Verificación e interpretación de modelos

Table of contents

0.1 Pasos comunes en una instancia de modelado 1
0.2 Verificación de modelos 1
0.2.1 Distribución predictiva posterior 1
0.3 Interpretación de modelos 24
0.3.1 Ejemplo con más predictoras 29
0.4 Simulaciones previas 43
0.4.1 Simulaciones previas 1: incendios 44

0.1 Pasos comunes en una instancia de modelado

1. Formulación de modelos cuantitativos a partir de conocimiento previo (definición de previas).

2. Toma de datos (y pre-procesamiento).
3. Ajuste de modelos a datos (estimación de la posterior).
4. Verificación del algoritmo de estimación (e.g., convergencia del MCMC).
5. Verificación del ajuste del modelo a los datos (e.g., análisis de residuos).
6. Interpretación del modelo (gráficos, predicciones, etc.).

0.2 Verificación de modelos

Lo mínimo que podemos pedirle a un modelo es que represente de manera razonable los datos con los que fue estimado.

En el caso de modelos estadísticos, que son generativos, lo deseable es que puedan simular datos similares a los observados.

0.2.1 Distribución predictiva posterior

Es la distribución de nuevos valores de la variable respuesta, a los que llamaremos $\tilde{y}$, condicionando en que observamos y.

$$p(\tilde{y} \mid y) = \int p(\tilde{y} \mid \theta)\ p(\theta \mid y)\ \mathrm{d}\theta.$$

En vez de evaluarla analíticamente, podemos simular de esta distribución utilizando las muestras de θ que obtuvimos con MCMC.

Para cada observación y_i (con $i \in \{1, ..., N\}$), y con cada muestra de la posterior θ_s (con $s \in \{1, ..., S\}$), se simula al menos una observación de la variable respuesta ($\tilde{y}_i^s$).

Si queremos utilizar esta distribución para verificar el modelo, tenemos que simular nuevos datos utilizando los mismos valores de las predictoras (x) asociados a cada observación (x_i).

Ejemplo con el modelo de la cantidad de incendios por año

```
# Cargamos datos
datos <- read.csv(here::here("datos", "barbera_data_fire_total_climate.csv"))

# Compilamos el modelo
model <- cmdstan_model(here::here("modelos", "nfuegos.stan"))
```

```
Warning in readLines(stan_file): incomplete final line found on
'/home/ivan/Insync/Curso Modelos y Datos - CRUB -
Gure25/curso-bayes-25/modelos/nfuegos.stan'
```

```
# Creamos una lista nombrada para pasarle los datos. Los nombres de cada
# elemento de la lista tienen que ser exactamente los que definimos en
# la sección data {}
stan_data <- list(
  N = nrow(datos),
  y = datos$fires,
  x = datos$fwi
)

# Y muestreamos la posterior
fit_mcmc <- model$sample(
  data = stan_data,
  chains = 4,
  parallel_chains = 4,
  iter_warmup = 1000,
  iter_sampling = 1000
)
```

```
Running MCMC with 4 parallel chains...
```

```
Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
```

```
Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1 finished in 0.1 seconds.
Chain 2 finished in 0.1 seconds.
Chain 3 finished in 0.1 seconds.
```

```
Chain 4 finished in 0.1 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.1 seconds.
Total execution time: 0.3 seconds.
```

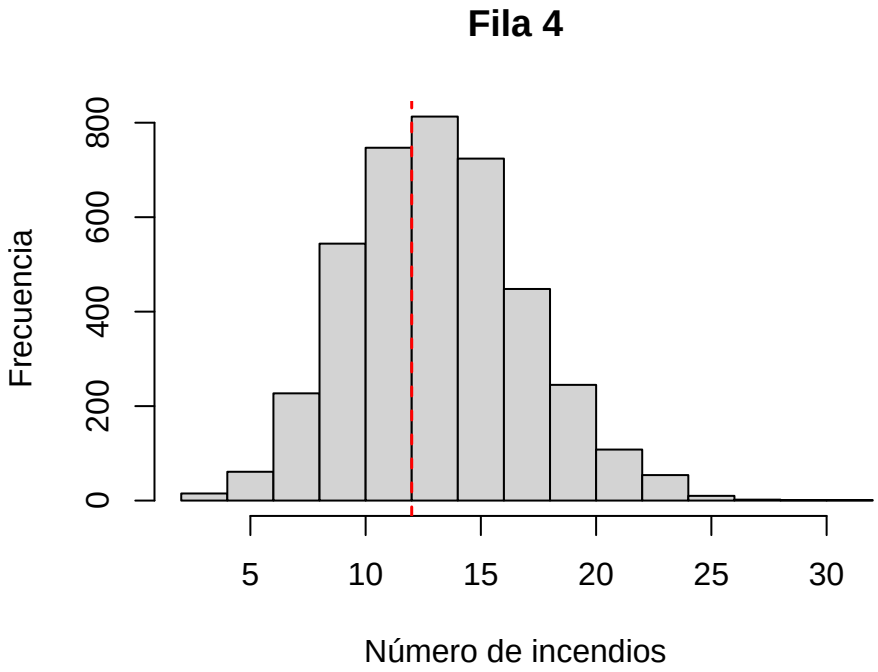
```r
# Verificamos el HMC
fit_mcmc$cmdstan_diagnose()
```

```
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

Rank-normalized split R-hat values satisfactory for all parameters.

Processing complete, no problems detected.
```

```r
# Extraemos las muestras de alpha y beta, en formato data.frame
d <- fit_mcmc$draws(variables = c("alpha", "beta"), format = "draws_df")
colnames(d) <- c("a", "b")

# Ahora podemos simular desde la distribución predictiva posterior.
# Para cada observación simularemos S = 4000 muestras de la predictiva posterior.
# Guardaremos las simulaciones en una matriz de N * S:

S <- nrow(d)     # número de muestras de la posterior
N <- nrow(datos) # número de observaciones

ysim <- matrix(NA, N, S)

# Simulamos
for (s in 1:S) {
  # calculamos lambda para la muestra "s"
  lambda <- exp(d$a[s] + d$b[s] * datos$fwi)
  # es un vector de largo N, porque datos$fwi también lo es.

  # llenamos una columna entera en ysim, que equivale a un set de datos simulado
  ysim[, s] <- rpois(N, lambda)
}
```

```r
# Ahora cada fila tiene muestras de la distribución predictiva posterior de y
# para el valor correspondiente de fwi
```

```r
fila <- sample(1:N, 1)

rr <- c(datos$fires[fila], ysim[fila, ]) |> range()

hist(ysim[fila, ], xlim = rr, main = paste("Fila", fila),
     xlab = "Número de incendios", ylab = "Frecuencia")
abline(v = datos$fires[fila], lty = 2, col = "red", lwd = 1.5)
```



Una vez obtenida la (muestra de la) distribución predictiva posterior, necesitamos evaluar si nuestros datos parecen salidos de ahí.

Para ello utilizamos la Transformación Integral de Probabilidad (*PIT*, de *Probability Integral Transform*):

sea Y una variable aleatoria continua y F_Y su función de probabilidad acumulada, la variable aleatoria Z, definida como $Z = F_Y(Y)$, sigue una distribución uniforme entre 0 y 1.

Esto implica que si nuestras observaciones se comportan como muestras de la distribución predictiva posterior, sus valores de probabilidad acumulada *en conjunto* siguen una distribución uniforme entre 0 y 1.

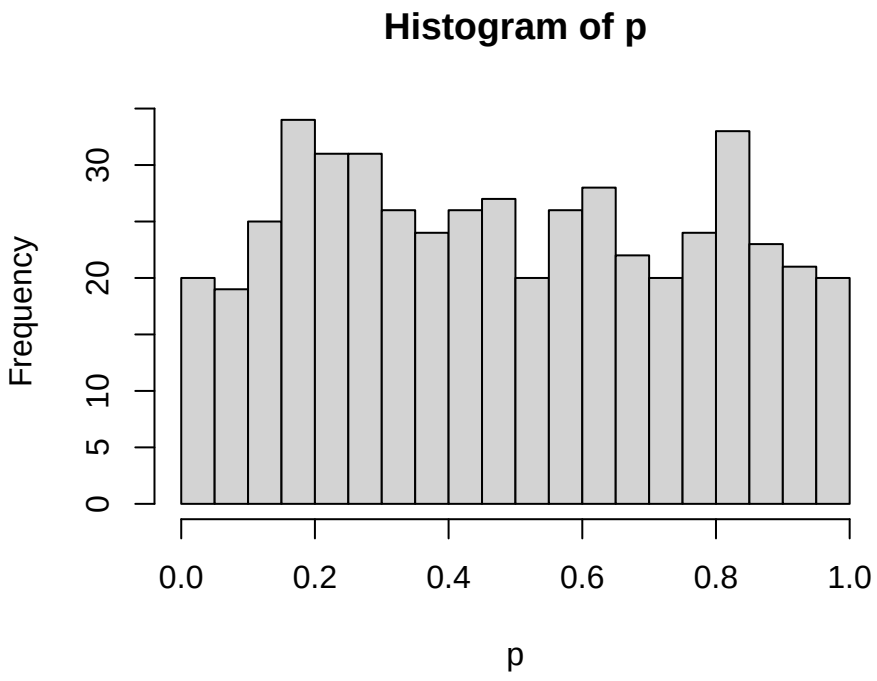
A diferencia de los distintos tipos de residuos que suelen calcularse, esto aplica sin importar qué características tenga la distribución predictiva posterior.

Sin embargo, esto no se cumple si Y es discreta, y en ese caso hay que aleatorizar los valores de probabilidad acumulada (Z) para que se distribuyan de forma uniforme.

Estos valores de probabilidad acumulada (aleatorizados en caso de variables discretas) son calculados por el paquete `DHARMa` (*DHARMa residuals*), pero también aparecen en `INLA`, `loo` y `bayesplot` bajo el nombre de *PIT values*.

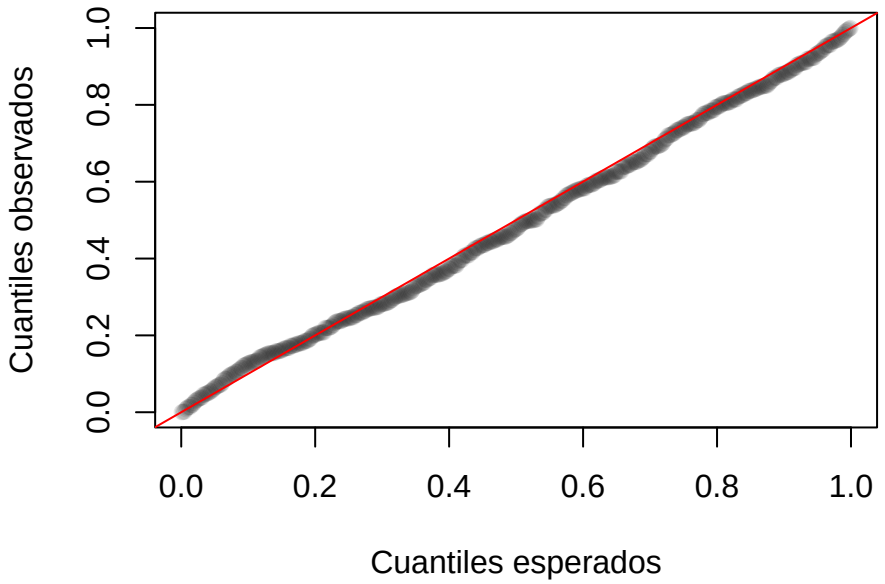
Ejemplo en `R`

```r
N <- 500
y <- rnorm(N)
p <- pnorm(y)

hist(p, breaks = 30)
```

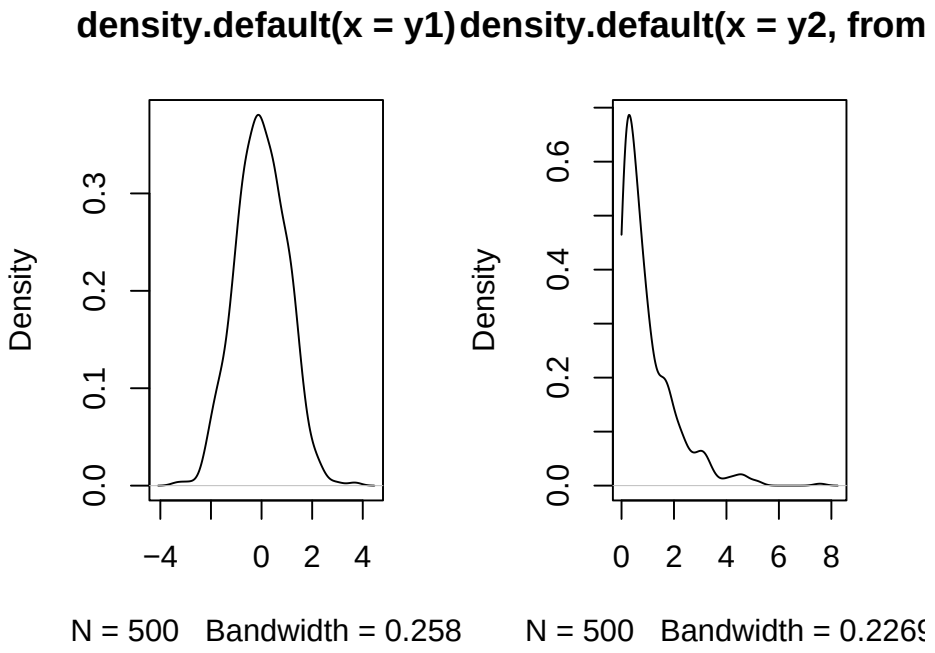



```r
# N estadísticos de orden de una uniforme:
q_unif <- ppoints(N)
```

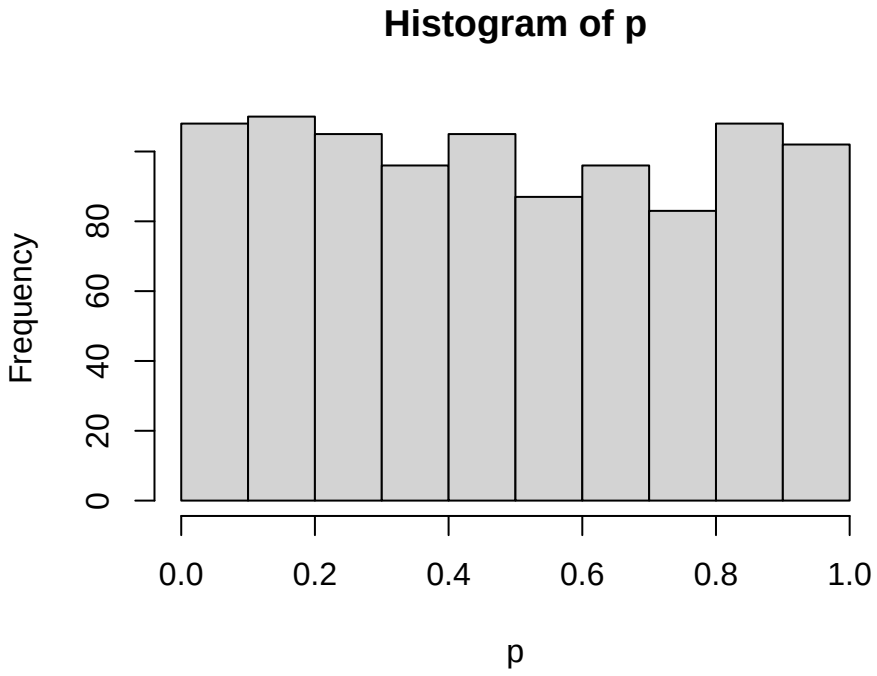
```r
plot(sort(p) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados")
abline(0, 1, col = "red")
```



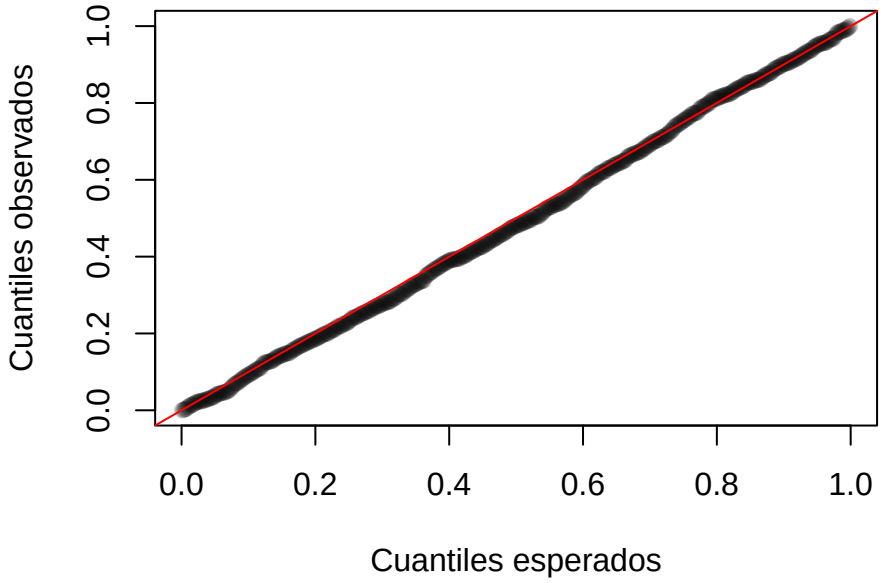

```r
# ¿Y si las muestras son una mezcla proveniente de distintas distribuciones?
y1 <- rnorm(N)
y2 <- rgamma(N, 1)

par(mfrow = c(1, 2))
plot(density(y1))
plot(density(y2, from = 0))
```




```r
par(mfrow = c(1, 1))

p1 <- pnorm(y1)
p2 <- pgamma(y2, 1)

p <- c(p1, p2)

hist(p)
```

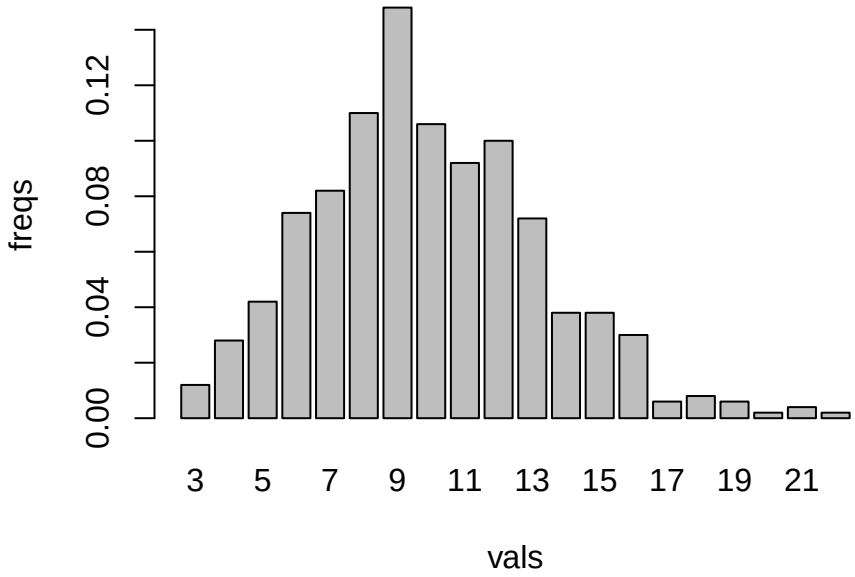



```r
q_unif <- ppoints(N * 2)
plot(sort(p) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados")
abline(0, 1, col = "red")
```

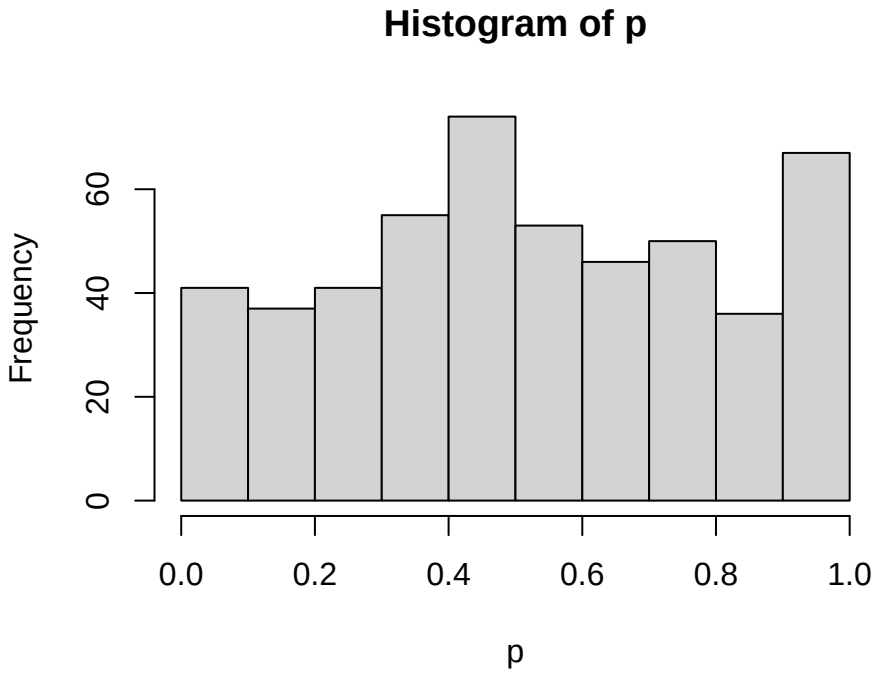


Ejemplo en R con distribución discreta

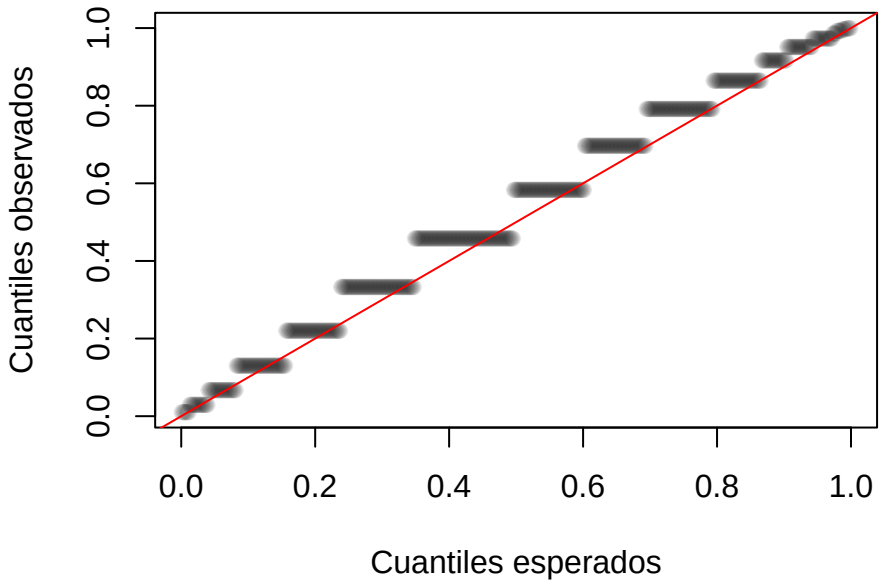
```
y <- rpois(N, 10)
freqs <- table(y) / N
vals <- as.numeric(names(freqs))
barplot(freqs ~ vals)
```




```
p <- ppois(y, lambda = 10)

hist(p)
```



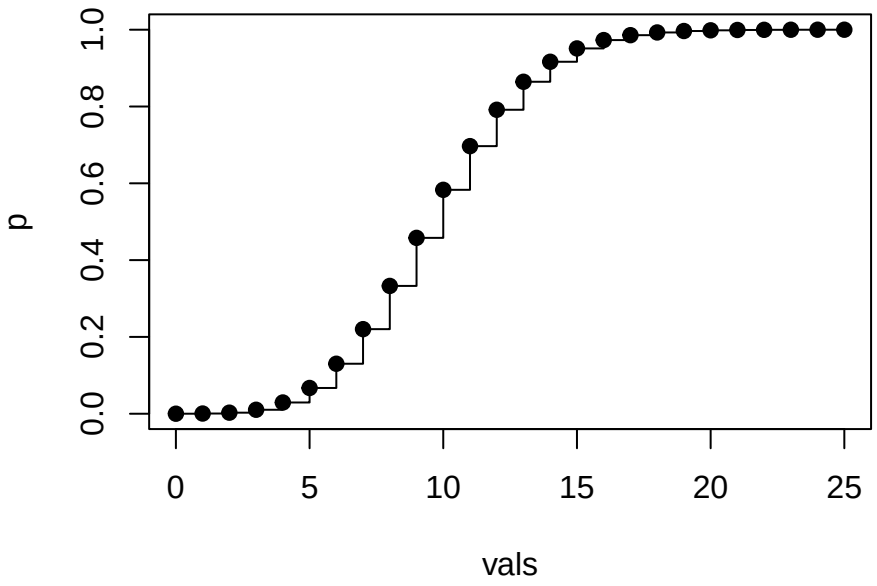

```r
q_unif <- ppoints(N)
plot(sort(p) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados")
abline(0, 1, col = "red")
```



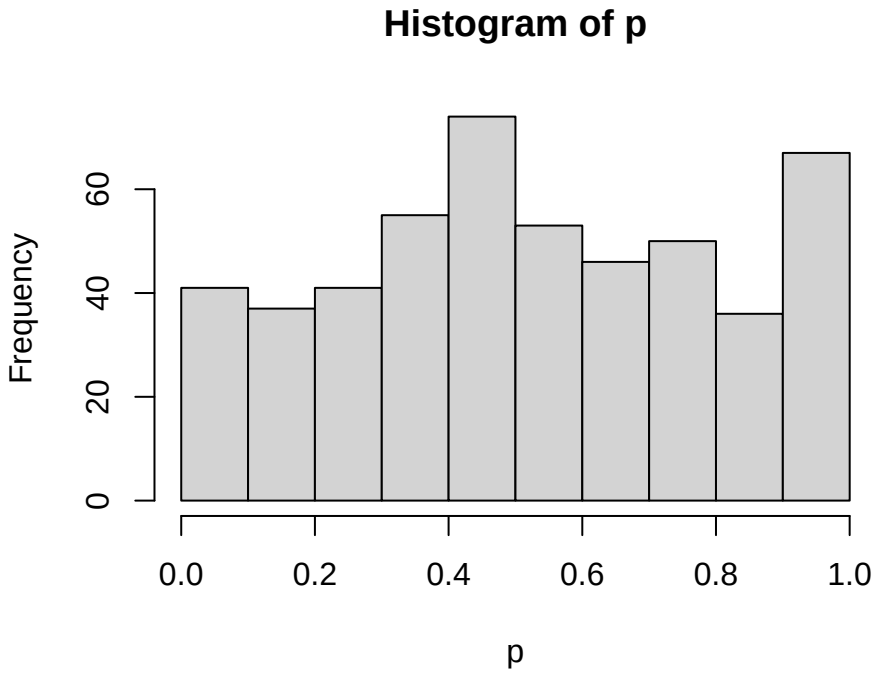
La función de probabilidad acumulada de una discreta es una función escalon-

ada:

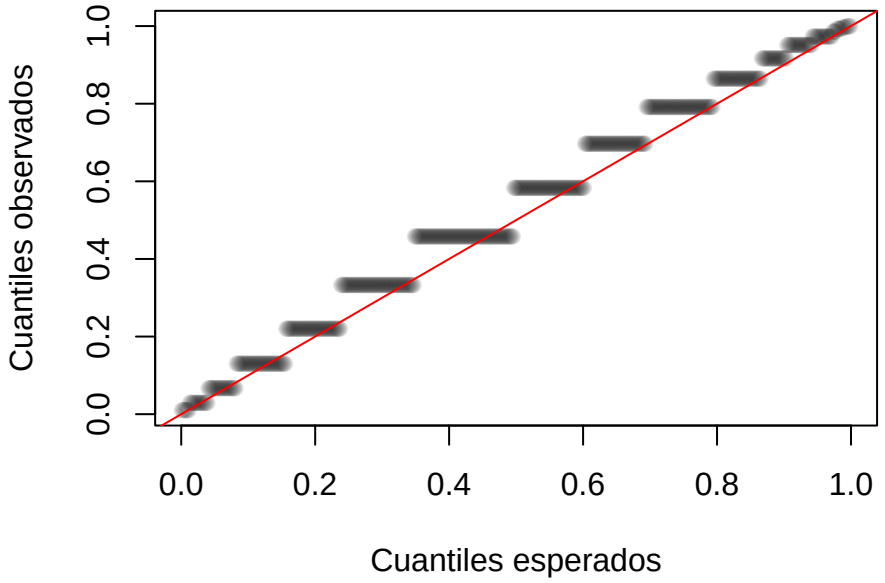
```
vals <- 0:25
p <- ppois(vals, 10)
plot(p ~ vals, type = "s")
points(p ~ vals, pch = 19)
```




```
p <- ppois(y, lambda = 10)

hist(p)
```




```r
q_unif <- ppoints(N)
plot(sort(p) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados")
abline(0, 1, col = "red")
```



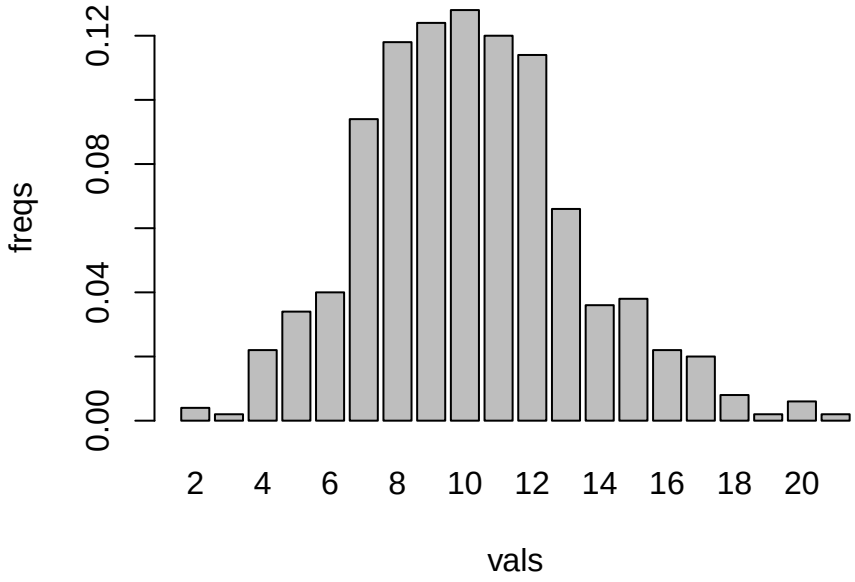
Para transformar estos valores de probabilidad acumulada en una variable con distribución uniforme, construimos nuestro valor Z (PIT) de la siguiente manera:

$$Z_i \sim \text{uniforme}(F_Y(y_i - 1), F_Y(y_i)).$$

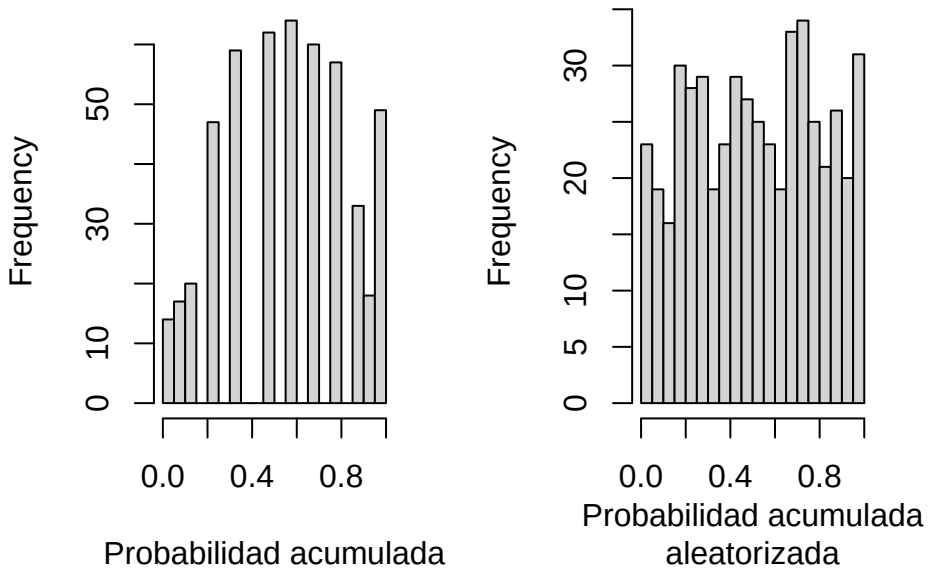
Es decir, en vez de quedarnos con la probabilidad acumulada ($F_Y(y_i)$), tomamos una muestra de una distribución uniforme limitada entre ese valor (por arriba) y la probabilidad acumulada de $y_i - 1$ ($F_Y(y_i - 1)$).

PIT para variable discreta

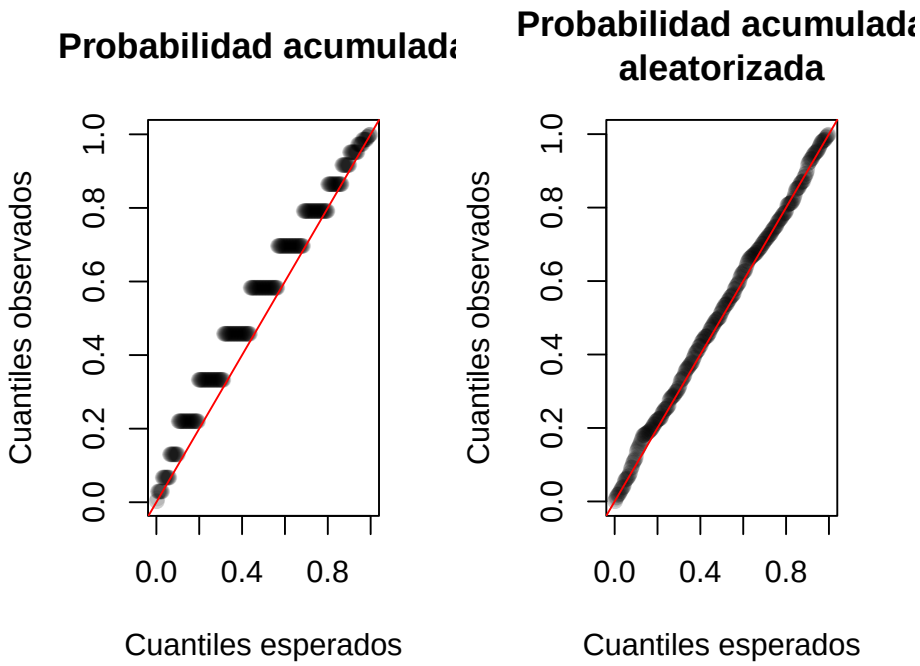
```
y <- rpois(N, 10)
freqs <- table(y) / N
vals <- as.numeric(names(freqs))
barplot(freqs ~ vals)
```




```
p <- ppois(y, lambda = 10) # Probabilidad acumulada en y
pl1 <- ppois(y-1, lambda = 10) # Probabilidad acumulada en y-1
pit <- runif(N, pl1, p)

par(mfrow = c(1, 2))
hist(p, breaks = 20, main = NA, xlab = "Probabilidad acumulada")
hist(pit, breaks = 20, main = NA, xlab = "Probabilidad acumulada\naleatorizada")
```




```r
par(mfrow = c(1, 1))

q_unif <- ppoints(N)

par(mfrow = c(1, 2))
plot(sort(p) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados",
     main = "Probabilidad acumulada")
abline(0, 1, col = "red")

plot(sort(pit) ~ q_unif, pch = 19, col = rgb(0, 0, 0, 0.1),
     ylab = "Cuantiles observados", xlab = "Cuantiles esperados",
     main = "Probabilidad acumulada\naleatorizada")
abline(0, 1, col = "red")
```




```
par(mfrow = c(1, 1))
```

Cuando evaluamos ajustes bayesianos, la distribución estimada por el modelo generalmente varía entre observaciones (sigue la misma estructura, e.g., poisson, pero tiene otros parámetros). En general aproximamos la acumulada mediante la función de probabilidad acumulada empírica (en `R`, la función `ecdf`).

```
N <- nrow(datos)
fila <- sample(1:N, 1)

# Obtenemos la función de distribución acumulada empírica de la distribución
# predictiva posterior para la fila elegida
ecdf_post <- ecdf(ysim[fila, ])

# al evaluarla, en el valor observado nos devuelve la probabilidad acumulada
(prob <- ecdf_post(datos$fires[fila]))
```

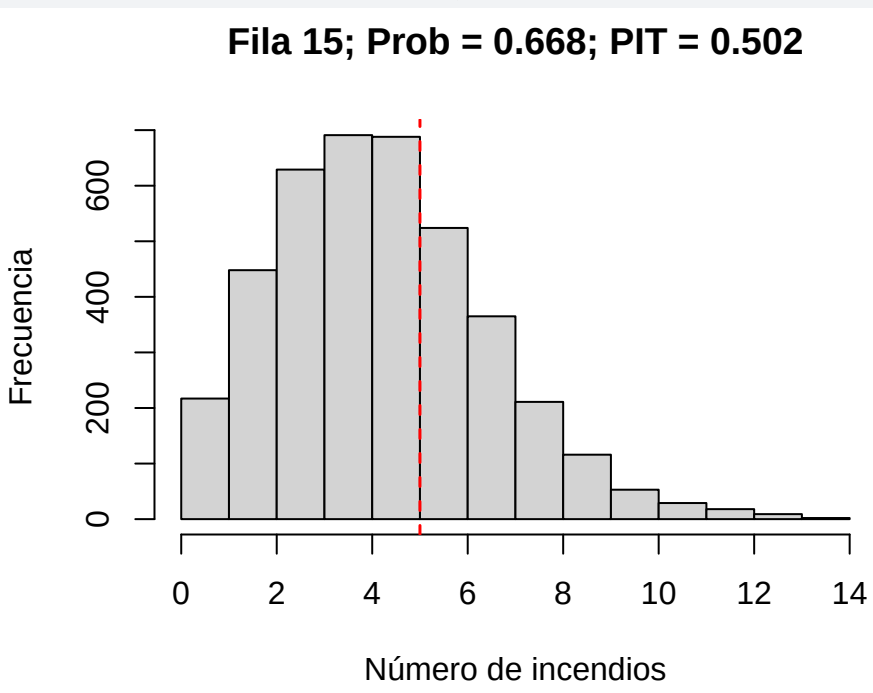
```
[1] 0.66825
```

```
# Calculamos el PIT:
(pit <- runif(
  1,
  min = ecdf_post(datos$fires[fila] - 1),
  max = ecdf_post(datos$fires[fila])
```

```
))
```

```
[1] 0.5022258
```

```
rr <- c(datos$fires[fila], ysim[fila, ]) |> range()

# Título
tt <- paste(
  "Fila ", fila, "; Prob = ",
  round(prob, 3),
  "; PIT = ",
  round(pit, 3),
  sep = ""
)
hist(ysim[fila, ], xlim = rr, main = tt,
     xlab = "Número de incendios", ylab = "Frecuencia")
abline(v = datos$fires[fila], lty = 2, col = "red", lwd = 1.5)
```



Pero `DHARMa` calcula los PIT por nosotros; sólo necesitamos darle una matriz de datos simulados.

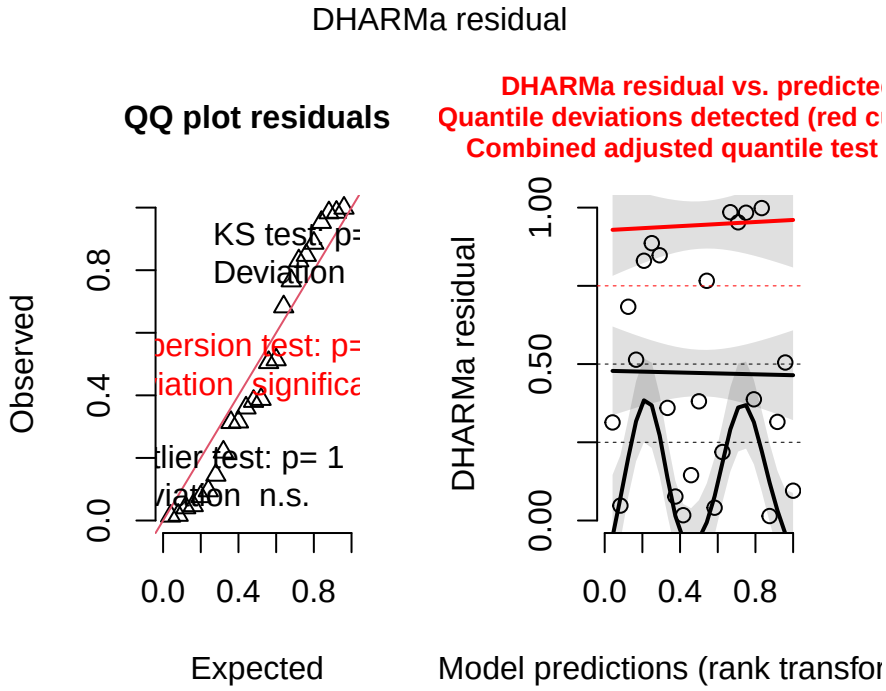
```
res <- createDHARMa(
  simulatedResponse = ysim,
  observedResponse = datos$fires,
```

```
  integerResponse = TRUE # se da cuenta solo, pero por las dudas
)
```

```
No fitted predicted response provided, using the mean of the simulations
```

```
plot(res)
```

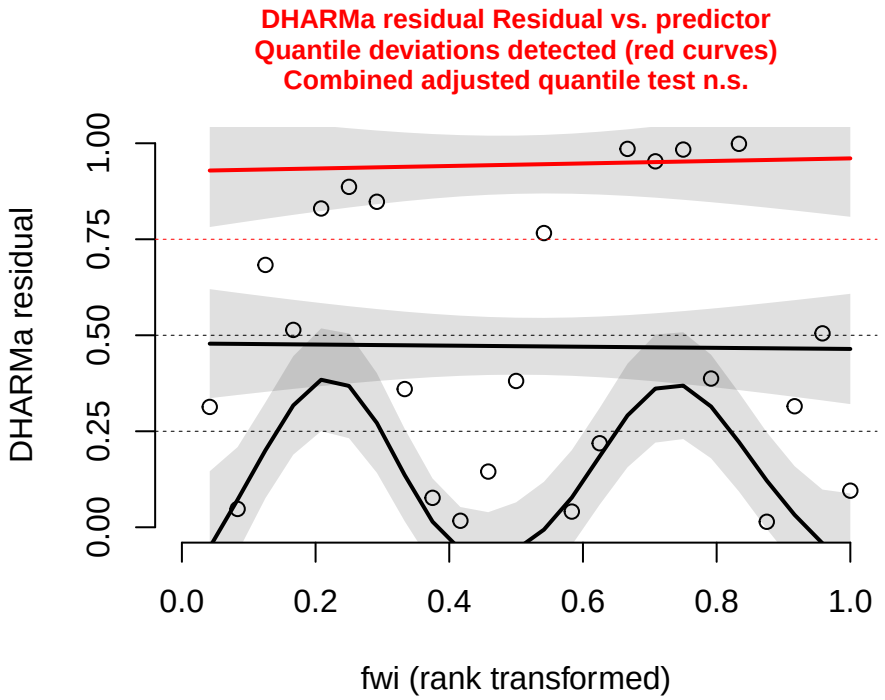
```
qu = 0.25, log(sigma) = -3.383699 : outer Newton did not converge fully.
```




```
hist(res$scaledResiduals)
```


```
# Podemos ver si hay alguna tendencia en función de una predictora
plotResiduals(res, form = datos$fwi)
```

```
qu = 0.25, log(sigma) = -3.383699 : outer Newton did not converge fully.
```

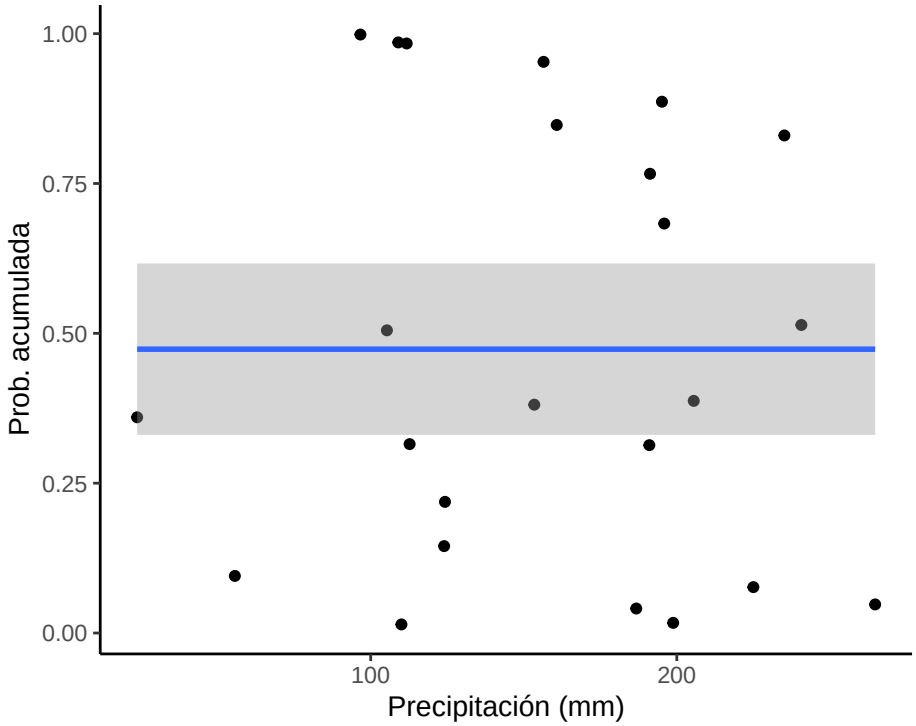


El modelo admite menor variabilidad que la encontrada en los datos.

Este modelo debe ser mejorado, pero podemos usar los residuos para ver si nos falta agregar alguna variable importante:

```
# Agregamos los PIT calculados por DHARMa a la tabla de datos
datos$pit <- res$scaledResiduals

ggplot(datos, aes(pp, pit)) +
  geom_point(size = 1.5) +
  labs(y = "Prob. acumulada",  # o "residuos dharma"
       x = "Precipitación (mm)") +
  geom_smooth(method = "gam")
```

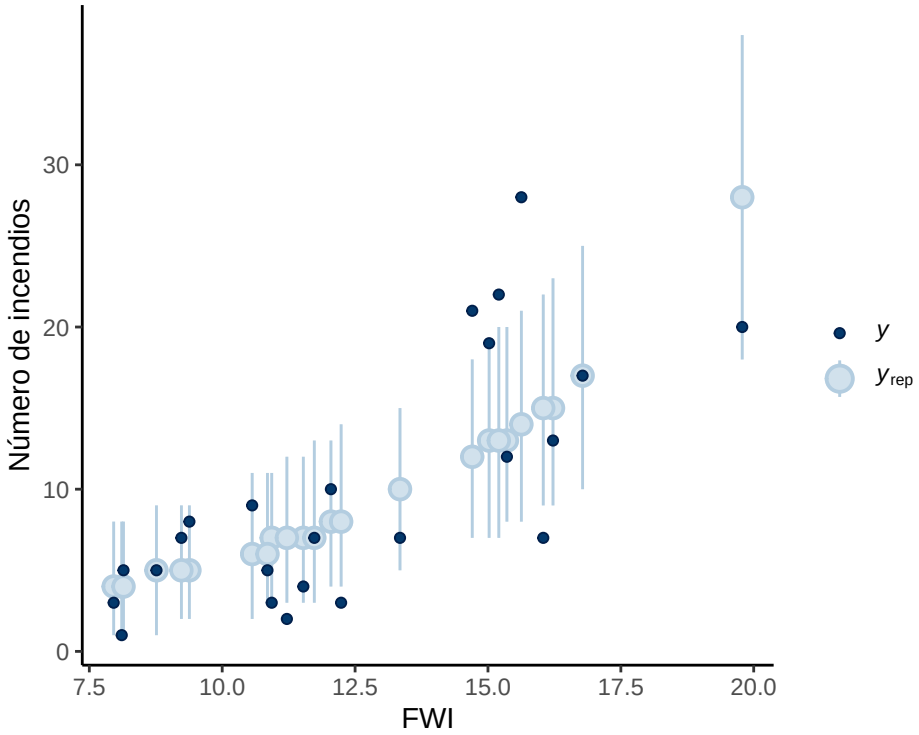
```
`geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



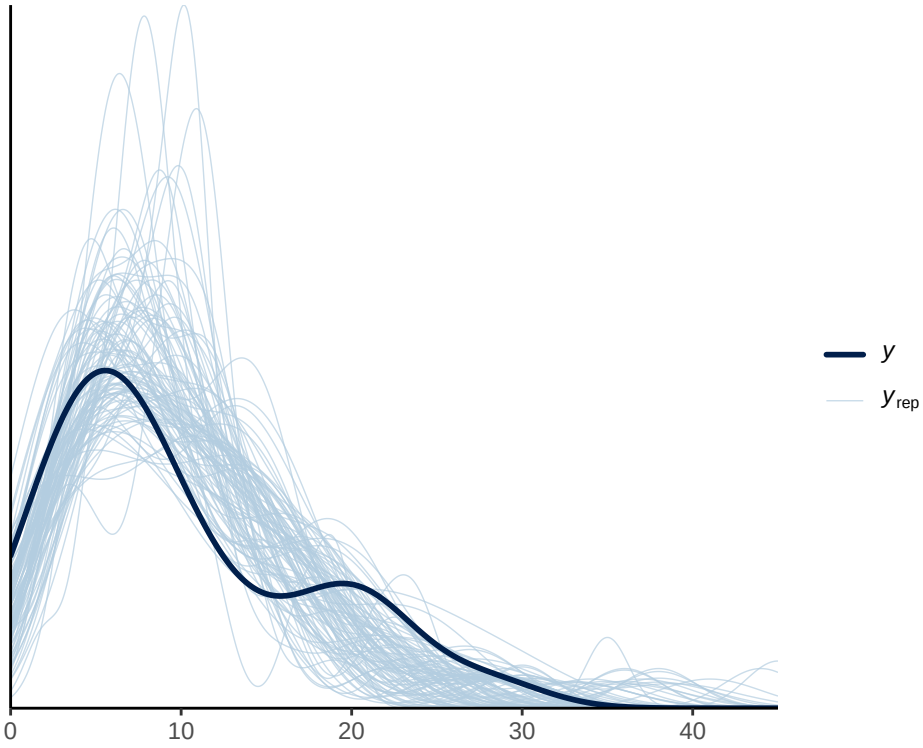

```
# No parece haber una tendencia, lo cual sugiere que no ganaríamos mucho
# incluyendo la precipitación.
```

Esa es la forma más robusta de comparar la predictiva posterior con los datos, pero también podemos usar al paquete `bayesplot`, de nuestros amigos de `Stan`, para visualizar el ajuste de otras formas.

```
# Trasponemos la matriz de datos simulados porque a bayesplot le gustan las
# iteraciones de la posterior como filas (las teníamos en las columnas)
ysim_t <- t(ysim)

# posterior predictive intervals en función de algo
ppc_intervals(
  y = datos$fires, # observaciones
  yrep = ysim_t,   # simulaciones de la predictiva posterior
  x = datos$fwi,   # alguna predictora de interés
  prob = 0.9       # probabilidad de los intervalos
) +
labs(x = "FWI", y = "Número de incendios")
```




```
# Y la densidad marginal de la respuesta, según lo observado (y) y los sets de
# datos simulados (yrep). Como graficará una curva por set de datos simulado,
# será mejor elegir al azar unos pocos (menos que 4000)
use <- sample(1:S, 100) # elige 100 al azar
ppc_dens_overlay(y = datos$fires, yrep = ysim_t[use, ])
```



Con estas herramientas podemos evaluar dónde falla el modelo y así intentar mejorarlo.

Cuánto hay que mejorarlo, depende del contexto.

0.3 Interpretación de modelos

Si el modelo se adecúa razonablemente a los datos, podemos comenzar a evaluar qué implica la posterior estimada. (Aunque a veces, visualizar el ajuste también puede ayudar a identificar potenciales problemas.)

En algunos casos los parámetros en sí resultan de interés, pero en otros casos, sólo nos interesan cantidades derivadas de ellos.

Ejemplo: nos interesa evaluar cómo varía λ en función del FWI.

Cada muestra s de la posterior ($\{\alpha_s, \beta_s\}$) define una función

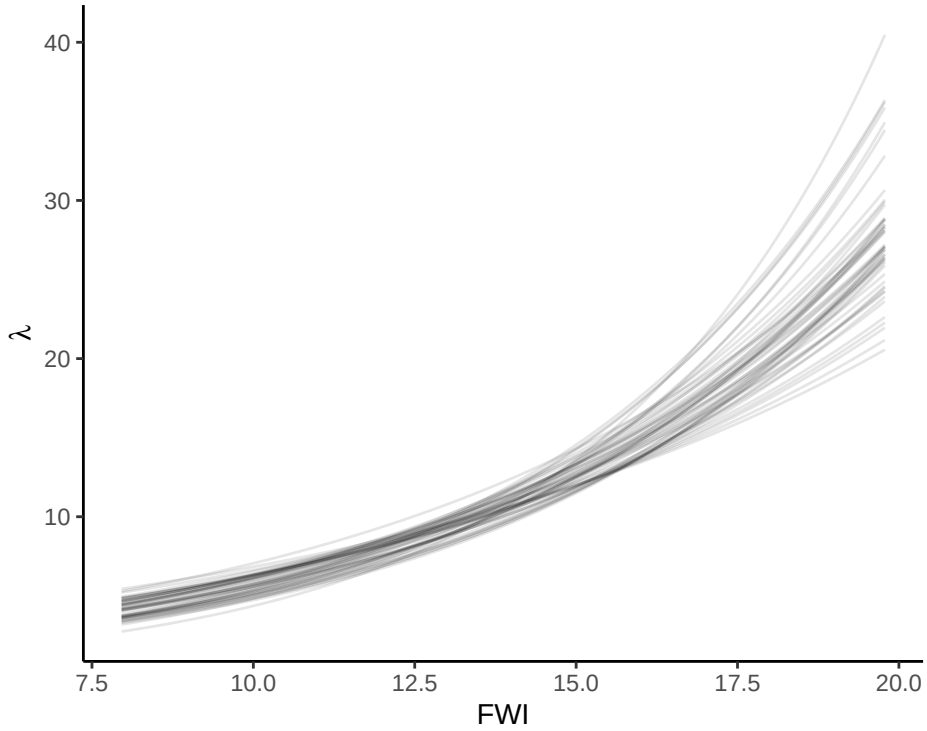
$$\lambda_s = f_s(\text{FWI}) = \exp(\alpha_s + \beta_s \text{ FWI})$$

.

Si nos interesa describir esa función considerando la incertidumbre en la estimación, podemos evaluarla sobre una secuencia de FWI utilizando todas o algunas muestras de la posterior.

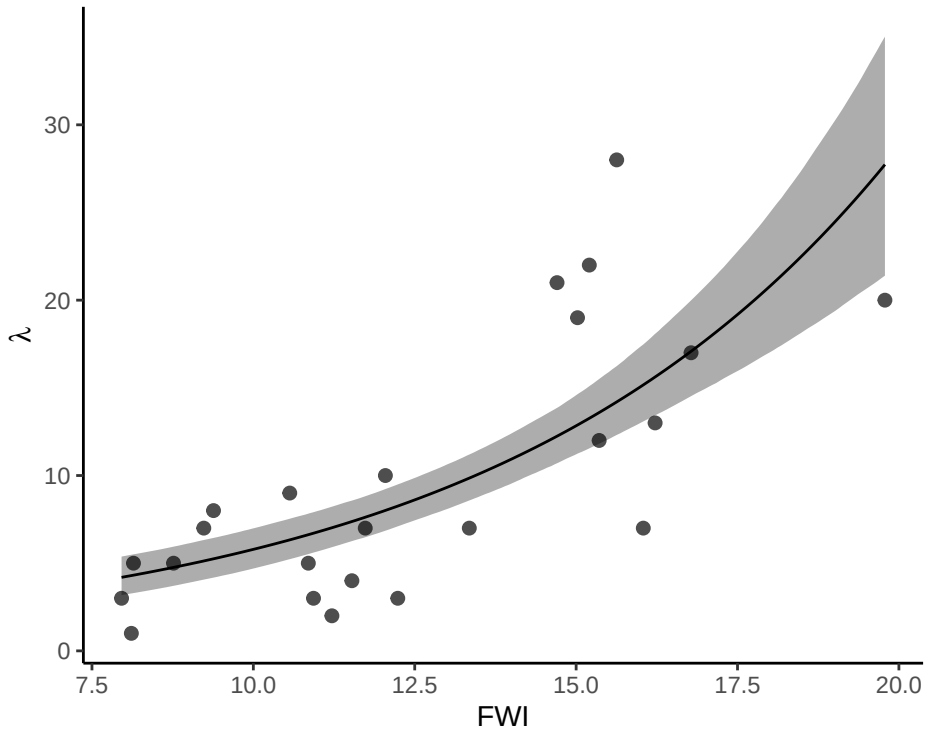
Importante: si guardamos las muestras en una matriz, donde cada fila es una iteración, no podemos mezclar valores de distintas filas. Esto es porque estaríamos rompiendo la estructura de correlaciones de la posterior conjunta.

```
nrep <- 200 # largo de la secuencia de FWI
fwi_seq <- seq(min(datos$fwi), max(datos$fwi), length.out = nrep)

# Guardaremos la función evaluada sobre la secuencia de fwi en una matriz,
# donde cada columna corresponderá a una muestra de la posterior
lambda_mat <- matrix(NA, nrep, S)

for (s in 1:S) {
  lambda_mat[, s] <- exp(d$a[s] + d$b[s] * fwi_seq)
}

# Podemos graficar algunas curvas ("spaghetti plot").
# Ordenaremos la matriz para graficarla con ggplot
ncurves <- 50
use <- sample(1:S, ncurves) # elegimos 150 curvas al azar
lambda_mat_sub <- as.data.frame(lambda_mat[, use])
lambda_mat_sub$fwi <- fwi_seq

# Elongamos la matriz
llong <- pivot_longer(
  lambda_mat_sub, cols = all_of(1:ncurves),
  names_to = "sim", values_to = "lambda"
)

ggplot(llong, aes(fwi, lambda, group = sim)) +
  geom_line(alpha = 0.1) +
  labs(x = "FWI", y = expression(lambda))
```



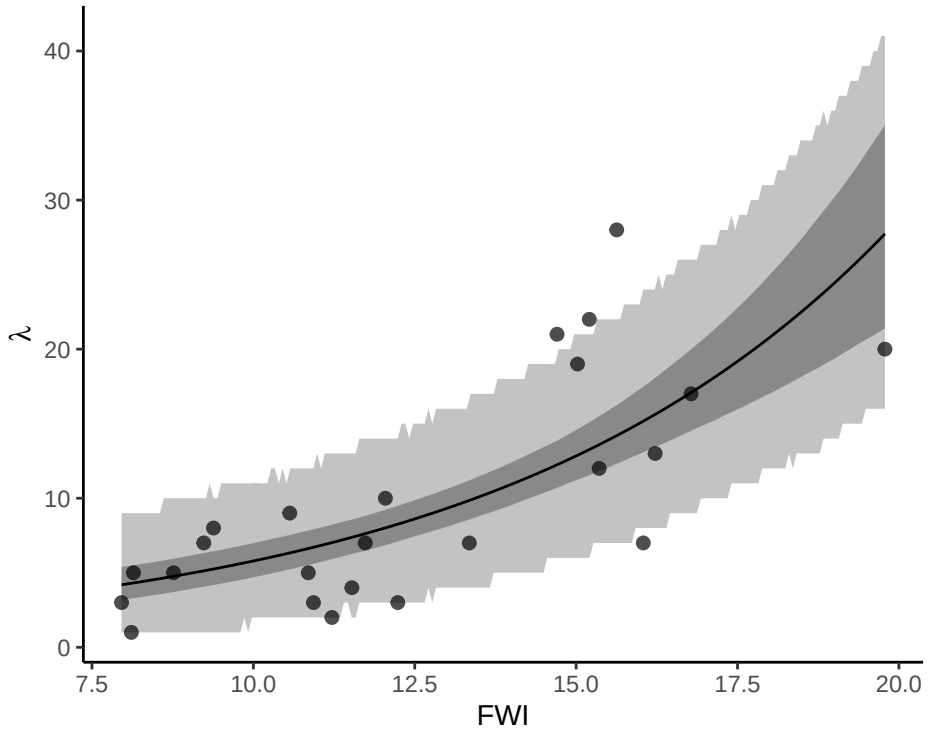

```r
# Lambda es una cantidad derivada de alpha, beta y el fwi. Todo lo que se calcula
# a partir de los parámetros tiene su propia distribución posterior. Y acá
# tenemos, además, una distribución posterior de lambda para cada valor de
# fwi que evaluemos.

# Otra forma de graficar la función es resumir la distribución de cada lambda,
# es decir, la distribución de lambda marginal a alpha y beta para cada x.

pred <- data.frame(
  fwi = fwi_seq,
  lambda_mean = rowMeans(lambda_mat),
  lambda_lwr = apply(lambda_mat, 1, quantile, probs = 0.025),
  lambda_upr = apply(lambda_mat, 1, quantile, probs = 0.975)
)

# apply() es una forma concisa de loopear. equivale a hacer
for (i in 1:N) {
  pred$lambda_lwr[i] <- quantile(lambda_mat[i, ], probs = 0.025)
}
# Es decir, aplica la función quantile, con argumento probs = 0.025, sobre el
# array llamado "lambda_mat", iterando sobre la dimensión 1 (filas)
```

```r
# Ahora graficamos la predicción media y su incertidumbre
ggplot(pred, aes(fwi, lambda_mean, ymin = lambda_lwr, ymax = lambda_upr)) +
  geom_ribbon(alpha = 0.4, color = NA) +
  geom_line() +
  labs(x = "FWI", y = expression(lambda)) +
  # Agregamos los datos
  geom_point(aes(fwi, fires), data = datos, inherit.aes = F,
             alpha = 0.7, size = 2)
```




```r
# La cinta muestra los percentiles 2.5 y 97.5 % de la distribución posterior de
# lambda. Pero también podemos explorar los percentiles extremos de la
# distribución predictiva posterior de y.
# Para eso, simulamos nuevas observaciones sobre la secuencia de FWI.
# (Antes, para el chequeo posterior, habíamos simulado nuevos valores de y
# usando los valores observados de FWI, no sobre una secuencia.)

ysim_seq <- matrix(NA, nrep, S)
for (s in 1:S) {
  ysim_seq[, s] <- rpois(nrep, lambda = lambda_mat[, s])
}

# Obtenemos los cuantiles extremos:
```

```r
pred$y_lwr <- apply(ysim_seq, 1, quantile, probs = 0.025)
pred$y_upr <- apply(ysim_seq, 1, quantile, probs = 0.975)

# Ahora graficamos la predicción media y su incertidumbre
ggplot(pred, aes(fwi, lambda_mean, ymin = lambda_lwr, ymax = lambda_upr)) +
  # Intervalo para y
  geom_ribbon(aes(fwi, ymin = y_lwr, ymax = y_upr), inherit.aes = F,
              alpha = 0.3, color = NA) +
  # Intervalo para lambda
  geom_ribbon(alpha = 0.4, color = NA) +
  geom_line() +
  labs(x = "FWI", y = expression(lambda)) +
  # Agregamos los datos
  geom_point(aes(fwi, fires), data = datos, inherit.aes = F,
             alpha = 0.7, size = 2)
```




```r
# La cinta más ancha es donde esperamos ver los datos con 95 % de probabilidad;
# la cinta más fina es donde esperamos ver la media de muchos datos, con 95 %
# de probabilidad.
# El más ancho se suele llamar intervalo de predicción; el más fino, intervalo de
# credibilidad para la media.
```

0.3.1 Ejemplo con más predictoras

[Datos de Alinari et al. 2023]

$$
\begin{aligned}
y_i &\sim \text{beta-binomial}(\mu_i, \phi, S_i) \\
\mu_i &= \frac{1}{1 + \exp(-\eta_i)} \\
\eta_i &= \alpha + \beta_D \, D_i + \beta_E \, E_i + \beta_H \, H_i
\end{aligned}
$$

$$
\begin{aligned}
\alpha &\sim \text{normal}(0, 10) \\
\beta_{\cdot} &\sim \text{normal}(0, 5) \\
\phi &\sim \text{log-normal}(\log(10), 2)
\end{aligned}
$$

y: Número de semillas germinadas
N: Número de semillas sembradas
D: Daño por fuego
E: Altidud
H: Altura inicial del árbol

Ajustamos el modelo

```
rm(list = ls()) # Limpiamos el entorno

# Cargamos datos
datos <- read.csv(here::here("datos", "alinari_data_espinillo.csv"))

# Conservamos los que tienen datos de germinación
datos <- subset(datos, !is.na(germinadas) & sembradas > 0)

# Estandarizamos las predictoras para tener efectos comparables y para
# facilitar la interpretación de las previas
D_mean <- mean(datos$danio)
D_sd <- sd(datos$danio)

E_mean <- mean(datos$altitud)
E_sd <- sd(datos$altitud)

H_mean <- mean(datos$altura)
H_sd <- sd(datos$altura)

datos$D <- (datos$danio - D_mean) / D_sd
```

```r
datos$E <- (datos$altitud - E_mean) / E_sd
datos$H <- (datos$altura - H_mean) / H_sd

# Datos para Stan
stan_data <- list(
  N = nrow(datos),
  y = datos$germinadas,
  S = 120,

  D = datos$D,
  E = datos$E,
  H = datos$H
)

# Compilamos modelo
model <- cmdstan_model(here::here("modelos", "germinadas.stan"))
```

```
Warning in readLines(stan_file): incomplete final line found on
'/home/ivan/Insync/Curso Modelos y Datos - CRUB -
Gure25/curso-bayes-25/modelos/germinadas.stan'
```

```r
# Muestreamos
fit <- model$sample(data = stan_data, parallel_chains = 4)
```

```
Running MCMC with 4 parallel chains...

Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 1 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 1 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 1 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 1 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 1 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 1 Exception: beta_binomial_lpmf: Second prior sample size parameter[19] is 0, but must
Chain 1 If this warning occurs sporadically, such as for highly constrained variable types I
Chain 1 but if this warning occurs often then your model may be either severely ill-conditio
```

```
Chain 1
Chain 1 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 1 Exception: beta_binomial_lpmf: Second prior sample size parameter[19] is 0, but must
Chain 1 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 1 but if this warning occurs often then your model may be either severely ill-conditio
Chain 1
Chain 1 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 1 Exception: beta_binomial_lpmf: First prior sample size parameter[1] is 0, but must b
Chain 1 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 1 but if this warning occurs often then your model may be either severely ill-conditio
Chain 1
Chain 1 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 1 Exception: beta_binomial_lpmf: Second prior sample size parameter[1] is 0, but must
Chain 1 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 1 but if this warning occurs often then your model may be either severely ill-conditio
Chain 1
Chain 1 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 1 Exception: beta_binomial_lpmf: Second prior sample size parameter[19] is 0, but must
Chain 1 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 1 but if this warning occurs often then your model may be either severely ill-conditio
Chain 1

Chain 2 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 2 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 2 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 2 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 2 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 2 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 2 Iteration: 1200 / 2000 [ 60%]  (Sampling)
```

```
Chain 2 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 2 Exception: beta_binomial_lpmf: First prior sample size parameter[1] is inf, but must
Chain 2 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 2 but if this warning occurs often then your model may be either severely ill-conditio
Chain 2
Chain 2 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 2 Exception: beta_binomial_lpmf: Second prior sample size parameter[3] is 0, but must
Chain 2 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 2 but if this warning occurs often then your model may be either severely ill-conditio
Chain 2
Chain 2 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 2 Exception: beta_binomial_lpmf: Second prior sample size parameter[19] is 0, but must
Chain 2 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 2 but if this warning occurs often then your model may be either severely ill-conditio
Chain 2
Chain 2 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 2 Exception: beta_binomial_lpmf: Second prior sample size parameter[21] is 0, but must
Chain 2 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 2 but if this warning occurs often then your model may be either severely ill-conditio
Chain 2
Chain 2 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 2 Exception: beta_binomial_lpmf: Second prior sample size parameter[6] is 0, but must
Chain 2 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 2 but if this warning occurs often then your model may be either severely ill-conditio
Chain 2
Chain 3 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 3 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 3 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 3 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3 Iteration:  700 / 2000 [ 35%]  (Warmup)
```

```
Chain 3 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 3 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 3 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 3 Exception: beta_binomial_lpmf: First prior sample size parameter[28] is 0, but must
Chain 3 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 3 but if this warning occurs often then your model may be either severely ill-conditio
Chain 3
Chain 3 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 3 Exception: beta_binomial_lpmf: Second prior sample size parameter[1] is 0, but must
Chain 3 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 3 but if this warning occurs often then your model may be either severely ill-conditio
Chain 3
Chain 4 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4 Iteration:  100 / 2000 [  5%]  (Warmup)
Chain 4 Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4 Iteration:  300 / 2000 [ 15%]  (Warmup)
Chain 4 Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 4 Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4 Iteration:  700 / 2000 [ 35%]  (Warmup)
Chain 4 Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4 Iteration:  900 / 2000 [ 45%]  (Warmup)
Chain 4 Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4 Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4 Iteration: 1100 / 2000 [ 55%]  (Sampling)
Chain 4 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 4 Exception: beta_binomial_lpmf: Second prior sample size parameter[26] is 0, but must
Chain 4 If this warning occurs sporadically, such as for highly constrained variable types l
Chain 4 but if this warning occurs often then your model may be either severely ill-conditio
Chain 4
Chain 4 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 4 Exception: beta_binomial_lpmf: Second prior sample size parameter[26] is 0, but must
Chain 4 If this warning occurs sporadically, such as for highly constrained variable types l
```

```
Chain 4 but if this warning occurs often then your model may be either severely ill-conditio
Chain 4
Chain 4 Informational Message: The current Metropolis proposal is about to be rejected becau
Chain 4 Exception: beta_binomial_lpmf: Second prior sample size parameter[1] is 0, but must
Chain 4 If this warning occurs sporadically, such as for highly constrained variable types I
Chain 4 but if this warning occurs often then your model may be either severely ill-conditio
Chain 4
Chain 1 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 1 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 1 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 2 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 3 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 3 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4 Iteration: 1300 / 2000 [ 65%]  (Sampling)
Chain 4 Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 1 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 2 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 2 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 3 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 4 Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4 Iteration: 1700 / 2000 [ 85%]  (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 2 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 3 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4 Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4 Iteration: 1900 / 2000 [ 95%]  (Sampling)
Chain 4 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1 finished in 0.4 seconds.
```

```
Chain 2 finished in 0.4 seconds.
Chain 3 finished in 0.4 seconds.
Chain 4 finished in 0.4 seconds.

All 4 chains finished successfully.
Mean chain execution time: 0.4 seconds.
Total execution time: 0.5 seconds.
```

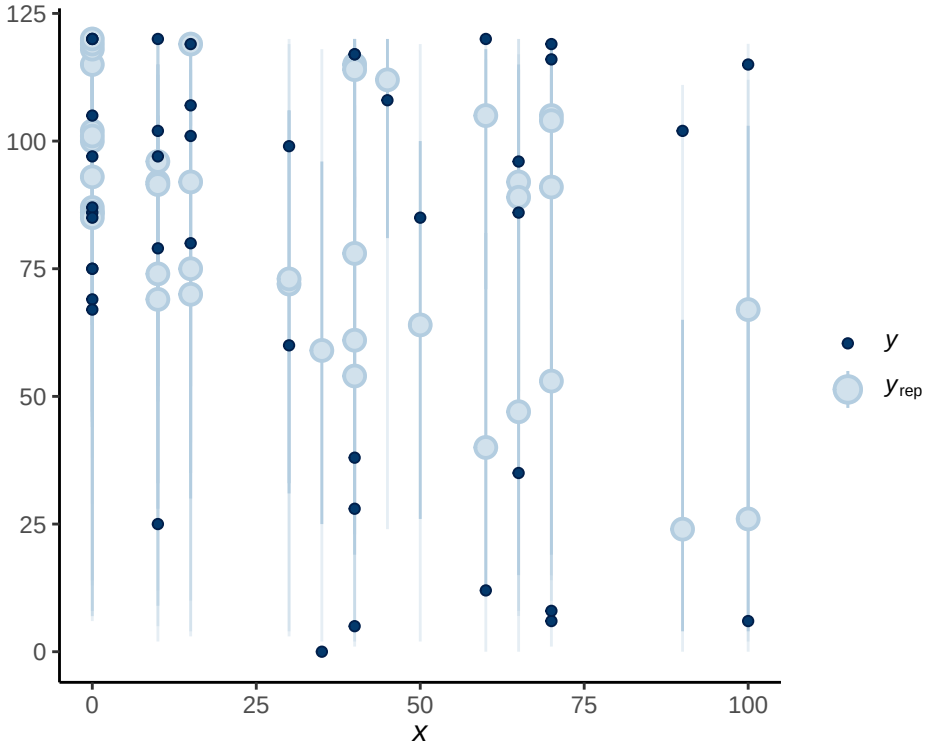
```
# Chequeamos MCMC
fit$cmdstan_diagnose()
```

```
Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Rank-normalized split effective sample size satisfactory for all parameters.

Rank-normalized split R-hat values satisfactory for all parameters.

Processing complete, no problems detected.
```

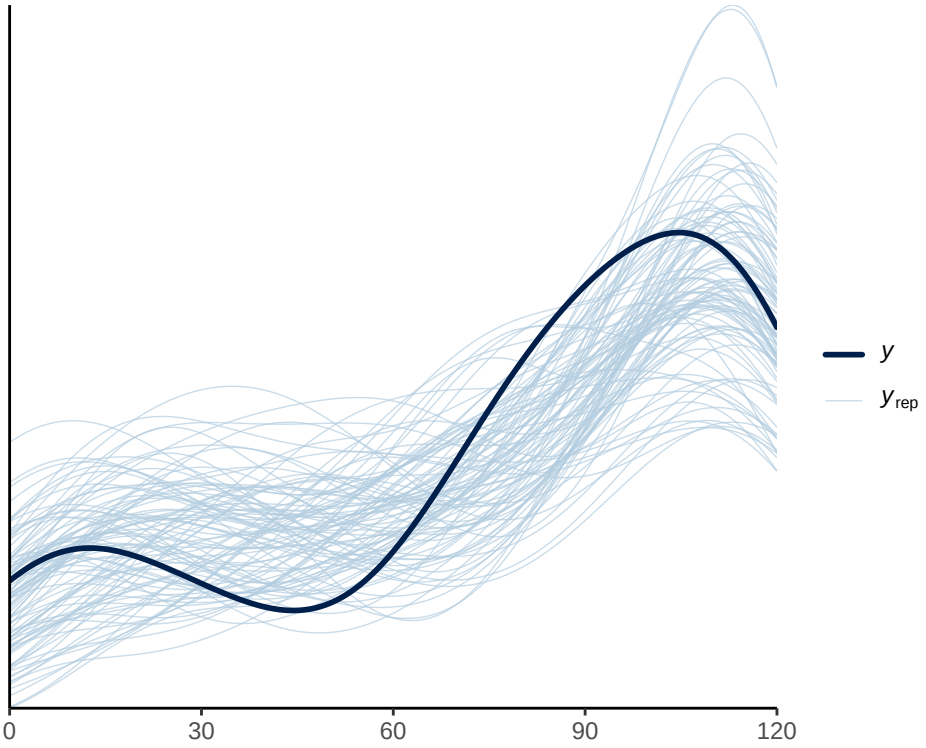
Verificación predictiva posterior

```
# Extraemos muestras de los parámetros que necesitamos
d <- fit$draws(
  c("alpha", "beta_D", "beta_E", "beta_H", "phi"), format = "draws_df"
)
colnames(d) <- c("a", "bD", "bE", "bH", "phi")
S <- nrow(d)
N <- nrow(datos)

# Verificamos el ajuste simulando de la predictiva posterior. Para simular de
# la beta-binomial en R usaremos el paquete extraDistr, que incluye la función
# rbbinom
ysim <- matrix(NA, N, S)
for (i in 1:S) {
  mu <- plogis( # equivale a inv_logit = logit inverso
    d$a[i] +
    d$bD[i] * datos$D + d$bE[i] * datos$E + d$bH[i] * datos$H
  )
  a <- mu * d$phi[i]
```

```
  b <- (1 - mu) * d$phi[i]
  ysim[, i] <- rbbinom(N, datos$sembradas, a, b)
}

ysim_t <- t(ysim)

# Miramos con bayesplot
ppc_intervals(
  datos$germinadas, yrep = ysim_t, x = datos$danio,
  prob = 0.5, prob_outer = 0.9
)
```



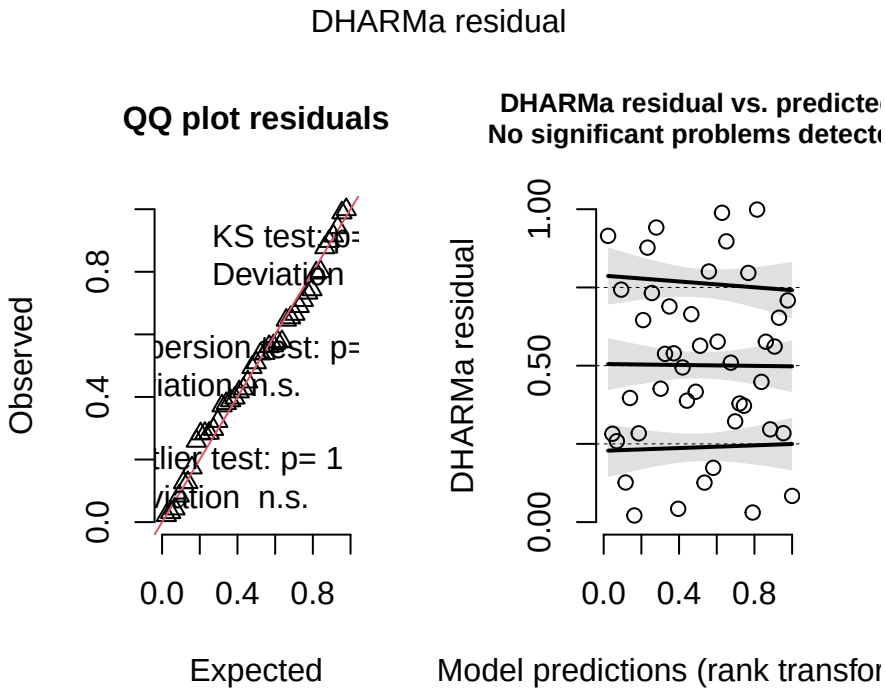

```
use <- sample(1:S, 100)
ppc_dens_overlay(datos$germinadas, yrep = ysim_t[use, ])
```



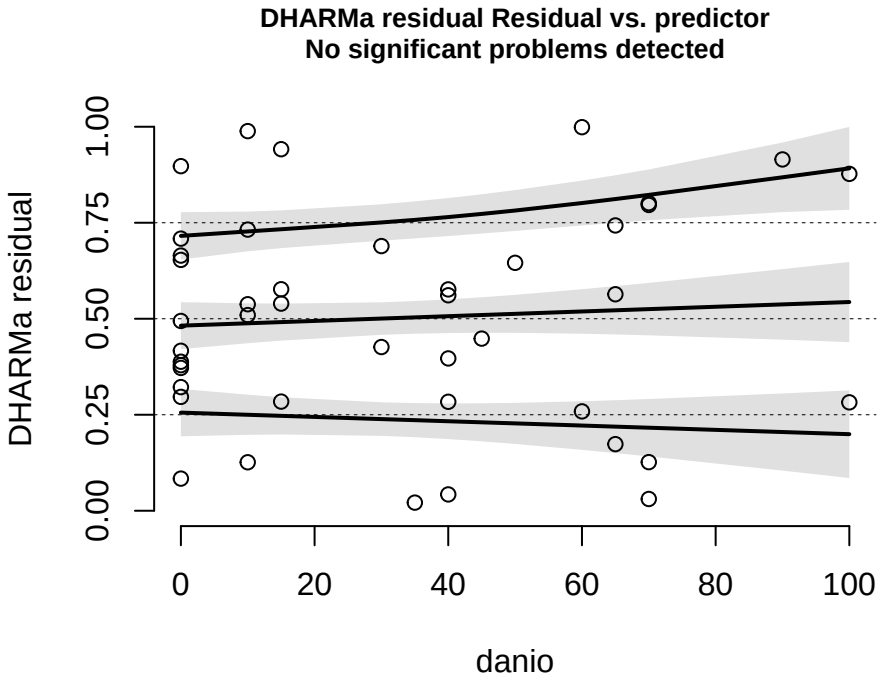

```
# DHARMa
res <- createDHARMa(ysim, datos$germinadas, integerResponse = T)
```

```
No fitted predicted response provided, using the mean of the simulations
```

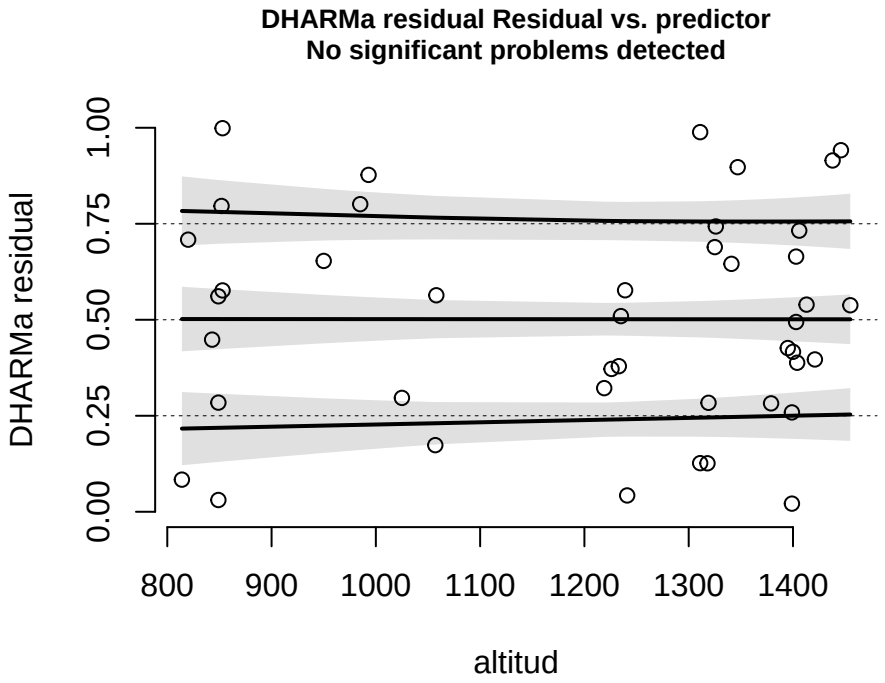
```
plot(res)
```



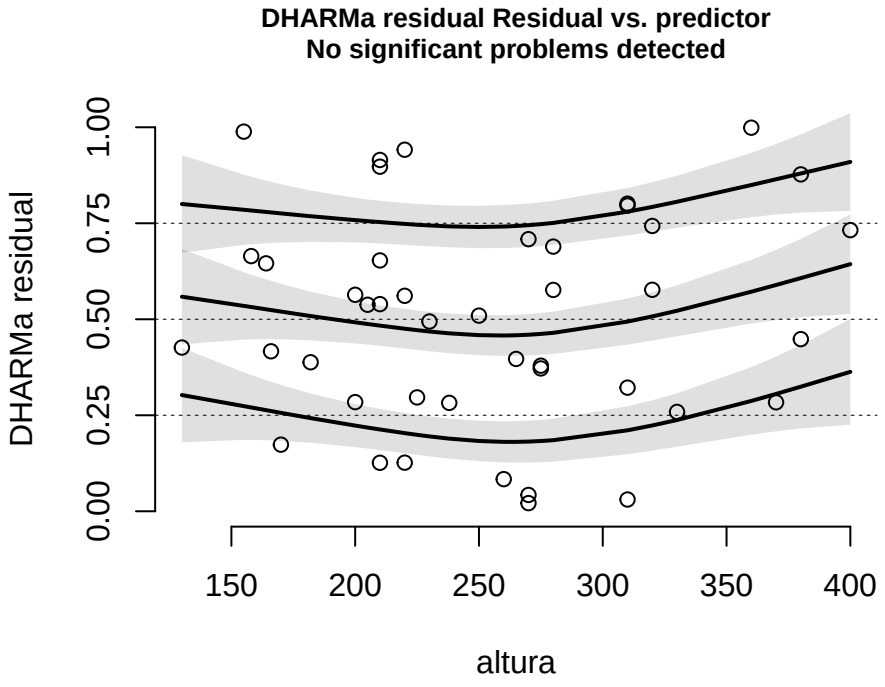

```
plotResiduals(res, form = datos$danio, rank = F)
```



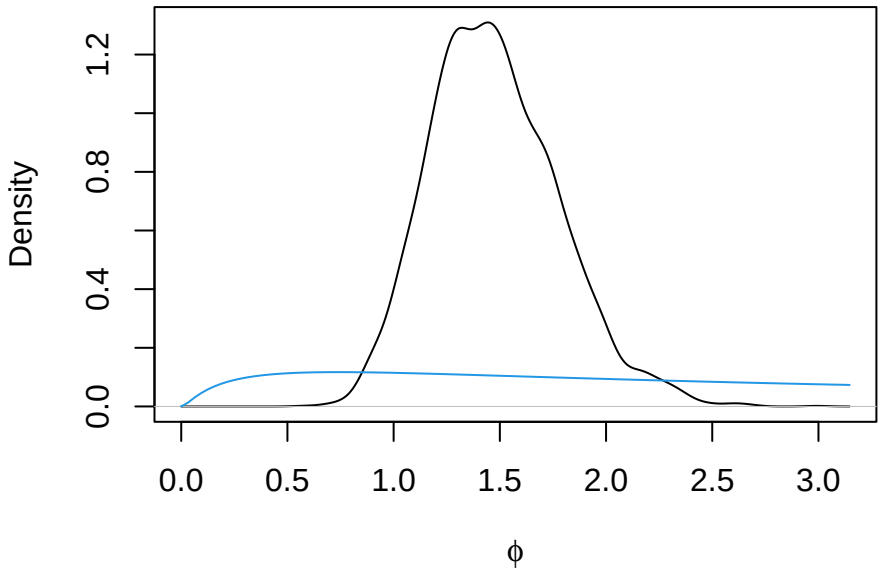
```
plotResiduals(res, form = datos$altitud, rank = F)
```




```
plotResiduals(res, form = datos$altura, rank = F)
```



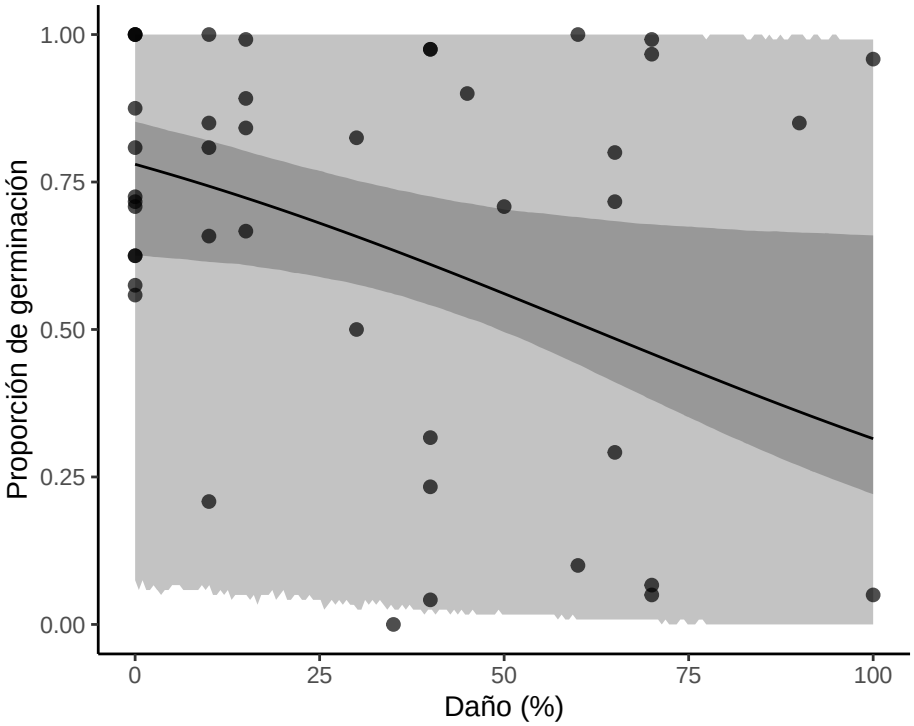

```
# Comparamos previa vs posterior de phi
plot(density(d$phi, from = 0), main = NA, xlab = expression(phi))
curve(dlnorm(x, log(7), 1.5), add = T, col = 4)
```



Visualizamos resultados de interés: predicciones parciales

```
# Predicción parcial para visualizar el efecto del daño por fuego (D).

# Graficaremos curvas basadas en nrep valores de daño en el rango observado
nrep <- 200
danio_seq <- seq(min(datos$danio), max(datos$danio), length.out = nrep)
# Estandarizamos la secuencia, porque los parámetros viven en escala estandarizada.
D_seq <- (danio_seq - D_mean) / D_sd

# Creamos data.frame de predicciones con las predictoras. Las predictoras no
# focales se fijan en su media (0 porque están estandarizadas), por eso es una
# predicción parcial.
pred <- data.frame(
  D = D_seq,
  E = 0,
  H = 0
)
pred$danio <- danio_seq # la agregamos en la escala original, para visualizar

# Matriz para llenar con predicciones de mu
mu_mat <- matrix(NA, nrep, S)

# Y matriz para llenar con valores simulados de semillas germinadas
y_mat <- matrix(NA, nrep, S) # se parece a ysim, pero tiene nrep filas, no N.

# Las llenamos
for (i in 1:S) {
  mu <- plogis( # equivale a inv_logit = logit inverso
    d$a[i] +
    d$bD[i] * pred$D + d$bE[i] * pred$E + d$bH[i] * pred$H
    # los dos últimos términos pueden obviarse, pero los dejamos para
    # explicitar.
  )
  mu_mat[, i] <- mu

  # calculamos alpha y beta para simular
  a <- mu * d$phi[i]
  b <- (1 - mu) * d$phi[i]
  y_mat[, i] <- rbbinom(nrep, datos$sembradas[1], a, b)
}

# Resumimos la distribución posterior de mu fila por fila
pred$mu_mean <- rowMeans(mu_mat)
pred$mu_lwr <- apply(mu_mat, 1, quantile, probs = 0.025)
pred$mu_upr <- apply(mu_mat, 1, quantile, probs = 0.975)
```

```r
# Convertimos la distribución de y en proporciones, para que esté en la misma
# escala que mu
y_mat_p <- y_mat / datos$sembradas[1]

# y resumimos
pred$y_lwr <- apply(y_mat_p, 1, quantile, probs = 0.025)
pred$y_upr <- apply(y_mat_p, 1, quantile, probs = 0.975)

# Calculamos la proporción observada en los datos
datos$prop <- datos$germinadas / datos$sembradas

# Graficamos
ggplot(pred, aes(danio, mu, ymin = mu_lwr, ymax = mu_upr)) +
  # Intervalo de predicción (posterior predictive)
  geom_ribbon(aes(danio, ymin = y_lwr, ymax = y_upr), inherit.aes = F,
              color = NA, alpha = 0.3) +
  # Intervalo para la media
  geom_ribbon(color = NA, alpha = 0.3) +
  # Media de la media
  geom_line() +
  # datos
  geom_point(aes(danio, prop), data = datos, inherit.aes = F,
             size = 2, alpha = 0.7) +
  labs(
    y = "Proporción de germinación",
    x = "Daño (%)"
  )
```



En esta predicción se fijaron en su media las demás predictoras. Esto hace que la predicción no sea comparable con los datos, ya que en los datos, las predictoras sí varían. En este caso, los datos no discrepan de la predicción porque las otras predictoras tienen efectos muy pequeños.

Para realizar otras predicciones hay que ponerse creativo al crear la tabla con nuevos valores para las predictoras. Por ejemplo, podríamos graficar más de una curva, para 3 valores contrastantes de alguna otra predictora. Esto es útil para visualizar interacciones.

0.4 Simulaciones previas

Al definir las previas de un modelo con ajuste bayesiano necesitamos inicialmente posicionarnos en relación a qué efecto deseamos de las previas. Algunas opciones pueden ser:

- reflejar conocimiento previo sobre el sistema,
- restringir levemente los parámetros para que tomen valores razonables,
- restringir fuertemente los parámetros para que el modelo sea estimable,

- influir lo menos posible en el resultado, limitándose a garantizar la convergencia de algoritmos de estimación.

Independientemente de la postura, para definir previas necesitamos poder traducir información de formas variadas a una distribución de probabilidad. Salvo que contemos con una intuición algebraica y probabilística envidiable, necesitamos graficar las implicancias de las previas en los modelos. Claramente, en contextos donde no hay información previa abundante, conviene investigar un poco qué tipo de previas se recomiendan para el tipo de modelo en cuestión. Para muchas estructuras hay recomendaciones disponibles.

En última instancia, si la definición de previas nos quita el sueño, podemos ajustar el modelo con distintas previas y comparar resultados. Esto incluso puede plantearse como un problema de comparación de modelos: cambiar la previa es equivalente a cambiar el modelo.

Para mostrar estrategias para definir previas utilizaremos los modelos ajustados arriba para el número de incendios y la cantidad de semillas germinadas. Supondremos que no hay mucha información previa y que deseamos distribuciones previas con poco efecto en el ajuste, de tal forma que la posterior esté dominada por la verosimilitud.

0.4.1 Simulaciones previas 1: incendios

El modelo estaba definido de la siguiente manera:

$$
\begin{aligned}
y_i &\sim \text{poisson}(\lambda_i) \\
\lambda_i &= \exp(\alpha + \beta\ FWI_i)
\end{aligned}
$$

$$
\begin{aligned}
\alpha &\sim \text{normal}(\mu_\alpha, \sigma_\alpha) \\
\beta &\sim \text{normal}(\mu_\beta, \sigma_\beta)
\end{aligned}
$$

$$
\begin{aligned}
\mu_\alpha &= \ ..., \sigma_\alpha = \ ... \\
\mu_\beta &= \ ..., \sigma_\beta = \ ...
\end{aligned}
$$

Si deseamos previas que generen valores razonables de λ sin afectar mucho el resultado, estamos buscando *previas levemente regularizadoras*.

Un buen primer paso es intentar interpretar los parámetros del modelo. Si eso es muy difícil, conviene graficar.

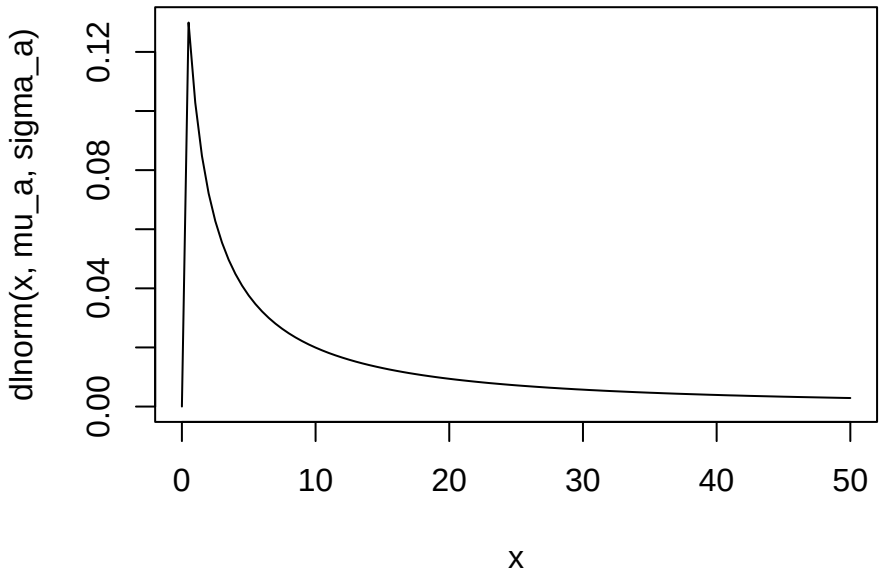
Consideremos que $\alpha = \log(\lambda)$ cuando $FWI = 0$. De esta manera,

$$\lambda_{FWI=0} \sim \text{log-normal}(\mu_\alpha, \sigma_\alpha).$$

Entonces, graficamos la lognormal variando sus parámetros (expresados en escala log).

Para graficar previas o sus implicancias rápidamente, resulta de gran utilidad la función `curve`, que evalúa una expresión conteniendo "x".

```
mu_a <- log(10) # la mediana de la lognormal sería 10
sigma_a <- 2
curve(dlnorm(x, mu_a, sigma_a), from = 0, to = 50)
```

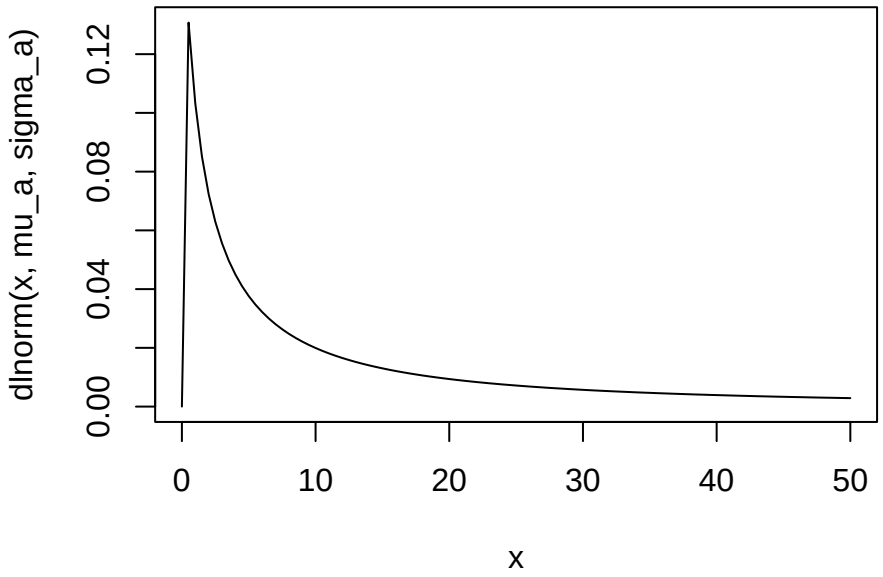


Con $\sigma_\alpha = 2$ la distribución se hace muy amplia. No nos dejemos engañar por su pico cerca de cero. Si bien en esa zona la densidad es alta, la masa de probabilidad (área bajo la curva) es baja, por lo que esta previa no sesgaría la estimación hacia cero si hay al menos unos pocos datos.

Pero si nos interesa no sesgar el resultado, otra estrategia es centrar la previa en la media de los datos, y darle un desvío grande:

```
# Cargamos datos
datos <- read.csv(here::here("datos", "barbera_data_fire_total_climate.csv"))

mu_a <- log(mean(datos$fires))
```

```r
sigma_a <- 2
curve(dlnorm(x, mu_a, sigma_a), from = 0, to = 50)
```

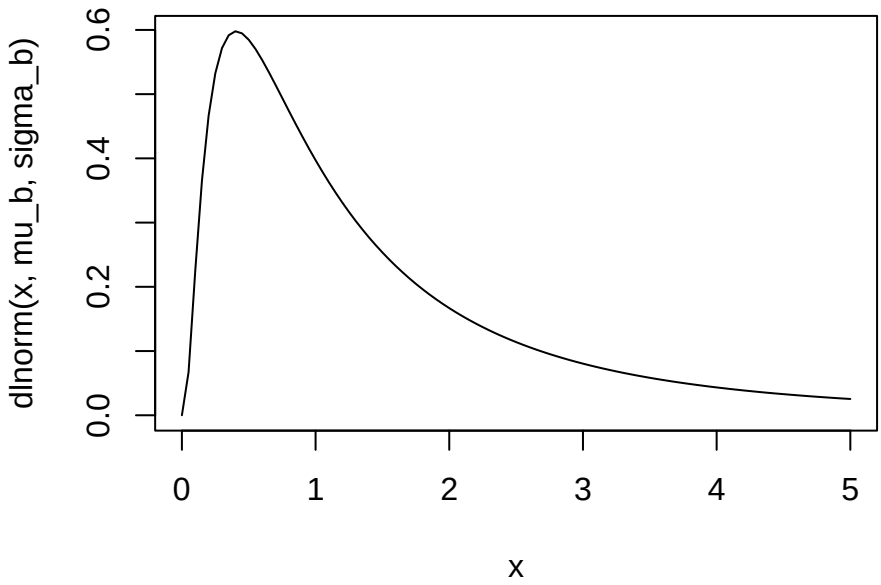


Si queremos informar aún menos, podemos aumentar un poco más $ _{ }.

Ahora nos centramos en β. Por suerte, para este modelo (GLM con enlace log), β tiene una interpretación clara: $\exp(\beta)$ es el factor en que aumenta λ cuando $x = FWI$ aumenta una unidad. Si suponemos que λ se duplica al aumentar el FWI una unidad, $\beta = \log(2)$. Pero como el FWI varía entre 0 y ~50, ese aumento sería absurdamente grande. Quizás es más sensato pensar en aumentos más pequeños. Probemos un aumento de 1.1 $\beta = \log(1.1)$.

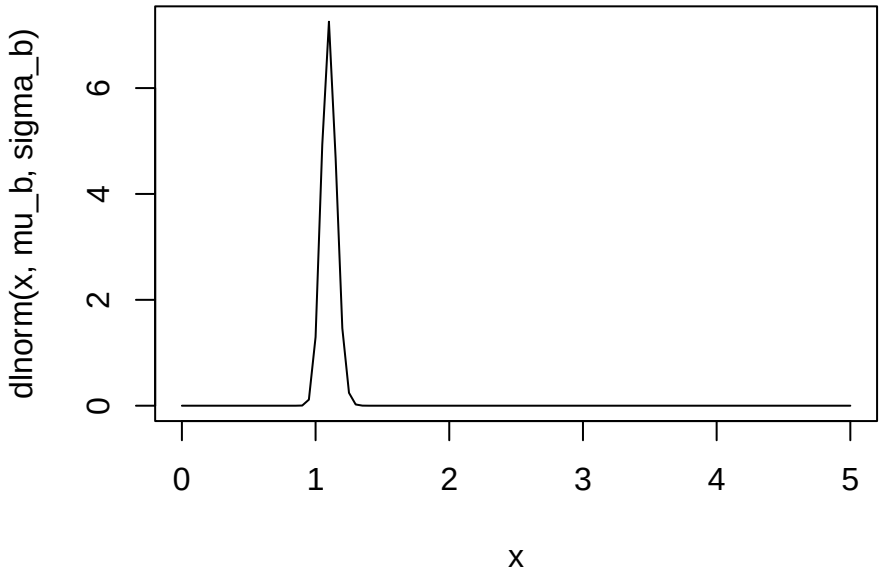
Nuevamente, $\exp(\beta)$ sigue una distribución log-normal:

```r
mu_b <- log(1.1)
sigma_b <- 1
curve(dlnorm(x, mu_b, sigma_b), from = 0, to = 5)
```



Esta previa tiene mediana y media positiva (1.1), pero también permite valores menores a 1, que implican disminución del número de incendios con el FWI. Si esto no nos parece razonable, podemos modificar la previa para dejar poca probabilidad abajo de 1 (restricción blanda) o truncar la previa en 1, para que sólo permita aumentos. Iremos por el primer enfoque.

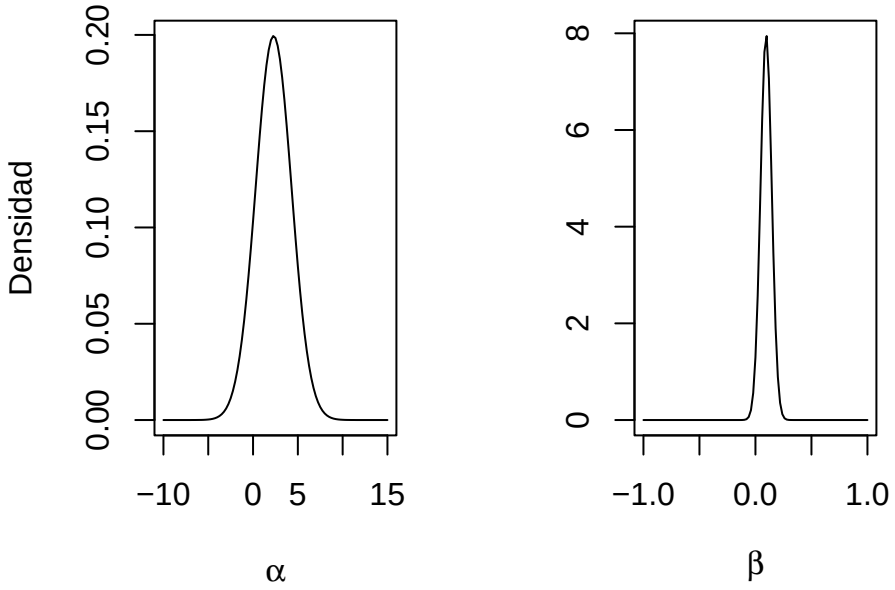
```
mu_b <- log(1.1)
sigma_b <- 0.05
curve(dlnorm(x, mu_b, sigma_b), from = 0, to = 5)
```



Ahora es una previa más razonable, quizás algo informativa. Pero cuán informativa es dependerá de cuán informativos sean los datos.

Ahora graficamos las previas elegidas para α y β (escala log, son normales):

```
par(mfrow = c(1, 2))
curve(dnorm(x, mu_a, sigma_a), from = -10, to = 15, xlab = expression(alpha),
      ylab = "Densidad")
curve(dnorm(x, mu_b, sigma_b), from = -1, to = 1, xlab = expression(beta),
      ylab = NA)
```

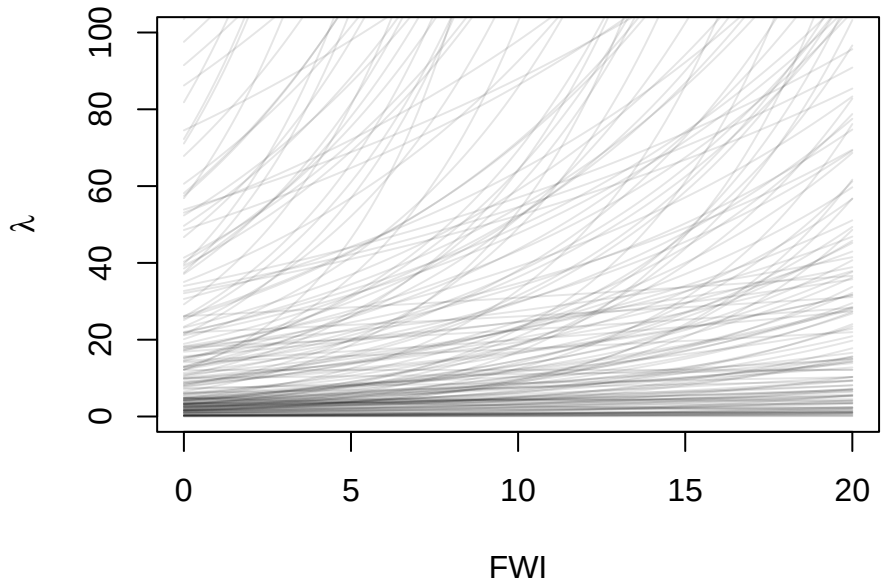



```
par(mfrow = c(1, 1))
```

Ahora podemos simular valores de estas previas para explorar qué curvas de $\lambda = f(FWI)$ implican:

```
light_col <- rgb(0, 0, 0, 0.1)

# gráfico inicial
a <- rnorm(1, mu_a, sigma_a)
b <- rnorm(1, mu_b, sigma_b)
curve(exp(a + b * x), from = 0, to = 20, ylim = c(0, 100), col = light_col,
      xlab = "FWI", ylab = expression(lambda))

# Repetimos con muchas muestras, usando el argumento "add = TRUE" en curve()
for (i in 1:200) {
  a <- rnorm(1, mu_a, sigma_a)
```

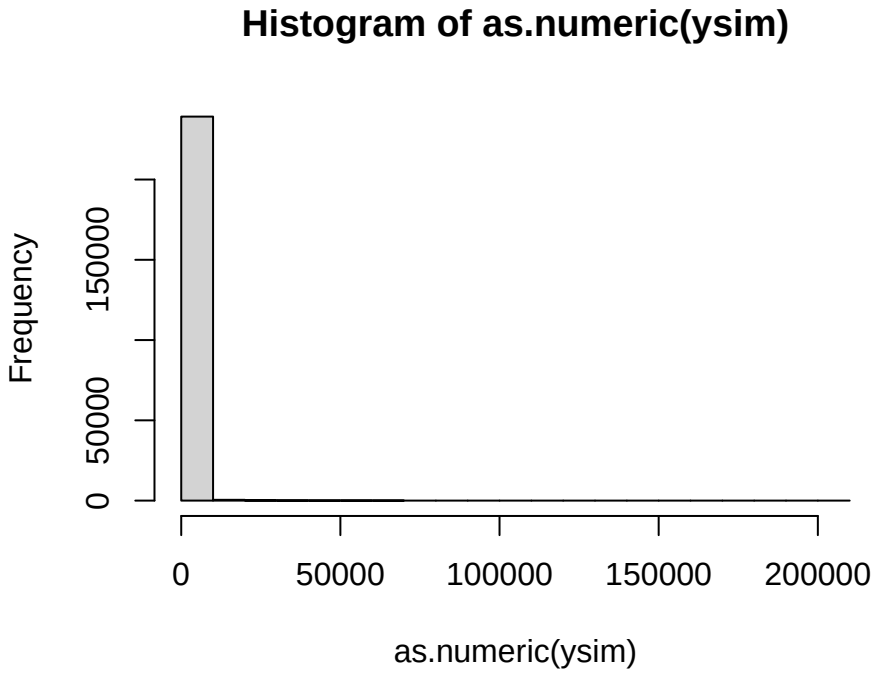
```
  b <- rnorm(1, mu_b, sigma_b)
  curve(exp(a + b * x), col = light_col, add = T)
}
```



Muchas curvas parecen poco razonables. Pero bueno, estábamos buscando previas poco informativas, ¿no?

También podemos simular datos utilizando los valores observados de la predictora. Esto implica simular de la distribución predictiva previa (análoga a la distribución predictiva posterior).

```
N <- nrow(datos)
S <- 10000
ysim <- matrix(NA, N, S)

for (i in 1:S) {
  a <- rnorm(1, mu_a, sigma_a)
  b <- rnorm(1, mu_b, sigma_b)
  lambda <- exp(a + b * datos$fwi)
  ysim[, i] <- rpois(N, lambda)
}

hist(as.numeric(ysim))
```




```
quantile(as.numeric(ysim), probs = c(0.9, 0.95, 0.98))
```

```
    90%     95%     98%
 496.00 1039.00 2427.02
```

Los datos simulados pueden tomar valores muy altos. Esto puede ser razonable si no queremos que la previa influya mucho en los resultados.

NEGBIN, phi

Beta-Binomial check, germinadas. enlace logit, pred estandarizadas.

R2