ait vu plus ou moins de dauphins. En d'autres termes, il est possible qu'il y ait une dépendence temporelle dans les données qui n'a pas été prise en compte.

On va donc tenir compte de cette effet année, et le modéliser en incluant dans le modèle l'année comme effet aléatoire, càd on va supposer que les années sont échangeables : il n'est pas possible $a\ priori$ de classer les années en fonction de l'abondance d'animaux. On va supposer une distribution commune aux effets années, et par convenance on va supposer une distribution normale:

$$\forall\ t,\ \mathrm{year}_t \sim \mathcal{N}(0, \sigma_{\mathrm{year}})$$

Le paramètre $\sigma_{\mathrm{year}}\ (> 0)$ quantifie la variabilité inter-annuelle. On suppose ici que l'effet moyen d'une année prise au hasard est centrée sur 0, mais peut s'écarter de 0 en fonction de ce qui est attendue sous une loi normale d'écart-type σ_{year}. En d'autres termes, en plus d'estimer les paramètres year_t, nous allons estimer l'hyperparamètre σ_{year} qui va gouverner la distribution normale. Il faut donc spécifier des priors pour cet hyperparamètre.

In [ ]:
```
model.jags <- '
 model{
  # Prior
  intercept ~ dnorm(0.0, hyperparam_inter)
  slope[1] ~ dnorm(0.0, hyperparam_slope[1])
  slope[2] ~ dnorm(0.0, hyperparam_slope[2])
  slope[3] ~ dnorm(0.0, hyperparam_slope[3])
  inv_overdispersion ~ dunif(0, 1)
  overdispersion <- 1/inv_overdispersion
  sigma_year ~ dnorm(0.0, 1.0)T(0.0,)
  tau_alpha <- pow(sigma_year, -2)
  for(j in 1:n_year) {
    alpha[j] ~ dnorm(intercept, tau_alpha)
  }

  # Likelihood
  for (i in 1:n_obs) {
    lambda[i] <- exp(alpha[YEAR[i]] + slope[1] * X1[i] + slope[2] * X2[i] + slop
e[3] * X3[i])
    r[i] <- lambda[i] / (overdispersion - 1)
    Y[i] ~ dnegbin(inv_overdispersion, r[i])
  }
 }
'

data.jags <- list(n_obs = nrow(obs),
                  n_year = length(unique(obs$an)),
                  Y = obs$nombreShift,
                  X1 = as.vector(scale(obs$DepthGebco)),
                  X2 = as.vector(scale(obs$Slope)),
                  X3 = as.vector(scale(obs$DistCot)),
                  YEAR = obs$an - min(obs$an) + 1,
                  hyperparam_inter = 1.0,
                  hyperparam_slope = 1/rep(log(2)/2, 3)^2
                  )

# Inits function
inits.jags <- function(idchain, glm) {
  list(intercept = rnorm(1, glm$coefficients[1], sqrt(10 * diag(vcov(glm)))[1]),
       slope = rnorm(3, glm$coefficients[2:4], sqrt(10 * diag(vcov(glm)))[2:4]),
       inv_overdispersion = runif(1, 0, 1),
       sigma_year = abs(rnorm(1)),
       alpha = rnorm(length(unique(obs$an)))
       )
}

params.jags <- c("intercept", "slope", "overdispersion", "alpha", "sigma_year")

# Start Gibbs sampler
temp <- jags(data = data.jags,
             inits = lapply(1:nc, inits.jags, glm = dauphin2.glm),
             param = params.jags,
             model.file = textConnection(model.jags),
             n.chains = nc,
             n.burnin = nb,
             n.iter = ni,
             n.thin = nt
             )

out4 = temp$BUGSoutput
# Inferences
# After Burn in period let's work inference assuming ergodic regime is reached
print(out4$summary, dig = 2)
```

On peut également vérifier la convergence des différents paramètres ...

In [ ]:
```
library(coda)
gelman.diag(x=out4, confidence = 0.95, transform=FALSE, autoburnin=TRUE)
gelman.plot(x=out4)
```

et leur trace

In [ ]:
```
traceplot(out4)
```

WAIC

On peut calculer le WAIC, du point de vue de ce critère le model est il meilleur que les précédents?

In [ ]:
```
### calcul du WAIC
## besoin de calculer la log vraisemblance sous le modele
y_rep_4 <- log_lik4 <- array(NA, dim = c(nc * (ni - nb)/nt, nrow(obs)))

for(i in 1:nrow(obs)) {
  overdispersion <- drop(out4$sims.list$overdispersion)
  linpred <- out4$sims.list$slope[, 1] * data.jags$X1[i] +
    out4$sims.list$slope[, 2] * data.jags$X2[i] +
    out4$sims.list$slope[, 3] * data.jags$X3[i] +
    out4$sims.list$alpha[, data.jags$YEAR[i]]
  log_lik4[, i] <- dnbinom(data.jags$Y[i],
                           prob = 1 / overdispersion,
                           size = exp(linpred) / (overdispersion - 1),
                           log = TRUE
                           )
  y_rep_4[, i] <- rnbinom(nrow(y_rep_4),
                          prob = 1 / overdispersion,
                          size = exp(linpred) / (overdispersion - 1)
                          ) + 1
}; rm(i, linpred, overdispersion)

loo::waic(log_lik4)
loo::loo(log_lik4)
```

L'histogramme des donnees se compare-t-il à l'histogramme attendu sous le modele?

In [ ]:
```
## regarder l'histogramme des donnees et le comparer a l'histogramme attendu sou
s le modele
rootogram_model_4 <- rootogram(countdata = obs$nombre, y_rep = y_rep_4)
rootogram_model_4 +
  coord_cartesian(xlim = c(1, 50))
```

Un ajustement des effets avec des incertitudes plus réalistes

In [ ]:
```
### un peu mieux que le modele precedent...
plot_relationships(data_df = obs, jagsfit = out4, cov_name = c("DepthGebco", "Sl
ope", "DistCot"))
```

On va regarder ce que ce modèle avec effets aléatoires change au niveau de l'estimation du paramètre de surdispersion:

In [ ]:
```
ggplot(data = data.frame(x = c(out3$sims.list$overdispersion,
                               out4$sims.list$overdispersion
                               ),
                         distribution = rep(c("Sans effet année", "Avec effet al
éatoire année"), each=length(out3$sims.list$overdispersion) )
                         ),
       aes(x = x, group = distribution)
       ) +
geom_histogram(breaks = seq(10, 25, 0.1)) +
facet_wrap(~distribution, ncol = 1) +
coord_cartesian(xlim = c(12, 22))+labs(x="paramètre d'overdispersion", y="répart
ition")
```

Une partie de la surdispersion est absorbée par l'effet année mais il reste une grande part qui n'est pas expliquée :

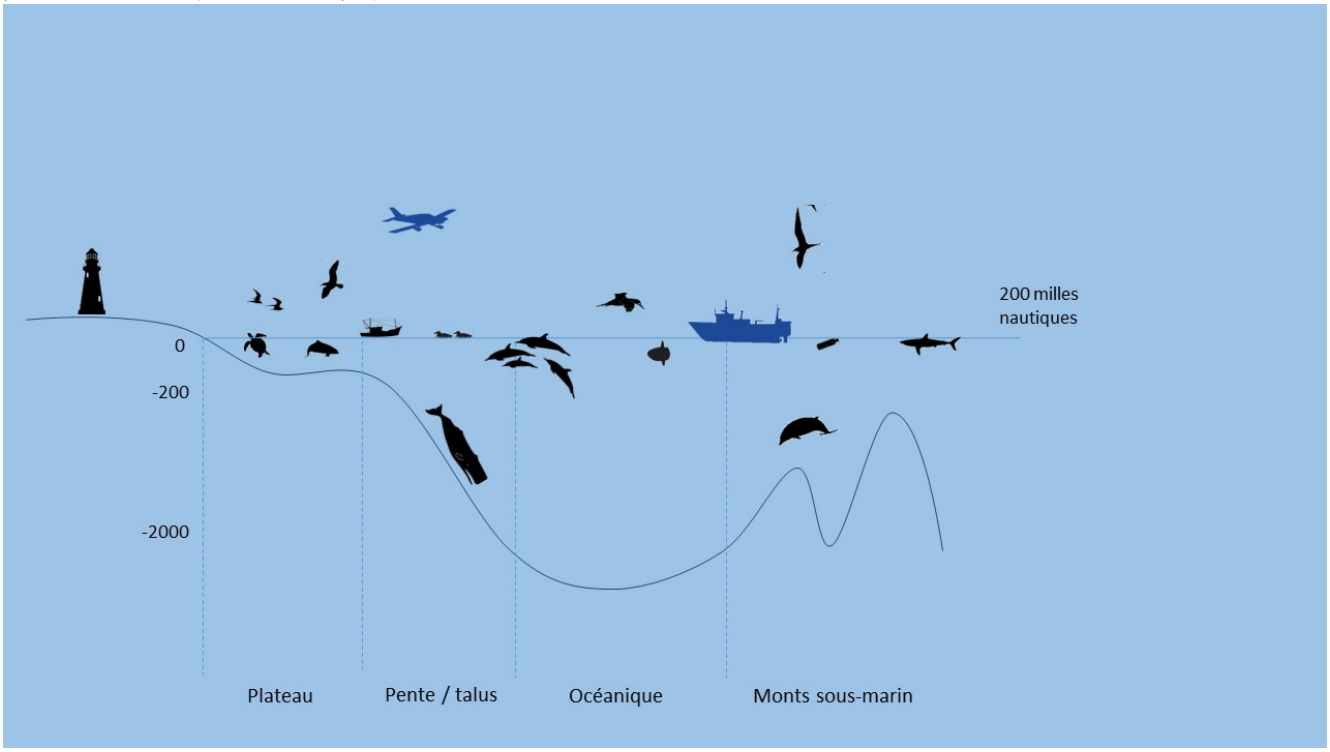
In [ ]:
```
1 - round(mean(out4$sims.list$overdispersion /
               out3$sims.list$overdispersion
               ), 2
          )
```

Une dernière option ici est de tenir compte d'un effet spatial avec un modèle de type Conditional Auto-Regressive (CAR). Ce modèle est facile à ajuster avec **Stan** (https://mc-stan.org/users/documentation/case-studies/icar_stan.html (https://mc-stan.org/users/documentation/case-studies/icar_stan.html)). Il faut cependant utiliser un nouveau bloc pour y définir une nouvelle distribution : la distribution CAR pour l'effet spatial. L'avantage de **Stan** ici est de pouvoir utiliser une paramétrisation qui va éviter des calculs coûteux. Ce modèle est ajustable dans un script supplémentaire mais ne sera pas couvert dans le TD.

On va donc ici considérer que notre meilleur modèle est le modèle surdispersé avec effet aléatoire de l'année . Les relations estimées avec l'environnement nous donnent la crédibilité de ces différentes variables environnementales sur la réponse conditionnellement à cette construction.

Conclusion

Nous avons ajuster une série de modèle à la complexité croissante pour estimer le nombre de dauphins dans le Golfe de Gascogne en fonction de diverses covariables environnementales. On peut voir ici qu'il y a plus de dauphins dans des zones de grandes profondeurs, loin des côtes et aussi des zones où la bathymétrie changent très rapidement sur de petites distances (talus océanique).



Néanmoins, le modèle sélectionné reste mauvais car il ignore un aspect important des données : les observateurs qui font les relevés sur le terrain ont tendance à arrondir les chiffres au delà d'une certaine abondance d'animaux, et cela induit des pics à des multiples de 5 ou 10. Il faudrait donc tenir compte du comportement des observateurs car celui-ci induit une source d'erreur de mesure qui n'est pas anodine, et qui contribue à générer de la surdispersion.