Resolución bayesiana de modelos lineales mixtos aplicados a muestras grandes

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

María López Álvarez

Trabajo Fin de Máster

Máster en Análisis de Datos e Inteligencia de Negocios

Universidad de Oviedo

Tutor: Carlos Enrique Carleos Artime

Julio 2024

Índice general

Índice de tablas

Índice de figuras

Resumen

Los modelos lineales mixtos para la mejora genética, como estimar la influencia del sexo y los progenitores de un individuo en el peso al nacimiento, se pueden resolver mediante métodos Markov Chain Monte Carlo (MCMC). Existen diferentes formas de abordar su implementación, y se trabajará con una base de datos real (931 689 animales) tratando de encontrar la más adecuada.

En un primer análisis descriptivo se observa como los machos tendrán un mayor peso al nacimiento que las hembras, sin embargo, poco se puede decir de los valores mejorantes o efectos genéticos. Por tanto, es necesario utilizar los métodos MCMC para obtener resultados más completos. En una primera implementación "rudimentaria", desarrollada por el usuario con el software R, se puede llegar a trabajar con una pequeña parte de la base de datos que apenas llega al 0.5 % del total. Es decir, esta primera opción no será suficiente para abordar problemas reales.

Estimar los pesos al nacimiento con paquetes de R parece ser una solución viable a la hora de aumentar el número de individuos con los que se trabaja. Con `rstan` se aumenta la cantidad de animales con los que se puede trabajar en un tiempo razonable, aunque se sigue sin obtener soluciones para más del 1 % de individuos de la población.

Con el paquete `MCMCglmm` se da un paso adelante en la cantidad de individuos, llegando a trabajar con más del 1.5 % de la población en un tiempo de varias horas. Esto supone una ventaja con respecto al otro paquete disponible, sin embargo, sigue sin solucionar el problema para la población completa.

Posteriormente se busca trabajar con otros lenguajes de programación. En C, Simon C. Heath propone una solución, aunque su implementación no puede ejecutarse en nuestros ordenadores (128 GB RAM) con la población completa de la que se dispone. Andrés

Legarra proporciona un algoritmo implementado en Fortran que permite resolver el problema completo, con todos los individuos. En algo más de un día se obtienen resultados fiables para todos los animales de la población.

Serán los machos los que proporcionen un mayor peso al nacimiento, pero, además, se puede ver qué individuo (o individuos) se debería seleccionar como madre o padre para obtener el mayor beneficio futuro. La influencia de un individuo en el modelo vendrá determinada por la cantidad de familiares en los datos, así como por el valor del peso al nacimiento en ellos, incluyendo si es desconocido o no. Por otro lado, este método permitirá determinar que la mitad de la variabilidad del peso al nacimiento se hereda.

Abstract

Linear mixed models for genetic improvement, such as estimating the influence of sex and parents of an individual on birth weight, can be solved by Markov Chain Monte Carlo (MCMC) methods. As there are different ways to approach their implementation, working with a real database (931 689 animals) allows find the most suitable one.

A first descriptive analysis shows that males will have a higher birth weight than females, however, little can be said about the breeding values or genetic effects. Therefore, it is necessary to use MCMC methods to obtain more complete results. In a first "rudimentary" implementation, developed by the user with the R software, it is possible to work with a small part of the database that hardly reaches 0.5 % of the total. In other words, this first option will not be sufficient to tackle real problems.

Estimating birth weights with R packages seems to be a viable solution to increase the number of individuals to work with. With `rstan`, the number of animals that can be worked with in a reasonable time is increased, although solutions are still not obtained for more than 1 % of individuals in the population.

The `MCMCglmm` package takes a step forward in the number of individuals, working with more than 1.5 % of the population in a time of several hours. This is an advantage over the other available package, however, it still does not solve the problem for all population.

Later, work is sought with other programming languages. In C, Simon C. Heath proposes a solution, although it cannot be implemented with the complete population available. Andrés Legarra provides an algorithm in Fortran that allows to solve the complete problem, with all the individuals. In a little more than a day, reliable results are obtained for all the animals in the population.

It will be the males that provide the greatest birth weight, but, in addition, it can be seen which individual or individuals should be selected as mother or father to obtain the greatest future benefit. The influence of an individual in the model will be determined by the number of relatives in the data, as well as by the value of birth weight in the data, including whether it is unknown or not. On the other hand, this method will make it possible to determine that half of the variability of birth weight is inherited.

Introducción

Uno de los objetivos fundamentales de la ganadería es la selección de individuos que puedan producir generaciones futuras superiores. Los ganaderos intentan elegir a sus animales de forma que su descendencia maximice los beneficios. Al principio, esta selección se hacía de manera intuitiva, observando y eligiendo a los animales que mostraban las características deseadas. Sin embargo, con el avance del conocimiento y la tecnología, se han desarrollado métodos científicos y matemáticos que permiten realizar esta selección de manera más precisa y eficiente.

Dentro de los posibles métodos de resolver los modelos lineales mixtos para la mejora genética se encuentran los métodos Markov Chain Monte Carlo (MCMC), que se sitúan dentro de la inferencia bayesiana cuando se trabaja con modelos complejos. Sin embargo, estos métodos también tienen algunos inconvenientes, como podría ser que cada ejecución proporciona un valor distinto y que es necesario utilizar largas cadenas para obtener resultados estables.

Para resolver un problema de mejora genética con métodos MCMC existen diferentes algoritmos, algunos de los cuales se estudiarán en este trabajo. Dentro de las posibilidades se encuentra una implementación propia realizada con anterioridad ([1]), lo cual servirá de punto de partida para obtener resultados con una base de datos real, previa limpieza y depuración.

En R existen diferentes paquetes y librerías que proporcionan alternativas para la resolución de modelos lineales mixtos mediante métodos MCMC. A lo largo de este trabajo se estudiarán algunas de ellas, tratando de encontrar aquella más eficiente y más parecida a la implementación propia realizada. Además, se estudiarán tanto las ventajas como los inconvenientes que podría presentar cada una de ellas, prestando especial cuidado a aquellos aspectos desconocidos de la programación.

Otras opciones para resolver el problema será utilizar otros lenguajes de programación, como C y Fortran, los cuales permiten implementar algoritmos de resolución más eficientes y sofisticados. Además, se verá como se requiere la utilización de equipos con una cantidad de memoria RAM bastante grande para trabajar con todos los datos del conjunto, algo que no es posible realizar con un equipo convencional.

Capítulo 1

Análisis descriptivo

En este capítulo se busca hacer un primer análisis descriptivo del conjunto de datos con el que se va a trabajar. Además, se desarrollará una limpieza de los datos, en busca de datos atípicos o erróneos. Se definirán las variables, se calcularán medidas de centralización y se realizarán gráficos con el objetivo de conocer la información disponible.

Se trabaja con un conjunto de datos donde se dispone de 6 variables medidas sobre 932 519 individuos. Estas variables son:

- *nombre*: nombre del individuo.
- *id*: identificador del individuo.
- *pa*: identificador del padre del individuo.
- *ma*: identificador de la madre del individuo.
- *pesonac*: peso al nacimiento del individuo en kilogramos (kg).
- *sexo*: sexo del individuo.

Se observa que existen 830 individuos repetidos, por lo que se crea un nuevo conjunto de datos sin los datos duplicados. Es decir, se trabaja con 931 689 datos. Existen datos

desconocidos (NAs) en cada una de las variables (Tabla 1.1).

Variable	**Valores desconocidos**	**Valores conocidos**
nombre	0	931 689
id	0	931 689
pa	287 273	644 416
ma	89 690	841 999
pesonac	862 363	69 326
sexo	47	931 642

Tabla 1.1: Valores conocidos y desconocidos por variables.

Como se puede comprobar no existe ningún individuo, cuyo *nombre* o *id* sea desconocido, es decir, podemos identificar correctamente a todos ellos. Sin embargo, vemos como se dispone de pocos datos de *pesonac*, es decir, el peso al nacimiento se conoce únicamente para 69 326 individuos de los 931 689 disponibles.

Veamos si tenemos datos incorrectos, como puede ser hembras que sean padre de otro individuo o machos que sean madre de otro individuo. Creamos dos conjuntos de datos, uno de machos y otro de hembras, y los comparamos con los padres y madres del conjunto completo. Se sacan dos conclusiones:

- Ninguna hembra como padre.
- Ningún macho como madre.

En la variable *sexo*, se cuenta con dos posibles opciones, ser macho (M) o ser hembra (H), siendo la distribución de los individuos como 388 662 machos y 542 980 hembras. Por tanto, ya hay que notar que se conocen mayor número de hembras. Además, hay que tener presente que de 47 individuos se desconoce el sexo.

Podemos hacer una primera descripción del peso al nacimiento del individuo. Tenemos valores del peso al nacimiento entre 20 kg y 60 kg con una media de 38.460 kg. Hacemos un histograma para ver cómo se distribuyen (Figura 1.1), que nos sirve para comprobar que no parecen existir datos atípicos en los pesos al nacimiento, puesto que no se observan saltos significativos.

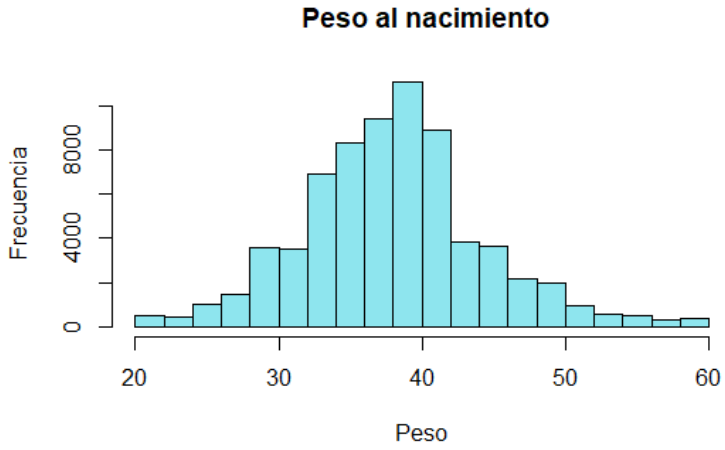

Figura 1.1: Peso al nacimiento (histograma).

1.1. Influencia del sexo

Para hacer un análisis directo de la influencia del sexo en el peso al nacimiento hacemos primero una estimación puntual, para ello calculamos directamente la media del peso al nacimiento para machos (μ_M) y hembras (μ_H):

$$\mu_M = 39.875 \text{ kg},$$

$$\mu_H = 37.169 \text{ kg}.$$

Otra forma de ver cómo se comporta el peso al nacimiento en función del sexo, es calcular los intervalos de confianza correspondientes a la media en cada grupo. Para ello utilizaremos las técnicas bootstrap con 1000 réplicas y calcularemos el intervalo por el

método “normal”. Para los machos, se obtiene:

$$IC_{\mu_M} = (39.805, 39.948) \text{ kg}.$$

En cambio, para las hembras, tenemos el siguiente resultado:

$$IC_{\mu_H} = (37.108, 37.230) \text{ kg}.$$

A la vista de los resultados de la estimación puntual de la media por sexos, la estimación por intervalo de la media por sexo y el diagrama de cajas (Figura 1.2) del peso al nacimiento, se espera que las hembras tengan un menor peso al nacimiento que los machos.

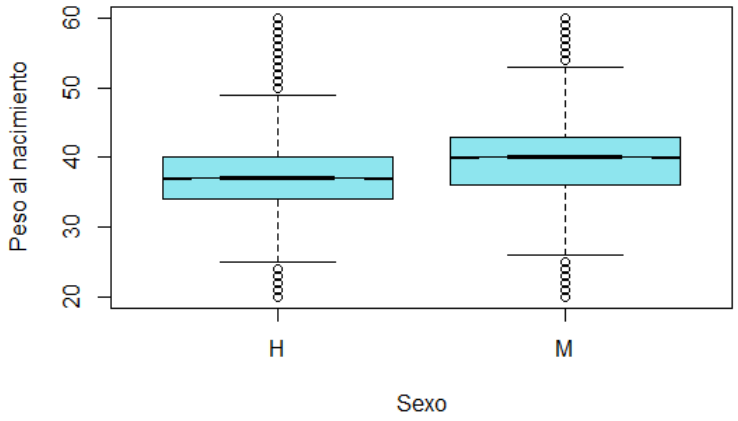

Figura 1.2: Peso al nacimiento por sexo (diagrama de cajas).

En estas conclusiones hay que tener en cuenta que no de todos los individuos se conoce el peso al nacimiento, lo que afecta a las estimaciones. Además, hay que recordar que no se dispone del mismo número de datos para las hembras que para los machos.

1.2. Influencia del padre

Para hacer un análisis directo de la influencia del padre en el peso al nacimiento consideramos los 5 individuos con mayor número de hijos. Hacemos primero una estimación

puntual, para ello calculamos directamente la media del peso al nacimiento para cada uno de los padres ($\mu_{id\,padre}$). Además, como en el caso del sexo, se utilizan técnicas bootstrap para calcular los correspondientes intervalos de confianza:

$$\begin{aligned}
\mu_{60205} &= 36.143 \text{ kg}, & IC_{\mu_{60205}} &= (35.837, 36.449) \text{ kg},\\
\mu_{32797} &= 38.014 \text{ kg}, & IC_{\mu_{32797}} &= (37.667, 38.361) \text{ kg},\\
\mu_{49834} &= 41.621 \text{ kg}, & IC_{\mu_{49834}} &= (41.165, 42.078) \text{ kg},\\
\mu_{75590} &= 39.007 \text{ kg}, & IC_{\mu_{75590}} &= (38.554, 39.460) \text{ kg},\\
\mu_{27027} &= 37.971 \text{ kg}, & IC_{\mu_{27027}} &= (37.474, 38.469) \text{ kg}.
\end{aligned}$$

A la vista de los resultado de las estimación puntual, la estimación por intervalo y el digrama de cajas (Figura 1.3), los individuos cuyo padre tenga id=49 834 tendrán mayor peso al nacimiento y los individuos cuyo padre tenga id=60 205 tendrán menor peso al nacimiento. Además, se debe tener en cuenta que no todos los individuos estudiados disponen del mismo número de hijos.

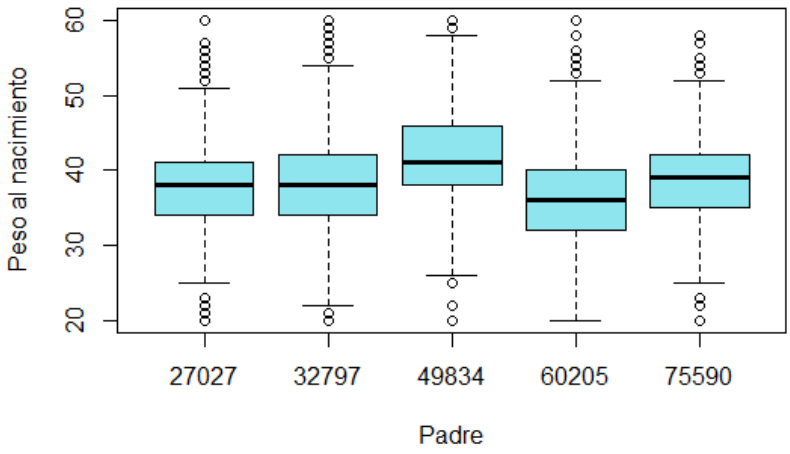

Figura 1.3: Peso al nacimiento por padre (diagrama de cajas).

Capítulo 2

Métodos MCMC

En este capítulo se propondrán distintos ejemplos y se solucionarán con el programa en R implementado en [1] para resolver el modelo mixto a través de métodos MCMC. Para ello usamos como efecto fijo el sexo ($\vec{\beta}$, que podrá tomar $p \in \mathbb{N}$ valores distintos) y como efecto aleatorio la información genética ($\vec{u}$, donde se dispone de $q \in \mathbb{N}$ individuos con sus relaciones de parentesco), tratando de predecir el peso al nacimiento ($\vec{y}$, con valor conocido para $n \in \mathbb{N}$ individuos).

2.1. Desarrollo teórico

En esta sección se explica de manera resumida (para más detalle ver [1]) el desarrollo matemático que se encuentra detrás de la aplicación de métodos MCMC a modelos lineales mixtos. Teniendo en cuenta que f representará la densidad a priori de las variables (parámetros y regresores), la fórmula es la siguiente:

$$\vec{y} = X\vec{\beta} + Z\vec{u} + \vec{e},$$

donde

- $\vec{y} \in \mathcal{M}_{n\times 1}$: vector respuesta con $\vec{y} \mid \vec{\beta}, \vec{u}, \sigma_e^2 \sim \mathcal{N}(X\vec{\beta} + Z\vec{u}, I\sigma_e^2)$.
- $\vec{\beta} \in \mathcal{M}_{p\times 1}$: vector de efectos fijos con $f(\vec{\beta}) \propto$ constante.

- $\vec{u} \in \mathcal{M}_{q\times 1}$: vector de efectos aleatorios con $\vec{u} \mid A\sigma_u^2 \sim \mathcal{N}(0, A\sigma_u^2)$ donde A es la matriz de parentescos (cada elemento a_{ij} hace referencia a la relación genética entre el individuo i y el individuo j) y $\sigma_u^2 \mid \nu_u, s_u^2 \sim \chi^{-2}(\nu_u, s_u^2)$.
- $X \in \mathcal{M}_{n\times p}$ y $Z \in \mathcal{M}_{n\times q}$: matrices de incidencia.
- $\vec{e} \in \mathcal{M}_{n\times 1}$: vector de residuos con $\vec{e} \mid I\sigma_e^2 \sim \mathcal{N}(0, I\sigma_e^2)$ donde $\sigma_e^2 \mid \nu_e, s_e^2 \sim \chi^{-2}(\nu_e, s_e^2)$.

Es importante notar que estamos trabajando con inferencia bayesiana, donde los parámetros del modelo (σ_u^2, σ_e^2) son variables aleatorias que dependen de hiperparámetros (ν_u, s_u^2, ν_e, s_e^2).

Entonces, considerando que $\vec{\beta}$, $\vec{u}$, σ_u^2 y σ_e^2 tienen distribuciones a priori independientes y utilizando el Teorema de Bayes, se puede determinar la distribución condicionada a posteriori de los parámetros:

$$f(\vec{\beta}, \vec{u} \mid \sigma_u^2, \sigma_e^2, \vec{y}) \propto f(\vec{\beta}) f(\vec{u} \mid \sigma_u^2) f(\vec{y} \mid \vec{\beta}, \vec{u}, \sigma_e^2).$$

Para poder determinar completamente esta distribución, es necesario introducir cierta notación:

$$\vec{\alpha} = [\vec{\beta}', \vec{u}']' \qquad W = [X, Z]$$

$$\Sigma^{-1} = \begin{bmatrix} 0 & 0 \\ 0 & A^{-1}\frac{\sigma_e^2}{\sigma_u^2} \end{bmatrix} \qquad \hat{\vec{\alpha}} = (W'W + \Sigma^{-1})^{-1} W'\vec{y}$$

$$C = W'W + \Sigma^{-1} \qquad \begin{bmatrix} C^{11} & C^{12} \\ C^{21} & C^{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}^{-1} = C^{-1}.$$

Así, se tiene que la distribución correspondiente a los efectos sigue una distribución normal:

$$\vec{\alpha} \mid \sigma_u^2, \sigma_e^2, \vec{y} \sim \mathcal{N}(\hat{\vec{\alpha}}, C^{-1}\sigma_e^2). \tag{2.1}$$

Dividiendo $\vec{\alpha}$ en dos partes arbitrarias, $\vec{\alpha} = [\vec{\alpha}_1', \vec{\alpha}_2']$, se busca determinar la distribución a posteriori condicionada, $\vec{\alpha}_1 \mid \vec{\alpha}_2, \sigma_u^2, \sigma_e^2, \vec{y}$, que deberá ser normal. Por otro lado, $C\hat{\vec{\alpha}} =$

$W'\vec{y} = \vec{r}$ es un sistema de ecuaciones cuya solución, $\vec{\hat{\alpha}} = C^{-1}\vec{r}$, se puede descomponer:

$$\vec{\hat{\alpha}}_1 = C^{11}\vec{r}_1 + C^{12}\vec{r}_2,$$

$$\vec{\hat{\alpha}}_2 = C^{21}\vec{r}_1 + C^{22}\vec{r}_2.$$

Aplicando propiedades de la distribución normal multivariante, se tienen los parámetros correspondientes a la distribución condicionada anterior:

$$\mathrm{E}(\vec{\alpha}_1 \mid \vec{\alpha}_2, \sigma_u^2, \sigma_e^2, \vec{y}) = C_{11}^{-1}(\vec{r}_1 - C_{12}\vec{\alpha}_2),$$

$$\mathrm{Var}(\vec{\alpha}_1 \mid \vec{\alpha}_2, \sigma_u^2, \sigma_e^2, \vec{y}) = C_{11}^{-1}.$$

El algoritmo Metropolis-Hastings (MH) es el algoritmo de simulación de cadenas de Márkov más general y de él se pueden derivar los demás, permitiendo muestrear cada componente de $\vec{\alpha}$ ($\vec{\alpha}_i$) a partir de las demás ($\vec{\alpha}_{-i}$). En este caso, trabajamos simulando un único valor de $\vec{\beta}$ o $\vec{u}$ de cada vez (α_i), por lo que nos interesa la siguiente expresión:

$$\alpha_i \mid \vec{\alpha}_{-i}, \sigma_u^2, \sigma_e^2, \vec{y} \sim \mathcal{N}(c_{ii}^{-1}((W'\vec{y})_i - \vec{C}_{i,-i}\vec{\alpha}_{-i}), c_{ii}^{-1}\sigma_e^2), \tag{2.2}$$

donde c_{ii}^{-1} es el inverso del elemento (i,i) de la matriz C y $\vec{C}_{i,-i}$ es la fila i-ésima de la matriz C sin el elemento c_{ii}.

Además, las distribuciones a posteriori de las varianzas se podrán escribir utilizando distribuciones conocidas que facilitarán el posterior muestreo:

- $\sigma_u^2 \mid \vec{\beta}, \vec{u}, \sigma_e^2, \vec{y} \sim \chi^{-2}(\nu_u', s_u'^2)$ con $\nu_u' = q + \nu_u$ y $s_u'^2 = \frac{\vec{u}'A^{-1}\vec{u} + \nu_u s_u^2}{q+\nu_u}$.
- $\sigma_e^2 \mid \vec{\beta}, \vec{u}, \sigma_u^2, \vec{y} \sim \chi^{-2}(\nu_e', s_e'^2)$ con $\nu_e' = n + \nu_e$ y $s_e'^2 = \frac{(\vec{y}-X\vec{\beta}-Z\vec{u})'(\vec{y}-X\vec{\beta}-Z\vec{u}) + \nu_e s_e^2}{n+\nu_e}$.

2.2. Programa propio

Se buscará construir cadenas markovianas de distintas distribuciones, empezando por el efecto del sexo ($\vec{\beta}$) y el valor mejorante o efecto genético ($\vec{u}$). Además también se deberán obtener las cadenas correspondientes a la varianza genética (σ_u^2), la varianza residual (σ_e^2) y la heredabilidad ($h^2 = \frac{\sigma_u^2}{\sigma_u^2+\sigma_e^2}$, proporción de la variabilidad debida a la herencia).

Caso 1

Estudiamos el efecto del sexo y el individuo considerando únicamente los 10 padres y 10 madres con mayor número de hijos. Antes de aplicar el modelo hay que seleccionar los individuos y ordenarlos para obtener la matriz de parentescos.

Con los individuos seleccionados y añadiendo los individuos faltantes (padres o madres que no están en el conjunto de datos), comprobamos que solo tenemos datos del peso la nacimiento para 4 de ellos, por lo que el vector $\vec{y}$ tendrá longitud 4; la matriz de efectos fijos, X, tendra 4 filas y 2 columnas (una por cada sexo); y la matriz de efectos aleatorios, Z, tendrá 4 filas y 58 columnas (esto nos indica que hay dos progenitores que se comparten entre los 20 individuos iniciales o que ya estaban en el conjunto de datos).

Si realizamos 100 000 iteraciones del algoritmo obtenemos que el efecto del sexo es:

$$\vec{\beta} = \begin{pmatrix} 30.977 \text{ kg} \\ 35.342 \text{ kg} \end{pmatrix}.$$

Por tanto, a partir del modelo MCMC, el sexo que tiene mayor peso al nacimiento es el segundo, que en este caso son las hembras.

Efectuamos una comparación de estos valores con las medias por sexos que se obtienen para esos datos:

$$\mu_M = 31.000 \text{ kg},$$

$$\mu_H = 35.333 \text{ kg}.$$

También lo podemos comparar con los valores del efecto del sexo que estima un modelo lineal, que en este caso es 35.333 kg para las hembras y 31.000 kg para los machos.

De todo esto no se pueden sacar buenas conclusiones porque hay que tener en cuenta que únicamente contamos con datos del peso al nacimiento de 4 de los individuos, de los

cuales 3 son hembras y uno macho. Podemos ver que de las tres formas se hacen estimaciones similares, teniendo más peso las hembras que los machos.

Esto último parece contradecir los resultados obtenidos en el análisis descriptivo, que estimaba mayor peso a los machos que a las hembras. Sin embargo, podemos encontrar una explicación sencilla en los datos, ya que estamos considerando conocidos el peso de un único macho y de 3 hembras, lo que afecta en gran medida al resultado. Para poder sacar conclusiones fiables tendríamos que tener mayor número de datos de cada sexo.

Caso 2

Estudiamos ahora el efecto del sexo y del individuo en los hijos del padre con mayor número de hijos. Es decir, seleccionamos los hijos del individuo con $id = 60\,205$ (tiene 8009 hijos) y eliminamos las madres que no están en el propio subconjunto.

- $\vec{y}$ tiene los pesos al nacimiento de 1420 individuos.
- La matriz X (efecto del sexo) tiene dimensiones 1420×2.
- La matriz Z (efecto aleatorio) tiene dimensiones 1420×8010.

Al aplicar el método, se observa que con una sola iteración el programa tarda más de 15 minutos en ejecutarse, lo cual impide realizar un número suficientemente elevado en un tiempo aceptable.

Aunque las siguientes conclusiones no son útiles ni fiables, puesto que una iteración no es suficiente, se obtiene:

- Efecto del sexo:
$$\vec{\beta} = \begin{pmatrix} 36.354 \text{ kg} \\ 35.932 \text{ kg} \end{pmatrix}.$$
Por tanto, el sexo que tiene mayor peso al nacimiento es el primero, que en nuestro caso son los machos. Además se pueden comparar estos resultados con los obtenidos a

partir de las medias por sexos ($\mu_M = 37.142\,\text{kg}$ y $\mu_H = 35.163\,\text{kg}$) , que coinciden con las estimaciones que proporciona un modelo lineal. Por lo tanto, podemos concluir que se espera que los machos tengan mayor peso al nacimiento.

- Valor mejorante: el modelo nos da como mejor individuo el que tiene $id = 636\,930$, que es una hembra. Esto no coincide con los resultados que da el fenotipo, según el cual el primer individuo sería el de $id = 209\,306$.

En este último caso no se está haciendo el número de iteraciones necesario por lo que el resultado no nos permite obtener predicciones fiables que expliquen el comportamiento de los individuos según el sexo.

2.3. Introducción a la implementación

A continuación se buscarán librerías de R que traten de realizar el método visto anteriormente, reduciendo los tiempos de ejecución y dando resultados fiables. Para ello se trabajará con diversos programas que ya han sido implementados con anterioridad y que realizan un estudio similar al visto, en cada caso se tratará de estudiar tanto las diferencias como las similitudes con el desarrollo previo.

En este apartado veremos tres paquetes que, aunque no permiten resolver el problema propuesto, servirán de gran ayuda a la hora de preparar los datos para cumplir con el objetivo planteado. Para ilustrar las funciones, se trabajará con un ejemplo sencillo, cuyos datos aparecen reflejados en la Tabla 2.1.

2.3.1. Librería *pedigree*

El paquete `pedigree` [3] es una herramienta que permite trabajar con datos de pedigrí (genealogía de un animal, conjunto de antepasados de un animal). El paquete proporciona funciones para importar y exportar datos de pedigrí, visualizar y manipular pedigríes, y

Animal (id)	Padre (pa)	Madre (ma)
1	-	-
2	-	-
3	-	-
4	1	2
5	3	2
6	1	4

Tabla 2.1: Datos de los ejemplos.

realizar análisis genéticos. Con este paquete no se realizarán las estimaciones propiamente dichas, pero servirá de apoyo a la hora de hacer cálculos previos con los datos de pedigrí.

De este paquete se utilizarán dos funciones en especial, cuya finalidad será adaptar los datos a lo necesario:

- `orderPed`: a partir de unos datos de pedigrí los ordena de manera que los índices de los padres y madres sean inferiores a los de los hijos. Como ejemplo de esta función, se introduce el pedigrí original desordenado (según los identificadores: 1,4,5,6,2,3).

```
id <- c(1,4,5,6,2,3) # Identificadores de los individuos.
padre <- c(0,1,3,1,0,0) # Identificadores de los padres.
madre <- c(0,2,2,4,0,0) # Identificadores de las madres.
ped <- data.frame(id,madre,padre) # data.frame con el pedigrí.
ord <- orderPed(ped) # Orden correcto de los individuos.
ped2 <- ped[order(ord),]; ped2 # data.frame ordenado.
  id madre padre
1  1     0     0
5  2     0     0
2  4     2     1
```

```
6  3     0     0
3  5     2     3
4  6     4     1
```

No es el orden original, pero se observa como los progenitores están situados previamente a los hijos (por ejemplo, el individuo 3 aparece antes que el individuo 5, del cual es padre).

- `add.Inds`: a partir de unos datos de pedigrí añade los individuos faltantes, en este caso considera los progenitores que no están en los datos y los introduce en la base de datos considerando todos los valores como desconocidos (salvo el identificador).

```
id <- 4:6 # Identificadores de los individuos (conocidos).
padre <- c(1,3,1) # Identificadores de los padres.
madre <- c(2,2,4) # Identificadores de las madres.
ped3 <- data.frame(id,madre,padre)
ped3 <- add.Inds(ped3); ped3 # Pedigrí completo.
  id madre padre
1  2    NA    NA
2  1    NA    NA
3  3    NA    NA
4  4     2     1
5  5     2     3
6  6     4     1
```

Se observa como, aunque no en el orden original, añade los tres individuos desconocidos que son progenitores de los que aparecen en los datos. A estos "nuevos" individuos les asigna como desconocido el identificador del padre y de la madre.

2.3.2. Librería *ggroups*

El paquete `ggroups` [4] es una paquete de R muy similar a `pedigree`. Es una herramienta que permite procesar datos de pedigrí y realizar análisis genéticos básicos. Aunque contiene funciones de diversos tipos y que buscan objetivos varios, se trabajará principalmente con una de ellas.

La función `buildA` esta diseñada para construir la matriz de parentescos. Para ello es necesario introducirle los valores correspondientes al *id* de los individuos, así como los de los padres y madres. Además, trabaja con el valor 0 para los datos faltantes.

```
id <- 1:6 # Identificadores de los individuos.
padre <- c(0,0,0,1,3,1) # Identificadores de los padres.
madre <- c(0,0,0,2,2,4) # Identificadores de las madres.
ped <- data.frame(id = id, pa = padre, ma = madre)
buildA(ped)
    1    2   3    4     5     6
1 1.00 0.00 0.0 0.50 0.000 0.750
2 0.00 1.00 0.0 0.50 0.500 0.250
3 0.00 0.00 1.0 0.00 0.500 0.000
4 0.50 0.50 0.0 1.00 0.250 0.750
5 0.00 0.50 0.5 0.25 1.000 0.125
6 0.75 0.25 0.0 0.75 0.125 1.250
```

El resultado obtenido nos indica que grado de relación hay entre cada par de individuos. Por ejemplo, los individuos `1` y `2`, no tienen nigun tipo de parentesco, lo que se observa en que el valor correpondiente (segundo elemento de la primera fila y primer elemento de la segunda fila) es `0`. Sin embargo, en el cuarto elemento de la primera fila, el valor es `0.5`, lo que significa que hay una relación padre/hijo. De manera similar ocurre con el resto de

valores, siendo el caso más especial el del individuo `6` consigo mismo, que tiene un valor `1.25`, por ser su madre hija de su padre.

2.3.3. Librería *pedigreemm*

La librería `pedigreemm` [5] está especializada en el análisis de modelos mixtos y modelos lineales generalizados aplicados a datos de pedigrí. Este paquete tiene muchas funciones y características útiles para el análisis: permite estimar coeficientes de consanguinidad (parentesco de dos individuos que tienen un antepasado común próximo) y relaciones de parentesco, modelar la variación genética y residual, simular datos de pedigrí, visualizar la estructura de pedigrí y analizar la correlación y la covarianza.

Algunas de las funciones más importante de las que dispone y que serán de ayuda a la hora de resolver el problema propuesto son:

- `pedigree:` a partir de datos del identificador del individuo y de los progrenitores, calcula un objeto de la clase "pedigree".

```
id <- 1:6 # Identificadores de los individuos.
padre <- c(NA,NA,NA,1,3,1) # Identificadores de los padres.
madre <- c(NA,NA,NA,2,2,4) # Identificadores de las madres.
ped4 <- pedigree(sire = padre, dam = madre, label = id); ped4
  sire  dam
1 <NA> <NA>
2 <NA> <NA>
3 <NA> <NA>
4    1    2
5    3    2
6    1    4
```

- `getA`: calcula la matriz de parentescos, en formato "sparse" (`A`). Da un resultado similar a `ggroups::buildA`, pero se utiliza directamente sobre datos guardados como "pedigree", por lo que puede ser de especial utilidad en algunos casos particulares.

```
A <- getA(ped4); A # Matriz de parentescos.
     1    2   3    4     5     6
1 1.00 .    .   0.50 .     0.750
2 .    1.00 .   0.50 0.500 0.250
3 .    .    1.0 .    0.500 .
4 0.50 0.50 .   1.00 0.250 0.750
5 .    0.50 0.5 0.25 1.000 0.125
6 0.75 0.25 .   0.75 0.125 1.250
```

- `getAInv`: calcula directamente la matriz inversa de la matriz de parentescos.

```
AInv <- getAInv(ped4); AInv # Matriz inversa.
     1    2    3    4  5  6
1  2.0  0.5  0.0 -0.5  0 -1
2  0.5  2.0  0.5 -1.0 -1  0
3  0.0  0.5  1.5  0.0 -1  0
4 -0.5 -1.0  0.0  2.5  0 -1
5  0.0 -1.0 -1.0  0.0  2  0
6 -1.0  0.0  0.0 -1.0  0  2
```

- `editPed`: permite completar y ordenar los datos de pedigrí, de manera que coloca los padres y madres con un índice inferior al de sus hijos (de manera similar a `pedigree::orderPed`). Además, añade una nueva variable que indica la generación a la que pertenece cada individuo.

```
id <- c(1,4,5,6,2,3) # Identificadores de los individuos.
padre <- c(NA,1,3,1,NA,NA) # Identificadores de los padres.
madre <- c(NA,2,2,4,NA,NA) # Identificadores de las madres.
ped5 <- editPed(sire = as.character(padre),
                dam = as.character(madre),
                label = as.character(id)); ped5 # Pedigrí.
  label sire  dam gene
1     1 <NA> <NA>    0
5     2 <NA> <NA>    0
6     3 <NA> <NA>    0
2     4    1    2    1
3     5    3    2    1
4     6    1    4    2
```

Tanto el paquete `pedigree` como los paquetes `ggroups` y `pedigreemm` permiten tratar los datos de manera similar y disponen de más funciones que las vistas aquí. Además, hay que tener especial cuidado a la hora de introducir los datos ya que, según el paquete, requerirán un formato u otro. Por ejemplo, en el paquete `pedigree` y en el paquete `ggroups` los datos desconocidos se introducen como `0`, sin embargo, en `pedigreemm` se introducen como `NA`.

Con estos tres paquetes se podrían obtener resultados básicos para el modelo mixto, pero se pretende utilizar métodos MCMC por lo que es conveniente buscar librerías que faciliten su uso. Existen varias librerías de R que permiten resolver el problema propuesto, alguna de las cuales utiliza lenguajes de programación no vistos anteriormente. Por tanto, en los capítulos posteriores nos centraremos en dos de esas librerías: `rstan` y `MCMCglmm`.

Capítulo 3

Librería *rstan*

El paquete `rstan` es un interfaz de R para el lenguaje de programación estadística Stan [6]. Es un lenguaje que se utiliza para ajustar modelos bayesianos que son difíciles de ajustar con métodos clásicos y para realizar inferencia bayesiana en R. El paquete `rstan` dispone de diversas funciones que nos permiten introducir el texto en dicho lenguaje Stan. En este caso, se trabajará principalmente con la función `stan` aunque se utilizarán otras funciones pertenecientes a paquetes ya vistos que ayudan en los cálculos previos relacionados con la genética.

Este capítulo se dividirá en tres partes. Se comenzará explicando las peculiaridades del lenguaje de programación Stan de manera general. Posteriormente se comentará el código necesario en el paquete `rstan` para obtener las estimaciones buscadas. Y, por último, se trabajará con diversos casos y se interpretarán los resultados obtenidos.

3.1. Lenguaje de programación Stan

Stan es un lenguaje de programación que se utiliza principalmente para el modelado estadístico y la estimación bayesiana. Este tipo de lenguaje fue diseñado por Andrew Gelman, un reconocido estadístico y científico de datos. Gelman y su equipo de la Universidad de Columbia han trabajado en la implementación de métodos bayesianos y han

colaborado en el desarrollo de herramientas para facilitar el análisis bayesiano. El lenguaje Stan fue diseñado principalmente para realizar inferencias estadísticas utilizando métodos MCMC. Stan se utiliza en diversos ámbitos para realizar predicciones, siendo los más comunes biología, física, ingeniería, ciencias sociales y negocios. Además dispone de diversas funciones que permiten desarrollar múltiples acciones. De este lenguaje de programación se pueden destacar las siguientes características:

- Es un lenguaje bastante flexible que permite describir modelos estadísticos complejos de una manera clara y concisa.
- Utiliza la inferencia bayesiana para estimar las distribuciones de probabilidad de los parámetros de un determinado modelo, pudiendo generar también intervalos de confianza.
- Es eficiente computacionalmente, ya que utiliza algoritmos MCMC optimizados, como el algoritmo HMC (Hamiltonian Monte Carlo).
- Es compatible con otros lenguajes de programación como R o Python.

En este lenguaje de programación, hay dos formas de introducir comentarios que sean ignorados en la ejecución del código. En este trabajo se utilizará que cualquier secuencia que siga a `//` o `#` se ignora hasta que se produce un cambio de línea. Por ejemplo,

```
# Esto es un comentario.
// Esto es un comentario.
```

Si nos centramos en la sintaxis propiamente dicha, Stan considerará como espacio en blanco cualquier espacio, tabulación, retorno de carro y salto de línea. Además la cantidad de espacios en blanco es irrelevante a la hora de ejecutar el código, ya que todos son tratados de manera idéntica.

Dentro de las expresiones de Stan, existen diferentes tipos de datos. En un programa Stan, cada variable debe tener el tipo de datos utilizado declarado con anterioridad y solo

se podrán asignar valores de ese tipo, salvo en algunas excepciones. Por lo tanto, esto es una de las principales diferencias con respecto a otros lenguajes de programación como R.

A continuación, se hace una pequeña descripción de los tipos de datos disponibles en Stan:

- **Datos primitivos:** hay dos tipos, `real` para valores continuos e `int` para valores enteros. De manera automática, se pueden transformar los valores enteros a reales, aunque al revés puede ocasionar problemas importantes, ya que no todo número real es entero. Por otro lado, si se está fuera de ciertos rangos, se tendrán 3 categorías extras: "no es un número", infinito positivo e infinito negativo.
- **Datos complejos:** es el tipo `complex`, en el que tanto parte real como imaginaria son de tipo `real`.
- **Datos de tipo vector y matriz:** se divide según sea matriz (`matrix`), vector columna (`vector`) o vector fila (`row_vector`).
- **Arrays:** cualquier tipo de dato se puede declarar como array añadiéndole delante `array[dimensiones]`.
- **Datos restringidos:** la declaración de variables puede incluir restricciones, como pueden ser límites inferiores o superiores. Para añadir este tipo de datos, en la declaración de las variables se añade `<restricción>` a continuación del tipo de dato con el que se está trabajando. Por ejemplo, `int<lower = 1>` para añadir un límite inferior a un valor entero.

De todos los mencionadas anteriormente, a la hora de implementar el programa para resolver el problema de estimación de los pesos al nacimiento con el que estamos trabajando, únicamente se trabajará con datos de tipo primitivo y con datos de tipo vector y matriz. Por lo tanto, de aquí en adelante nos centraremos en explicar con especial detalle

estos datos.

En un programa Stan, todas las variables (tipo de dato y nombre) utilizadas deben estar declaradas de forma explícita previamente. Además, si dicha variable tiene restricciones, será necesario introducirlas al declararlas, aunque en este caso no se utilizarán, por lo que no se entrará en más detalles. Por ejemplo,

```
int a  # Declara una variable entera de nombre a.
real b # Declara una variable real de nombre b.
vector[5] v # Declara un vector columna, v, con 5 componentes.
matrix[4,4] A # Declara una matriz, A, con 4 filas y 4 columnas.
```

Hay tres tipos de acciones que solo permiten objetos de tipo vector o matriz: las operaciones aritméticas de matrices, las funciones de álgebra lineal y los parámetros y resultados de funciones multivariadas. Además, hay ciertos comandos que permiten crear estos objetos de manera especial, como un vector ordenando de manera ascendente, una matriz de correlaciones o un factor de Cholesky.

De manera similar a la declaración de variables se puede usar esa misma instrucción para definir el valor que toma dicha variable. Para definir las matrices se deben usar dobles corchetes especificando los vectores fila que la componen. Por ejemplo,

```
int a = 10; # Le asocia a la variable a el valor 10.
real b = 3.5; # Le asocia a la variable b el valor 3.5.
vector[5] v  = [1,2,3,4,5]; # Declara y define el vector v.
matrix[2,3] B = [ [1,2,3], [4,5,6] ]; # Declara una matriz, B,
            # y define sus valores.
matrix[4,4] C = cholesky_decompose(A); # Declara la matriz C y la
            # define como la descomposión de Cholesky de la matriz A.
```

Para acceder a los elementos de un vector o una matriz se usan corchetes de manera similar a como se hace en lenguaje de programación C. Considerando las matrices

y vectores definidos con anterioridad, se puede acceder a sus elementos de la siguiente manera:

```
v[2] # 2, segundo elemento de v.
B[1] # [1,2,3], primera fila de B.
B[2][2] # 5, segundo elemento de la segunda fila de B.
```

Los nombres de las variables no pueden contener puntos, tienen que empezar con una letra y no pueden terminar por dos guiones bajos. Es importante recordar que Stan es un lenguaje "case sensitive", es decir, distingue entre minúsculas y mayúsculas (`A` es distinto de `a`). Además hay ciertos nombres de cadenas que no se pueden utilizar para dar nombre a las variables, porque están reservadas para uso interno de Stan. Por ejemplo,

```
real a # Se puede utilizar.
real a.b # No puede declararse una variable con ese nombre.
real a__ # No se puede utilizar.
```

En lenguaje Stan, los paréntesis sirven para agrupar, ya que cualquier expresión entre ellos es también una expresión. Además, se pueden hacer las operaciones aritméticas suma (`+`), resta (`-`), multiplicación (`*`), división (`/`), modulo (`%`) y exponenciación (`^`). Para matrices, se tiene también la transposición (`'`).

Como ocurre con otros lenguajes de programación, Stan tiene una serie de funciones matemáticas predeterminadas. Estas funciones aceptan un tipo concreto de argumentos y dan un tipo de resultado específico. El nombre de la función y sus argumentos la determinan de manera única. Las funciones que hacen referencia a constantes no tienen argumentos.

```
real c = pi(); # Le asocia a la variable c el número pi.
```

Las instrucciones de asignación son aquellas en las que se le proporciona un valor a una determinada variable. Para que dicha asignación sea válida es necesario que vaya seguida

de un punto y coma. También es posible hacer asignaciones y operaciones a través de la misma orden, siendo de forma general la instrucción `x = x op y` ó `x op = y`, donde `op` hace referencia al tipo de operación. Es decir,

```
a = a + b; # Le asocia el valor a+b a la variable a.
a + = b; # Lo mismo que la instrucción anterior.
```

Con Stan también es posible asignar distribuciones de probabilidad a las variables, esto no quiere decir que la variable tenga un valor concreto. En este caso, de lo que se informa es que una variable está distribuida como una distribución determinada, que viene dada por sus parámetros.

```
real y ~ normal(0,1) # La variable y se comporta como una normal de
        # media 0 y desviación típica 1.
```

Por otro lado, hay que tener en cuenta que la base de ejecución en Stan es la evaluación de una densidad de probabilidad logarítmica para un conjunto dado de parámetros. Es decir, Stan trabaja con el logaritmo de la función de densidad de los datos, lo que hace que todos los cálculos estén en esa escala logarítmica. Sin embargo, esto no afecta a la forma de introducir los datos por el usuario ni a resultados devueltos por el programa, ya que las estimaciones se dan en escala natural.

El lenguaje Stan también puede tener bucles, siendo los más comunes `if-then-else`, `for` y `while`. Al no ser necesarios para la resolución del problema planteado, no se tratan con más detalle, pero se podría decir que los bucles en Stan funcionan de manera similar a como lo hacen en R.

Un programa en Stan se puede dividir en bloques, cuyos cuerpos contienen declaraciones de variables e instrucciones, que se ejecutan en el orden en el cual están escritas. Estos bloques pueden ser de distintos tipos según la información que contengan, aunque no todos tienen que aparecer en un programa. En caso de que aparezcan, deberán estar bien delimitados y aparecer en el orden siguiente:

- **Bloque de definición de funciones:** contiene las funciones definidas por el usuario.

```
functions {
# Declaración de funciones.
# Definición de funciones.
}
```

- **Bloque de datos:** contiene la declaración de los datos necesarios en el modelo. Este bloque no permite instrucciones.

```
data{
# Declaración de variables.
}
```

- **Bloque de datos transformados:** contiene la definición de constantes y las transformaciones de datos. Solo se permiten asignaciones a las variables declaradas en este bloque.

```
transformed data {
# Declaración y definición de datos transformados.
}
```

- **Bloque de parámetros:** contiene la declaración de los parámetros del modelo, es decir, las variables que son muestreadas en los algoritmos de muestreo Stan. A los parámetros no se les puede asignar un valor.

```
parameters {
```

```
# Declaración de los parámetros del modelo.
}
```

- **Bloque de parámetros transformados**: contiene la definición de las variables en términos de datos y parámetros que se usarán en el modelo.

```
transformed parameters {
# Declaración y definición de variables que dependen de los
# parámetros.
}
```

- **Bloque de modelo**: contiene la definición de la función de probabilidad logarítmica. Las variables de este bloque no se pueden definir con restricciones porque no hay forma de que sean fáciles de validar.

```
model {
# Definición de las distribuciones de probabilidad.
}
```

- **Bloque de cantidades generadas:** permite obtener cantidades derivadas de los parámetros y datos. Es el más diferente a los demás, ya que nada de lo que ocurre en él afecta a los parámetros muestreados porque se ejecuta después de generar las muestra.

```
generated quantities { }
```

Las variables deben ser declaradas con anterioridad a utilizarse y son, en general, globales, se pueden usar en todos los bloques posteriores, salvo las del bloque de modelo, que

son locales. En el caso de las funciones, se definen en el bloque de funciones y se pueden usar en cualquier bloque posterior.

Las funciones definidas por el usuario aparecen en un bloque inicial independiente de los demás bloques del programa. Los nombres de las funciones siguen las mismas normas que se utilizaban en las variables. Hay que tener en cuenta que dos funciones pueden compartir nombre si el tipo de argumentos necesarios es distinto. Además, las funciones no tienen argumentos predeterminados, es necesario introducirlos todos.

Con todo esto se tiene una introducción básica a lo que es el lenguaje de programación Stan. Hay muchos aspectos que no se han mencionado, pero que también pueden ser útiles según el programa con el que se esté trabajando. Con lo descrito anteriormente será suficiente para la explicación del código utilizado para cumplir con el objetivo de este trabajo. En caso de necesidad de alguna explicación más profunda, se hará directamente basándose en el propio código.

Por otro lado, es importante tener en cuenta que, aunque Stan es un lenguaje de programación bastante completo, tiene ciertas limitaciones. Algunas de ellas pueden derivar directamente de la necesidad de tener ciertos conocimientos estadísticos previos o de problemas con la sintaxis. Por ejemplo, dentro del bloque de parámetros no sería posible la utilización de explesiones condicionales en la declaración de los mismos.

3.2. Implementación para R

Como se comentó al principio del capítulo, se trabajará con el paquete `rstan` de R [7]. Para utilizarlo será necesario aplicar el lenguaje Stan visto en la sección anterior, además de las funciones propias de R que sirven para tratar con datos de pedigrí y con métodos de resolución de los modelos lineales mixtos con cadenas de Márkov a través de simulaciones de Monte Carlo.

En este caso se trabaja con efecto fijo el sexo y efecto aleatorio el efecto genético, por lo que el objetivo principal es la obtención de las estimaciones para cada sexo y para cada individuo. En el caso de tener muchos individuos, localizar el que proporciona una mayor variación en el peso al nacimiento puede resultar muy costoso, por lo que, de momento, se tratará de localizar el sexo con mayor peso al nacimiento.

Para trabajar con este método hay que tener en cuenta que es necesario conocer el peso y el sexo de todos los individuos. Esto da lugar a un problema: al completar los datos y añadir los padres y madres que no están en la muestra, se desconocen todos los datos de dichos indiviudos. El sexo, aunque en un principio se considera como un valor perdido, es fácil de obtener, pero el peso al nacimiento no. Por tanto, para solucionarlo, el usuario debe imputar dichos pesos desconocidos a partir de los datos disponibles, es decir, considerar los pesos de la muestra y generar valores aleatorios a partir de la media y la desviación típica.

Por otro lado, el programa no trabaja con la variable sexo como tal, sino que crea dos variables para las cuales una de ellas toma el valor 1 cuando se trabaja con M (macho) y 0 en caso de H (hembra), y la otra al revés. Es decir, se trabajará con dos variables ficticias en lugar de con la variable sexo original.

Por último, es necesario calcular la matriz de parentescos ya conocida, aunque realmente se utilizará su descomposición de Cholesky. Con todo esto se tratará de abordar los dos casos vistos anteriormente para tratar de encontrar una resolución más rápida.

La función utilizada será `stan`, que permite compilar y ajustar modelos bayesianos utilizando el lenguaje Stan. Esta función permite introducir distintos argumentos como el código del modelo, los datos que se pretenden ajustar, el número de cadenas de Márkov, el número de iteraciones por cadena, los parámetro de interés, etc.

El código del modelo es donde aparece el lenguaje de programación Stan, ya que dicho código debe estar introducido con la estructura de bloques vista en la sección 3.1. Por lo tanto, aunque en este caso el bloque de definición de funciones no es necesario, se puede ver cómo se utilizan los demás de la manera indicada previamente.

A continuación se dará una explicación detallada del contenido de cada bloque en el programa de R utilizado para resolver los distintos problemas propuestos. Es importante tener en cuenta que, además de la parte escrita en Stan, el ejemplo se completa con los comandos de R explicados previamente y que permiten adaptar los datos a la estructura necesaria (ver Apéndice A.1).

Se comienza con el **bloque de datos**, donde se declaran las distintas variables del modelo. En este caso se tendrán el número de individuos, el vector de la variable respuesta, variables ficticias para el sexo, y la matriz de parentescos.

```
data {
    int N; # Número de individuos de la muestra.
    vector[N] y; # Vector N-dimensional con los pesos al nacimiento.
    vector[N] g1; # Vector N-dimensional referente a los machos.
    vector[N] g2; # Vector N-dimensional referente a las hembras.
    matrix[N, N] G; # Matriz de parentescos de dimensión NxN.
}
```

Posteriormente se tendrá el **bloque de datos transformados**, donde se declara y se define una matriz `C`, que será la descomposición de Cholesky de la matriz de parentesco.

```
transformed data {
    matrix[N, N] C;
    C = cholesky_decompose(G);
}
```

La descomposición de Cholesky es una forma de factorizar una matriz simétrica y definida positiva en el producto de una matriz triangular superior y su transpuesta. La matriz de parentescos cumple dichas condiciones por lo que se puede escribir como `A = C * C'`, donde `C` es la correspondiente matriz triangular. Cuando se tienen gran cantidad de individuos, la utilización de esta descomposición reducirá el coste computacional al aumentar el número de ceros respecto a la matriz original.

Además, la descomposición de Cholesky tiene una propiedad que reducirá los cálculos posteriormente. Si $G = CC'$ y X es una variable aleatoria normal ($X \backsim N(\mu, \Sigma)$), entonces $CX \backsim N(C\mu, C\Sigma C)$. Además, si se cumple que existe $\sigma \in \mathbb{R}$ tal que $\Sigma = \sigma I$, donde I es la matriz identidad del tamaño correspondiente, entonces $CX = N(C\mu, \sigma G)$.

A continuación aparece el **bloque de parámetros**, donde, como ya se comentó, aparecen declarados los parámetros que serán muestreados en el modelo.

```
parameters {
    vector[2] b; # Estimación de efectos fijos (uno por cada sexo).
    vector[N] z; # Valor mejorante (uno por cada individuo).
    real sigma_p2; # Varianza de los efectos aleatorios.
    real sigma_r2; # Varianza del error.
}
```

En el **bloque de parámetros transformados** se declaran y se definen los tres parámetros que formarán parte del modelo utilizado.

```
transformed parameters {
    real sigma_p; # Desviación típica de los efectos aleatorios.
    real sigma_r; # Desviación típica del error.
    vector[N] g; # Parte de la variable respuesta explicada por los
                 # efectos aleatorios.
    sigma_p = sqrt(sigma_p2); # Definición de la variable sigma_p.
```

```
    sigma_r = sqrt(sigma_r2); # Definición de la variable sigma_r.
    g = sigma_p * C * z; # Definición de la variable g.
}
```

Con esto ya podemos introducir el **bloque de modelo**, donde se definen las distribuciones de probabilidad de las variables. Cabe destacar que todas tienen parámetros arbitrarios, salvo la variable respuesta (`y`), cuyos parámetros dependen de los datos conocidos (recodando que `g1 + g2 = 1` con `g1, g2` $\in \{0,1\}$).

```
model {
    b ~ normal(0, 1000); # Distribución normal.
    sigma_p2 ~ inv_gamma(0.001, 0.001); # Distribución gamma inversa.
    sigma_r2 ~ inv_gamma(0.001, 0.001); # Distribución gamma inversa.
    z ~ normal(0, 1); # Distribución normal.
    y ~ normal(b[1] * g1 + b[2] * g2 + g, sigma_r); # Distribución
                                                    # normal.
}
```

En las distribuciones de las variables se trabaja con distribuciones conocidas como la normal, cuyos parámetros representan la media y la desviación típica. En el caso de la distribución gamma inversa, también consta de dos parámetros, donde el primero de ellos es considerado parámetro de forma (controla la altura de la función de densidad) y el segundo se llama parámetro de escala (controla la dispersión). Según se demostró en [1], los parámetros de las distribuciones de las varianzas afectan poco a los resultados que se obtendrán, en ese caso se trabajaba con la distribucion χ-cuadrada inversa escalada, pero esta no deja de ser un caso especial de la distribución gamma inversa.

Por último, se tiene el **bloque de cantidades generadas** donde se declaran y definen tres variables de tipo `real` que harán referencia a ciertas cantidades que pueden resultar interesantes a la hora de resolver el problema.

```
generated quantities {
```

```
    real h2; # Heredabilidad.
    real p; # Variabilidad de efectos aleatorios (genética).
    real r; # Variabilidad del error.
    p = sigma_p2;
    r = sigma_r2;
    h2 = p / (p + r); # Heredabilidad.
}
```

Con todos los datos, parámetros y distribuciones vistas en los distintos bloques del codigo en Stan, se tiene el código del modelo necesario para trabajar con la función `stan`. Los demás argumentos de esta función cambiarán según las necesidades. En general, trataremos de muestrear los variables definidas por `b`, `p`, `z`, `r` y `h2`, aunque nos centraremos en los resultados proporcionados por `b` y `z`, que hacen referencia a las estimaciones.

Como se comentó anteriormente, se trabajará con la función `stan`, que permite ajustar datos a modelos estadísticos usando un método de inferencia bayesiana basado en el muestreo de HMC (Hamiltonian Monte Carlo). Esta función dispone de diversos argumentos de entrada que se modificarán según el objetivo buscado y los datos disponibles.

- `model_code`: código del modelo escrito en Stan, es decir, el sistema de bloques explicado previamente.
- `data`: datos utilizados en el modelo.
- `chains`: número de cadenas de Márkov Monte Carlo que se utilizarán en paralelo. Por defecto es 4.
- `iter`: número de iteraciones que se realizarán en cada cadena. Por defecto es 2000.
- `warmup`: número de iteraciones de calentamiento antes de almacenar las muestras, es decir, número de iteraciones que se desprecian a la hora de proporcionar un resultado. Por defecto es la mitad de `iter`.

La salida que proporciona esta función es un objeto de la clase "stanfit" con las estimaciones de los parámetros. Sobre este objeto se pueden usar diversos comandos para acceder a los datos, como puede ser `summary`, `plot` o `traceplot`.

El muestreo se hace a través del algoritmo HMC que se utiliza para estudiar la distribución a posteriori de un modelo bayesiano a partir del algoritmo Metropolis. Este algoritmo se basa en dos ideas: la primera es considerar una variable auxiliar que representará la velocidad de una partícula ficticia en el espacio de parámetros del modelo y cuya posición siguiente tiene distribución normal centrada en la posición actual; la segunda es simular la trayectoria de dicha partícula por el método "leapfrog integrator". Se podría resumir este algoritmo en los siguientes pasos:

1. Se inicializa el estado actual de los parámetros con un valor arbitrario.
2. Se genera un valor aleatorio de la variable auxiliar a partir de la distribución normal mencionada anteriormente.
3. Con un tamaño de paso dado, se simula la trayectoria de la partícula.
4. Se acepta o rechaza el estado final usando el criterio de Metropolis-Hastings [8].
5. Si se acepta, se actualiza el estado actual. Si se rechaza, se mantiene el estado.

Por tanto, a partir de lo anterior, ya se está en disposición de resolver los dos casos que se habían comentado anteriormente, así como de añadir casos nuevos que serán factibles de realizar con este paquete. En la sección posterior se abordarán dichos casos y se tratará de interpretar los resultados obtenidos, teniendo en cuenta los posibles problemas que puedan surgir.

3.3. Resultados e interpretación

En este apartado se estudiarán, con ayuda del programa explicado en la sección 3.2, los resultados proporcionados para los dos casos sencillos desarrollados anteriormente, asi

como las estimaciones para casos más complejos.

En todos los ejemplos que se proponen, se utilizan diferentes métodos para imputar los pesos al nacimiento que son desconocidos. El objetivo de esto es encontrar cuál es el efecto de dichas imputaciones en el resultado final del estudio.

Caso 1

El primero de los casos consiste en estudiar el efecto del sexo (y el individuo) considerando únicamente los 10 padres y 10 madres con mayor número de hijos. Como ya se vio en la sección 2.2, se tienen 58 individuos pero solo se conoce el peso al nacimiento de 4 de ellos.

Con el código del Apéndice A.1, es posible estimar el efecto del sexo y del individuo, pero hay que tener en cuenta que se están generando 54 pesos al nacimiento a partir de los 4 conocidos, por lo que, dependiendo de esos 54 valores aleatorios, los resultados pueden cambiar. La imputación de los pesos desconocidos se hará considerando valores aleatorios de una normal con media y desviación típica las correspondientes a los datos conocidos.

En este caso, se obtienen los siguientes resultados para 100 000 iteraciones del algoritmo y considerando una única cadena de Márkov:

$$\vec{\beta} = \begin{pmatrix} 33.530 \text{ kg} \\ 34.340 \text{ kg} \end{pmatrix}.$$

Por tanto, con este modelo se obtiene que el sexo que tiene mayor peso al nacimiento es el segundo, es decir, las hembras. Como se comentó antes, este resultado se puede ver muy influenciado por los valores asignados a los pesos desconocidos.

Es factible llevar a cabo una imputación de los pesos basada en valores aleatorios extraídos de una distribución normal, utilizando tanto la media como la desviación estándar

de la población completa en lugar de depender exclusivamente de los datos de los 4 individuos conocidos de la muestra. Con estos nuevos datos, las estimaciones relacionadas con la influencia del sexo experimentan modificaciones, sin embargo, siguen dependiendo de los pesos al nacimiento considerados. El sexo con mayor peso al nacimiento sigue siendo el mismo, es decir, las hembras.

$$\vec{\beta} = \begin{pmatrix} 36.923 \text{ kg} \\ 37.730 \text{ kg} \end{pmatrix}.$$

Faltaría ver dos casos un poco más concretos, es decir, considerar qué pasaría si las imputaciones se hiciesen teniendo en cuenta también el sexo. En el primer caso, imputamos los pesos al nacimiento con valores aleatorios de una normal que tenga parámetros provenientes de los individuos de la muestra, hembras o machos, según el sexo del individuo considerado.

$$\vec{\beta} = \begin{pmatrix} 30.885 \text{ kg} \\ 34.734 \text{ kg} \end{pmatrix}.$$

Como en casos anteriores, se observa que las hembras parecen tener mayor peso al nacimiento que los machos. Sin embargo, se pueden comparar estos resultados con los que utilizan imputaciones por sexos pero basadas en todos los datos de los individuos, estén o no en la muestra.

$$\vec{\beta} = \begin{pmatrix} 41.057 \text{ kg} \\ 36.884 \text{ kg} \end{pmatrix}.$$

En este caso ya se observa como los machos tendrían un mayor peso al nacimiento que las hembras; esto concuerda con el análisis descriptivo que se estudió anteriormente, pero se está trabajando con más datos de los estrictamente necesarios, aunque parece indispensable para que la estimación tenga sentido.

A priori, lo más correcto sería considerar únicamente los datos de la muestra considerada, ya que son los que tenemos la seguridad de conocer independientemente del problema en el que trabajemos. Sin embargo, estamos utilizando un enfoque bayesiano, lo que permite disponer de esa información previa. Además, podemos comprobar como el

resultado concuerda con lo estudiado en el programa propio desarrollado para seleccionar el sexo con mayor peso al nacimiento, aunque los valores numéricos de las estimaciones no coincidan exactamente.

Caso 2

Cuando se estudia el otro caso, es decir, los hijos del padre con mayor número de hijos, se empiezan a ver problemas a la hora de realizar un número suficiente de iteraciones, ya que se dispone de 8010 individuos, de los que conocemos 1420 pesos al nacimiento. Como se vio anteriormente, la primera imputación de los pesos al nacimiento desconocidos podría ser aquella dada por valores aleatorios de una normal con parámetros procedentes de la muestra considerada.

Teniendo en cuenta la cantidad de datos de los que se dispone, el número de iteraciones a realizar no puede ser tan elevado como en el caso 1, ya que el tiempo de ejecución sería demasiado alto. De todas formas, considerando únicamente 60 iteraciones del algoritmo para una cadena Márkov, se obtienen resultados estables y fiables:

$$\vec{\beta} = \begin{pmatrix} 36.203 \text{ kg} \\ 35.723 \text{ kg} \end{pmatrix}.$$

Esto nos está indicando que se espera un mayor peso al nacimiento para los machos, lo cual coincide con lo visto en el análisis descriptivo estudiado previamente. Pero, como vimos en el caso anterior, nada nos asegura que la forma de generar los pesos desconocidos sea la correcta, así que probamos con otros métodos para comprobar si afectan o no esos valores aleatorios.

La siguiente opción para imputar los 6590 pesos al nacimiento desconocidos que haremos será considerar una normal con parámetros que provienen de los pesos de todos los individuos del conjunto de datos. Teniendo en cuenta esto y con el mismo número de

iteraciones (60), se obtienen los siguientes resultados:

$$\vec{\beta} = \begin{pmatrix} 38.145 \text{ kg} \\ 37.846 \text{ kg} \end{pmatrix}.$$

Los resultados obtenidos son similares a nivel cualitativo, ya que estima mayor peso al nacimiento a los machos. Sin embargo, se observa como los valores se ven modificados, algo esperable, teniendo en cuenta que los parámetros de las distribuciones con los que se imputan los pesos al nacimiento son distintos a los que se utilizaban anteriormente.

Nos faltaría comprobar que los resultados son similares cuando, para imputar los pesos desconocidos, se utiliza también la información referente al sexo de los individuos. Primero, se trabaja únicamente con los pesos de los individuos correspondientes a la muestra seleccionada, es decir, los que provienen de los 1420 pesos conocidos. Así se tienen las siguientes estimaciones para el peso al nacimiento según el sexo:

$$\vec{\beta} = \begin{pmatrix} 36.777 \text{ kg} \\ 35.066 \text{ kg} \end{pmatrix}.$$

Como vimos anteriormente, es importante comprobar las diferencias que pueden aparecer cuando los pesos se obtienen teniendo en cuenta toda la población. Es decir, todos los individuos de la base de datos, con los que se dan los siguiente resultados:

$$\vec{\beta} = \begin{pmatrix} 38.747 \text{ kg} \\ 36.139 \text{ kg} \end{pmatrix}.$$

Comprobamos, con estos últimos valores, que las estimaciones a nivel cualitativo son iguales. Es decir, todas las opciones van a predecir mayor peso al nacimiento en el caso de los machos. Sin embargo, los valores cambian en función del método utilizado, lo cual puede dar lugar a problemas en casos donde los resultados de machos y hembras toman valores similares.

Por tanto, a raíz de los resultados vistos para las cuatro opciones de imputación de pesos desconocidos, se puede concluir que el peso al nacimiento será mayor en machos que

en hembras. Esto coincide con lo obtenido a través del análisis descriptivo. Sin embargo, no todos los métodos proporcionan el mismo resultado numérico, lo que evidencia la influencia de esas imputaciones en el resultado final. Esto puede dar lugar a problemas en casos donde ambos sexos tenga pesos parecidos, o el comportamiento de los datos de la muestra no sea representativo del conjunto de datos completos.

Caso 3

Se considera un ejemplo nuevo; en este caso, se busca predecir el efecto del sexo cuando se completa el conjunto de individuos del Caso 2 añadiéndole nuevos individuos. Se tenían los 8009 hijos del individuo con id=60 205 y el padre. En este caso, se buscan completarlo con las madres de dichos individuos. Por tanto, se dispone de 14 578 individuos con los cuales pretendemos estudiar el efecto del sexo en el peso al nacimiento. Hay que tener en cuenta que se desconoce el peso al nacimiento para 12 616 individuos, lo que hace que se tengan que generar los pesos al nacimiento con los 1962 restantes.

Como en los casos anteriores, es conveniente trabajar con distintos métodos para completar los pesos al nacimiento y poder estudiar el efecto del sexo. Es importante darse cuenta de que el número de individuos es casi el doble que el del Caso 2, lo que requerirá un tiempo, a priori, superior a la hora de ejecutar el algoritmo y obtener soluciones. Por tanto, se probará a trabajar únicamente con 40 iteraciones, las mínimas recomendadas por el manual de `rstan` para obtener resultados fiables.

Como cabía esperar, este aumento tan grande en el número de individuos a tratar hace imposible ejecutar el algoritmo. Esto se debe a que, tras varias horas de espera, apenas se habían realizado el 20 % de las iteraciones. Por lo tanto, no se puede abordar el ejemplo, ya que realizarlo completo, repitiéndolo un número adecuado de veces para que se vea la estabilidad de los resultados, podría llevar varias semanas.

Por tanto, el paquete `rstan` proporciona estimaciones que parecen ir acordes al análisis descriptivo. En esta implementación se tienen en cuenta las distribuciones a priori de los parámetros, pero es necesario imputar los valores desconocidos de la variable respuesta, lo cual parece afectar al resultado final. Además, se puede observar un problema importante en el tiempo de ejecución, ya que al aumentar el número de individuos los tiempos aumentan considerablemente, haciendo los problemas difícilmente abarcables.

Capítulo 4

Librería *MCMCglmm*

El paquete `MCMCglmm` de R [9] es una librería que permite ajustar modelos lineales mixtos utilizando cadenas de Márkov Monte Carlo (MCMC) para la estimación de parámetros y la inferencia bayesiana. El paquete, al igual que lo hacía `rstan`, utiliza algoritmos MCMC para aproximar la distribución a posteriori del modelo, lo que permite la inclusión de efectos aleatorios y la especificación de modelos jerárquicos y complejos.

El paquete es especialmente útil cuando se trabaja con grandes conjuntos de datos, modelos no lineales o cuando se desea obtener una descripción completa de la distribución a posteriori de los parámetros del modelo. Algunas de las características de la librería `MCMCglmm` incluyen:

- Resolución de modelos lineales mixtos con efectos fijos y aleatorios, incluyendo la posibilidad de incluir estructuras de correlación entre efectos aleatorios.
- Posibilidad de especificar distribuciones previas no informativas o informativas para los parámetros del modelo.
- Inclusión de diagnósticos para verificar la convergencia de las cadenas de Márkov y para evaluar la calidad de los resultados del modelo.

- Posibilidad de realizar simulaciones de Monte Carlo y generar intervalos de confianza bayesianos para los parámetros del modelo.

Comparando estas características se observan varias similitudes, pero también algunas diferencias, entre la librería `rstan` y la librería `MCMCglmm`. Por un lado, las estructuras de correlación entre efectos aleatorios son tratadas de manera diferente. Mientras que `MCMCglmm` lo considera como una posibilidad dentro de su propio programa y hay que proporcionarle únicamente la matriz de covarianzas, para `rstan` es necesario especificarla en el propio modelo. De manera similar, las distribuciones a priori de `MCMCglmm` son limitadas, pudiendo modificarse los parámetros. Sin embargo, en el modelo de `rstan` hay libertad para especificar la distribución a priori que más convenga.

Por otro lado, en cuanto a los diagnósticos de convergencia y herramientas para evaluar la calidad, aunque ambas librerías las proporcionan, son más completas y detallas en `rstan`. En la última de las características ocurre algo similar, ambos paquetes son capaces de realizar las simulaciones de Monte Carlo y generar intervalos de confianza, pero `rstan` proporciona mayor flexibilidad para adaptarse a modelos complejos.

4.1. Implementación para R

De este paquete nos centraremos en la función que lleva su mismo nombre, `MCMCglmm`, que permitirá resolver el modelo mixto con cadenas de Márkov Monte Carlo. Para utilizarla es necesario disponer de las relaciones entre los individuos, del efecto fijo y de los valores conocidos de la variable respuesta.

La función `MCMCglmm` tiene diversos parámetros de entrada que permiten personalizar el análisis según el problema que se busca resolver [10]:

- `fixed`: fórmula que indica la variable respuesta y el efecto fijo correspondiente (se podrían tener varias variables respuesta y varios efectos fijos).

- `random`: fórmula que indica el efecto aleatorio (o los efectos aleatorios).
- `data`: conjunto de datos.
- `nitt`: número de iteraciones de la cadena de Márkov.
- `prior`: distribuciones a priori, divididas en tres categorías:
    - `R (R-structure)`: distribución correspondiente a los residuos, parámetros de una distribución inversa de Wishart (generalización de χ-cuadrado a varias dimensiones) denominados `nu` (grados de libertad) y `V` (matriz de escala).
    - `G (G-structure)`: distribución (o distribuciones) correspondiente a los efectos aleatorios, parámetros de una distribución inversa de Wishart denominados `nu` (grados de libertad) y `V` (matriz de escala).
    - `B (fixed effects)`: distribución correspondiente a los efectos fijos, parámetros de una distribución normal multivariada denominados `mu` (media) y `V` (covarianzas).
- `ginverse`: inversa de la matriz de parentescos, tiene que ser una matriz rala. El nombre de la matriz debe coincidir con el efecto aleatorio considerado. Todos los niveles del efecto aleatorio deben aparacer en los nombres de las filas de la matriz de parentescos inversa.

Con todo esto, ya se puede aplicar la función a los diferentes casos que se están estudiando. Hay que tener en cuenta que, al igual que ocurría en el caso anterior, hay que adaptar los datos a la función, ya que estos deben estar ordenados de manera adecuada.

4.2. Resultados e interpretación

Caso 1

Aplicando al primero de los casos que estamos considerando, los 10 padres y 10 madres con mayor número de hijos, se puede obtener una estimación del efecto del sexo (código

en el Apéndice A.2). Por similitud a los métodos vistos anteriormente, se llevan a cabo 100 000 iteraciones de la cadena de Márkov y se obtienen las siguientes estimaciones del efecto del sexo:

$$\vec{\beta} = \begin{pmatrix} 31.160 \text{ kg} \\ 35.100 \text{ kg} \end{pmatrix}.$$

Por tanto, se observa que el sexo con mayor peso al nacimiento son las hembras. Esto, como ya se había comentado, es lógico, pues se dispone únicamente de 4 datos de pesos, 3 de ellos correponden a hembras y su media es bastante superior a la de los machos en este conjunto de datos.

En relación con el paquete `rstan` explicado anteriormente se pueden observar divesas relaciones. En este caso no es necesario introducir el peso al nacimiento en los individuos en los que se desconoce, por lo que se está quitando la posibilidad que los resultados se vean afectados por nuestra elección de las imputaciones. Por otro lado, el tiempo de ejecución es menor y parecen obtener resultados igual de fiables.

Por otro lado, estos dos paquetes también tienen ciertas similitudes; para empezar, los individuos tiene que estar ordenados para poder calcular las matrices correctamente. Además, en ambos se pueden introducir las distribuciones a priori de los efectos, tanto fijos como aleatorios.

Caso 2

Con este paquete se pueden obtener las estimaciones del sexo también para el otro caso que estamos estudiando, es decir, los hijos del padre con mayor número de hijos. Para hacer dichas estimaciones se hacen 100 000 iteraciones, lo que proporciona el siguiente efecto del sexo:

$$\vec{\beta} = \begin{pmatrix} 38.040 \text{ kg} \\ 36.079 \text{ kg} \end{pmatrix}.$$

Como se puede observar, el sexo con mayor peso al nacimiento parece que cambia. En este caso son los machos, cosa que según el análisis descriptivo parece tener cierto sentido. Además, a diferencia de lo que ocurría con otros algoritmos, se han conseguido realizar un número de iteraciones elevado en un tiempo razonable, por lo que podemos suponer el resultado como fiable.

Por tanto, este algoritmo proporciona un número suficientemente grande de iteraciones aunque crezca el número de individuos. Esto hace que se puedan obtener resultados fiables sin la necesidad de tanto tiempo de ejecución, ya que en este caso apenas se tarda unos minutos en ejecutar el algoritmo. Es decir, la función proporcionada por el paquete `MCMCglmm` ejecuta el algoritmo con mayor rapidez que la correspondiente al paquete `rstan`.

Caso 3

De la misma manera que se hizo con el paquete `rstan`, se busca trabajar con el conjunto del Caso 2 añadiendo las madres de los individuos. Es decir, se trabaja con 14 578. Mientras que en la implementación anterior no era posible obtener resultados debido al tiempo de ejecución, en este caso se pueden realizar las 100 000 iteraciones de la cadena de Márkov como en los dos primeros.

Por tanto, con estos datos, se obtienen las siguientes estimaciones del efecto del sexo:

$$\vec{\beta} = \begin{pmatrix} 40.876 \text{ kg} \\ 38.921 \text{ kg} \end{pmatrix}.$$

Como en el caso descriptivo, se determina que los machos tienen mayor peso al nacimiento que las hembras. En este caso, se está trabajando con el 1.5 % de los datos totales de los que se dispone y, aunque el algoritmo tarda en ejecutarse varios minutos más que en el Caso 2, sigue siendo un tiempo razonable y que posibilita repetirlo varias veces para comprobar la estabilidad del resultado.

Por tanto, después de haber ejecutado los tres ejemplos con ambos paquetes, se observa una gran diferencia entre ellos. Esta es la relacionada con el tiempo de ejecución, siendo el segundo mucho más rápido. Lo que con `rstan` eran varios días, con `MCMCglmm` son apenas unas horas.

Aunque una posibilidad sería buscar ejemplos con mayor número de individuos, esto sí que aumentaría el tiempo de cálculo de la matriz de parentescos. En el Caso 3, gran parte del tiempo empleado en ejecutar el algoritmo se debe, precisamente, al cálculo de la matriz, lo que supondría un tiempo elevado si se aumenta el número de individuos e imposibilita llegar a trabajar con todos los animales disponibles. Por ejemplo, para casi 50 000 individuos el cálculo de la matriz ya requería aproximadamente 3 días de ejecución.

Capítulo 5

Programas en C y en Fortran

Fortran, acrónimo de "Formula Translating", es un lenguaje de programación de alto nivel que fue desarrollado por IBM en la década de 1950. Fue diseñado para facilitar la programación de cálculos numéricos y aplicaciones científicas y de ingeniería. Entre las ventajas que presenta este lenguaje se encuentran: la eficiencia en cálculos numéricos (alta velocidad y precisión), la facilidad de aprendizaje, la compatibilidad y portabilidad, y la facilidad para codificar programas que manejan matrices.

Hay numerosos programas en Fortran que se utilizan para la evaluación genética (por ejemplo, los desarrollados en [12] y [13], aunque únicamente trabajaremos con el primero de ellos). Debido a las limitaciones derivadas del uso de R para resolver el problema central de este trabajo, se busca en este lenguaje una forma más rápida y completa de abordarlo. Una de las primeras medidas que se tomaron para mejorar la ejecución fue la de trabajar con un ordenador de 128GB, en lugar de los 12GB de memoria RAM del ordenador en el que se ejecutaron los programas en R. Con esto se pretende aumentar el número de individuos que se introducen al modelo, pudiendo llegar a todos los disponibles.

Se ejecutarán dos programas diferentes, uno de ellos programado en C [11] y el otro en Fortran [12], cada uno de los cuales tiene sus peculiaridades a la hora de introducir datos

o parámetros. Se buscará aplicar cada uno de ellos a todos los datos disponibles, es decir, a los 931 689 individuos. Se explicará con detalle el formato de cada uno de los archivos requeridos, asi como la solución que proporciona, en caso de que sea posible ejecutarlo.

5.1. loki prep de Simon C. Heath

Este paquete fue desarrollado por Simon C. Heath, quien implementó análisis de segregación y vinculación en predigrís grandes y complejos. Para ello utilizó cadenas de Márkov con simulaciones de Monte Carlo con salto reversible, es decir, la cadena puede saltar entre modelos con distinto número de parámetros. Se centrará la ejecución en dos comandos.

Por un lado, se tiene el comando `prep`, cuya utilidad radica en la preparación previa de los datos. Para ello, es necesario crear un fichero de texto en el que se especifiquen los datos disponibles. Lo primero es obtener el fichero de datos completo, guardándolo en formato "csv". Esta operación se realiza directamente desde R, a partir del conjunto de datos completo, sin duplicados, utilizado para los ejemplos de capítulos anteriores.

El fichero de texto que necesita el comando `prep` como argumento, y que llamaremos *parametros.prep*, especificará los diferentes parámetros del modelo con la siguiente estructura:

```
file[FS=";", skip=1] "avanac.csv2" ignorable,nombre,id,pa,ma,pesonac,sexo
pedigree id,pa,ma
missing "NA"
sex sexo "M","H"
model pesonac=id+sexo
```

Lo primero es identificar el fichero de datos, con el separador correspondiente (en este caso " ; ") y el nombre de cada variable. La primera se identifica con "ignorable" debido a que es el número de cada fila, así que no aporta información. Hay que tener especial

cuidado con el formato de las variables, comprobando que efectivamente los indicadores de individuos aparecen como variables numéricas.

Posteriormente se identifican las variables que forman el pedigrí, en este caso, el identificador del individuo, su padre y su madre. A continuación, se establece la denominación que tendrán los valores desconocidos. Por último, se especifica el efecto fijo (el sexo del animal), así como sus niveles, y se escribe la fórmula que caracteriza el modelo (la variable respuesta es el peso al nacimiento, con efecto aleatorio la componente genética y efecto fijo el sexo).

Una vez se dispone del fichero adecuado, se ejecuta el comando correspondiente:

```
prep parametros.prep
```

Como ya se comentó, con esta orden se preparan los datos, identificándose casos de consanguinidad donde los hijos tiene padre y madre con algún tipo de parentesco cercano. Además, calcula la matriz inversa de la matriz de parentescos en un tiempo aceptable, de aproximadamente 1 minuto, para 931 689 animales.

A continuación, es necesario crear un archivo especificando los parámetros del modelo, al cual llamamos *parametros.loki* y tiene la siguiente estructura.

```
ITERATIONS 10000
START OUTPUT 50
OUTPUT FREQUENCY 1
OUTPUT FILE "loki.out1"
OUTPUT id,sexo
OUTPUT POLYGENIC "loki.out2"
```

En este archivo se identifican diferentes elementos del modelo, comenzando por el número de iteraciones a realizar (en este caso, 10 000) y cuántas de ellas se realizan de

calentamiento. Posteriormente, se identifica la frecuencia con la que se conservan las iteraciones, en este caso se mantienen todas, y se le da nombre al archivo de salida. Por último, se determinan los efectos a estudiar y el nombre del archivo en el que se guardan los valores mejorantes.

Para concluir, se debe implementar el modelo con los parámetros correspondientes, para ello se ejecuta la siguiente orden:

```
loki parametros.loki
```

Tras varios minutos de espera, se produce un mensaje de error, este mensaje indica que la memoria disponible en el ordenador (128 GB) no es suficiente para tratar con la cantidad de datos de la que se dispone (931 689 individuos). Es decir, con este primer programa no es posible resolver el problema propuesto para todos los individuos.

5.2. TM de Andrés Legarra

Andrés Legarra desarrolló un método en el lenguaje Fortran90 para la estimación de componentes de varianza (σ_u^2, σ_e^2), méritos genéticos ($\vec{u}$) y efectos fijos ($\vec{\beta}$) en distintos modelos, como los modelos lineales que se buscan resolver en este trabajo. Se basa en métodos MCMC y muestreo de Gibbs. El programa hace diversos cálculos, destacando:

- Distribuciones a posteriori para las componentes de la varianza y ratios relevantes como la heredabilidad o las correlaciones.
- Distribuciones a posteriori de méritos genéticos y efectos fijos con componentes de la varianza conocidos o desconocidos.

Además, el programa va a permitir trabajar con diferentes tipos de variables y modelos, incluyendo:

- Cualquier número de caracteres continuos, es decir, cualquier número de variables respuesta.

- Varios caracteres continuos, varios caracteres continuos ordinales y un carácter binario.
- Varios caracteres binarios, que podrían tener alguna restricción.
- Valores faltantes.
- Modelos animal y modelo macho.
- Varios efectos ambientales aleatorios.
- Variables explicativas nominales y variables explicativas continuas.

Sin embargo, cabe destacar que también tiene algunas limitaciones, ya que no puede utilizar ciertos elementos como covariables anidadas, aunque no se entrará en más detalle debido a que nuestro problema no requiere de su utilización.

La implementación se realiza con matrices y una estructura de lista dinámica enlazada (listas en las que cada elemento de la lista está relacionado con el siguiente). Por lo tanto, no es necesario volver a compilar el software para nuevos problemas. No hay límite de tamaño, pero para 100 000 incógnitas (cada uno de los valores mejorantes a estimar) se empieza a observar cierta lentitud.

De manera similar a lo visto con `loki`, para poder ejecutarlo, es necesario contar con diversos archivos, cuya estructura es muy específica y esencial para el correcto funcionamiento del modelo. Sin embargo, veremos como `tm` es menos flexible de lo que lo era `loki`. A continuación, se detallan cada uno de ellos, explicado sus componentes.

El primero será el archivo de **pedigrí**, que contiene los individuos ordenados numéricamente. Estará compuesto por tres columnas: el animal (id), el padre (pa) y la madre (ma). Además, los identificadores deben estar recodificados, de manera que tengan valores desde el 1 hasta el número de individuos disponibles (932 711, la cantidad de individuos

es superior a la inicial porque se han añadido aquellos padres y madres que no estaban en el conjunto de datos). Los individuos desconocidos deberán aparecer con el número correspondiente a su grupo genético que, como estamos considerando únicamente uno, será el número de individuos más 1 (932 712). Para crear este archivo se trabajó con R para renumerar y, posteriormente, se guardó como *pedigri.csv*. El código correspondiente esta disponible en el Apédice A.3.1.

El segundo será el archivo de **datos**, en el cual se especifican las variables que intevienen en el modelo y que se llamará *datos.csv*. Las columnas deben estar separadas por espacios y en el siguiente orden:

- Variables cuantitativas explicativas continuas.
- Efectos fijos, variables categóricas nominales.
- Efectos ambientales aleatorios.
- Efectos genéticos.
- Caracteres continuos.
- Caracteres polítomos (carateres cualitativos ordinales).
- Caracteres dicótomos.

Obligatoriamente, el modelo debe contener un efecto genético y un carácter. La variables cualitativas están codificadas por números (1,2,3,...), incluyendo los valores desconocidos, que supondrán un nivel más (por ejemplo: 1 = hembra, 2 = macho, 3 = NA). Sin embargo, en el caso de los rasgos continuos, los valores desconocidos tomarán el valor 0.

Por tanto, a la vista de lo mencionado en los párrafos anteriores y, teniendo en cuenta las estructura de nuestro modelo, el archivo de datos deberá tener 3 columnas. La primera de las columnas será el *sexo*, ya que se trata del efecto fijo. Posteriormente, se tendrá la

columna de *id* debido a que es un efecto genético. Y, por último, el *pesonac*, porque el peso al nacimiento es un carácter continuo (ver Apéndice A.3.1).

El tercero, y último, de los archivos es el de **parámetros** (*parametros.tm*), que será el que se pasará al programa de `tm`. En él se combinan títulos y comentarios con órdenes, algunas de las cuales hacen referencia a los dos archivos anteriores (pedigrí y datos). Este archivo tiene una estructura determinada para que el programa funcione correctamente:

```
Data file
datos.csv
Pedigree file
pedigri.csv
Model
animal
2 Number of effects
0 Number of covariates
1 Number of genetic groups
1 Number of traits
0 Number of threshold traits
0 Categories for threshold traits
0 Number of animal effects
3 932711 Levels for each effect
1 1 Model
Task
VCE
Total number of iterations
100000
Burn-in
30000
```

```
Thin interval
1
Genetic variance
1
Permanent
Residual
1
```

Dentro de este conjunto de instrucciones se comienza identificando el nombre del archivo de datos (*datos.csv*) y de pedigrí (*pedigri.csv*), así como el tipo de modelo que se pretende implementar, en este caso, modelo animal (modelo que permite descomponer la variación fenotípica según sus componentes genéticas). A continuación se deteminan el número y tipo de las variables: 2 efectos (*id*, *sexo*), ninguna covariables, un grupo genético, un rasgo continuo (*pesonac*) y un efecto animal (*id*). Posteriormente, de cada uno de los efectos se determina el número de niveles, siendo 3 para el *sexo* y 932 711 para el *id* (al ser un efecto genético, ya considera que los individuos con *id* superior a 932 711 son los valores desconocidos). Además, se consideran los dos efectos en el mismo modelo (`1 1 Model`; para no considerar alguno, se pondría un `0` en el lugar correspondiente).

En la segunda parte, se especifican aspectos concretos del modelo, como la forma de estimar los parámetros (VCE, significa que se estiman componentes de la varianza en vez de usar estimaciones fijadas) o el número de iteraciones a realizar (100 000, de las cuales 30 000 son de calentamiento) y cada cuánto se van a considerar resultados (en este caso, en todos las iteraciones, 1). Por último, se da un valor inicial a la varianza genética (σ_u^2) y a la varianza de los residuos (σ_e^2), que será 1 para ambas.

Una vez que se dispone de los 3 archivos necesarios, se puede ejecutar el programa `tm` (para evitar que solicite el nombre del fichero de parámetros se introduce mediante la función `echo` y se utiliza un tubo "`|`").

```
echo parametros.tm | ./tm
```

Cabe destacar que el archivo que contiene el modelo también se puede editar para que muestre por pantalla el progreso en la carga de datos y en las iteraciones con mayor o menor frecuencia, pero esto no afectará al resultado obtenido. Además, hay que tener especial cuidado con posibles problemas en el pedigrí y los datos, causados por la forma de guardar las potencias de 10 por R mediante notación exponencial (por ejemplo, $1.2e-8$), que no son admitidas por `tm`.

Una vez ejecutado el programa, el tiempo de ejecución que necesita es de aproximadamente un segundo por iteración, a lo que hay que añadirle un tiempo previo para la carga del pedigri (`Pedigree file`) y de los datos (`Data file`). Por tanto, para obtener resultados, se require de más de un día, lo que puede parecer excesivo, pero hay que tener en cuenta que se está trabajando con casi 1 millón de individuos.

Transcurrido ese tiempo, el programa proporciona tres archivos (*solutions.txt*, *results.txt* y *samples.txt*), cada uno de los cuales aporta información diversa sobre la ejecución y el resultado del problema considerado. A continuación, se detallan y analizan cada uno de ellos por separado.

solutions.txt

El primero de los ficheros de texto que proporciona el programa es el de soluciones. Aquí se tiene el efecto de los niveles de *sexo* y los méritos genéticos (valores mejorantes). Es decir, para cada nivel del sexo se obtiene el valor de β del modelo, y para cada individuo el correspondiente valor de u. Además, para cada estimación también proporciona el error típico.

Centrándonos primero en el efecto del sexo, es conveniente recordar que era una variable factor con tres niveles. Las hembras tomaban valor 1, los machos valor 2 y los valores

desconocidos eran el 3. Por tanto, se obtiene un vector $\vec{\beta}$ con tres componentes, una por cada nivel.

$$\vec{\beta} = \begin{pmatrix} 37.818 \text{ kg} \\ 40.270 \text{ kg} \\ 43.746 \text{ kg} \end{pmatrix}.$$

En este caso, puesto que el último son los valores desconocidos, solo nos interesa estudiar las dos primeras estimaciones (hembras y machos, respectivamente). Se observa como el peso al nacimiento es mayor para los machos (40.270 kg) que para las hembras (37.818 kg). Es decir, se estima que los machos tendrán mayor peso al nacimiento que las hembras.

Esta conclusión concuerda con lo obtenido a través del análisis descriptivo, donde el peso medio al nacimiento de los machos era de 39.875 kg por los 37.169 kg de peso medio de las hembras. Aunque hay que recordar que son dos análisis distintos, siendo el primero una media calculada directamente y esto estimaciones de un modelo lineal mixto utilizando métodos MCMC.

Además, si se utilizan los errores típicos se pueden obtener intervalos de confianza para la estimación de cada nivel de la variable *sexo*. Destaca como, para el nivel 3, los individuos con sexo desconocido, el error típico (1.804 kg) es 18 veces más que para los machos (0.110 kg) y para las hembras (0.106 kg). Por tanto, los intervalos de confianza tendrán menor longitud para los niveles conocidos:

$$\text{IC}_{\text{hembras}} = (37.610, 38.026) \text{ kg},$$

$$\text{IC}_{\text{machos}} = (40.255, 40.685) \text{ kg},$$

$$\text{IC}_{\text{NAs}} = (40.210, 47.281) \text{ kg}.$$

En conclusión, podemos decir que la precisión de las estimaciones varía de unas categorías a otras en la variable *sexo*. Como era de esperar, aunque el peso de los machos es mayor que el de las hembras, las estimaciones tiene una fiabilidad similar. Sin embargo,

las de los individuos con sexo desconocido son menos precisas y fiables. Además, estos últimos animales parecen ser machos, o por lo menos, se observa como el intervalo de confianza de los machos se encuentra dentro del de los individuos desconocidos.

De manera similar se puede ver qué individuos proporcionan mejores valores genéticos. Hay que tener en cuenta que aquí se esta trabajando con 932 712 niveles del identificador, por lo que se genera una estimación por cada uno de ellos. Como analizarlos todos individualmente es una tarea casi imposible, nos centramos en un análisis global y en aquellos individuos que supondrán mayor o menor peso al nacimiento para sus hijos (ver A.3.2).

Considerando las 932 712 estimaciones podemos observar (Figura 5.1) como se concentran todas alrededor del 0, con una estimación de la desviación típica de 4.937 kg, aunque un test de Lilliefors rechaza la normalidad (p-valor $<2.2e-16$). En concreto, estudiando la distribución de los distintos valores, se tiene que más del 50 % de las estimaciones se localizan en el intervalo $(-1.239 \text{ kg}, 0.444 \text{ kg})$. Sin embargo, serán las que se encuentra más alejadas de esos rangos las que se comentarán posteriormente de manera individual.

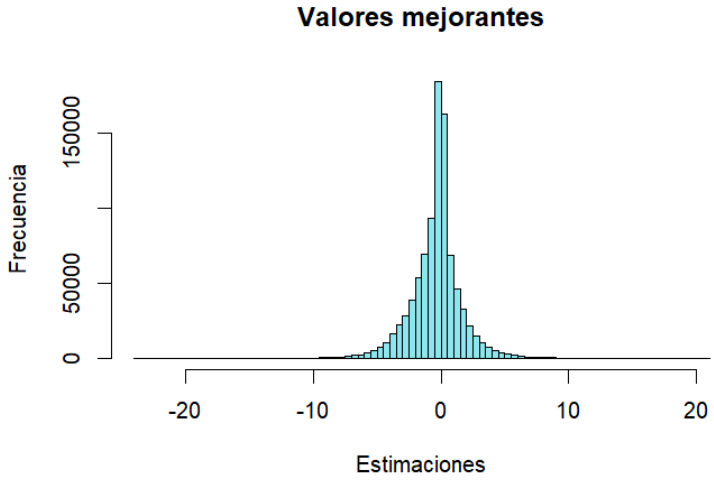

Figura 5.1: Distribución de los valores mejorantes (histograma).

En cuanto a la distribución de positivos y negativos, se observa como casi el 60 % de las estimaciones toman valores menores que 0. El 40 % restante (salvo un valor 0) son positivos, es decir, individuos cuyos hijos tendrán un peso mayor a la media. A continuación, se analizan de manera individual aquellas estimaciones que se consideran destacables.

La primera particularidad que se observa es para el último nivel, el 932 712, donde la estimación es 0. Esto es consecuencia de que se trata de un individuo ficticio, ya que son aquellos animales cuyos datos están ausentes, por lo que no se puede conocer el efecto que causan. Su efecto medio debería ser a priori 0, por lo que parece una estimación correcta.

Buscando el individuo con mayor valor mejorante se obtiene que la mayor estimación de u es para el individuo con id 624 998, cuyo valor es 20.790 kg. Es decir, se espera que los hijos de este individuo (macho) tengan mayor peso al nacimiento que el resto de animales. Un hijo suyo pesará al nacer en promedio 10.395 kg más que la media, pues sería el resultado de cruzar a ese individuo con otro con mérito genético igual a cero.

En el extremo opuesto estaría el inidviduo con id 535 533, cuya estimación de u es -23.926 kg. Es decir, se espera que los hijos de este individuo tengan un peso al nacimiento muy por debajo de la media. Además, es la estimación de mayor valor absoluto lo que quiere decir que es el individuo que tiene más influencia genética en el peso al nacimiento.

Como ya hemos comentado, el menor valor genético lo proporcionan los individuos desconocidos. Sin embargo, hay otros animales cuyo valor mejorante es tan pequeño que se puede considerar que no tienen influencia en el peso al nacimiento. Este será un individuo con pocos familiares, y estos sin datos de peso al nacimiento. Por ejemplo, la estimación de u para el individuo con id 435 992 es $2 \cdot 10^{-7}$ kg, es decir, prácticamente 0.

Una vez analizados los valores mejorantes es necesario estudiar la precisión de las estimaciones. Con un histograma (Figura 5.2) se ve como se distribuyen entre 0 kg (tie-

nen que ser positivos) y valores ligeramente superiores a 6 kg. Sin embargo, el valor 0 únicamente aparece para los individuos desconocidos, siendo el siguiente error típico más pequeño 0.304 kg. De manera similar a lo que ocurría con los valores mejorantes, hay un rango, en este caso entre 4 kg y 5 kg, donde se sitúan más del 50 % de los errores típicos. Para aquellos individuos de los que no se tienen datos, la varianza de su valor no debería superar la varianza de la población (24.371 kg^2).

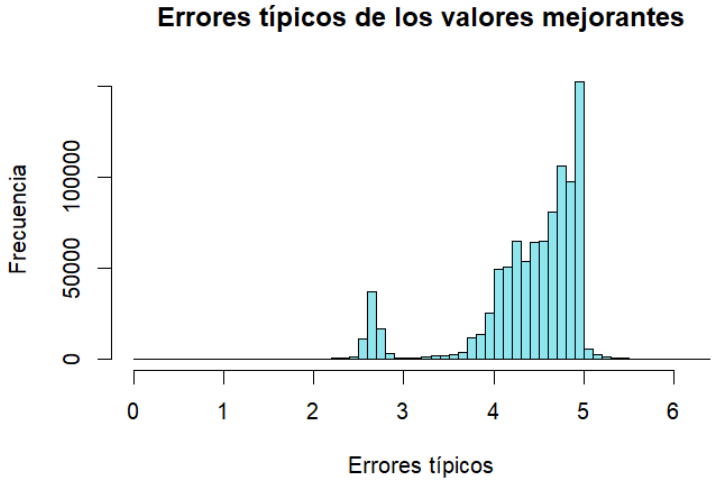

Figura 5.2: Distribución de los errores típicos (histograma).

Hemos resaltado 4 estimaciones, aquellas que suponían valores extremos, con lo que, utilizando intervalos de confianza construidos a partir de los errores típicos, podemos ver su precisión:

$$\text{IC}_{624998} = (18.285, 23.296) \text{ kg},$$

$$\text{IC}_{535533} = (-27.246, -20.606) \text{ kg},$$

$$\text{IC}_{\text{NA}} = (0.000, 0.000) \text{ kg},$$

$$\text{IC}_{435992} = (-9.658, 9, 658) \text{ kg}.$$

Se determinó como, para los mayores valores mejorantes (en valor absoluto) se tienen intervalos de confianza de longitud 5.010 kg (mayor positivo) y 6.639 kg (menor negativo). Sin embargo, para el individuo con *id* 435 992, la longitud aumenta considerablemente

(19.317 kg). Además, como ya se comentó previamente, los individuos desconocidos no aportan información, siendo la estimación 0.

Por otro lado, es posible analizar qué individuos proporcionan mayores y menores errores típicos y estudiar si tienen alguna o algunas características comunes. Para ello, se analizan los individuos a los que corresponden los errores, asi como sus hijos, en el caso de tenerlos.

Para los mayores errores típicos se tienen individuos cuyo peso al nacimiento es desconocido y no tienen hijos en los datos. En el caso de los que sí tienen descendencia, de ellos no se dispone del dato de peso al nacimiento. Es decir, se cometerá mayor error en la estimación cuando el individuo no proporcione información de la variable *pesonac* o no se tenga dicha información para sus hijos, si los tiene.

En el otro extremo, con errores típicos inferiores a 2 kg, estarían los individuos que tienen hijos en los datos y, además, se conoce el peso al nacimiento de la descendencia. Estos animales tendrán menor error típico en las estimaciones, intervalos de confianza más estrechos. Aunque, este error típico pequeño es casi independiente de si se conoce o no el peso al nacimiento para el propio individuo. Es decir, la mayor parte de la información proviene de tener o no muchos hijos.

Entre ambos tramos, con valores de los errores típicos entre 2 kg y 3 kg, se encuentra un último grupo de individuos. Aquellos con peso al nacimiento conocido, pero que no han tenido descendencia o, en su defecto, han tenido pocos hijos. Además, el peso al nacimiento de los hijos, en el caso de tenerlos, es desconocido.

results.txt

Este fichero de texto proporciona las estimaciones de las componentes de la varianza y las correlaciones genéticas, heredabilidades, etc. Las calcula como la media de los valores que toman en cada iteración (quitando los valores de las iteraciones de calentamiento, las primeras 30 000). Además, calcula también el error típico de cada varianza y heredabilidad.

En este caso se dispone de dos estimaciones para componentes de la varianza: la varianza genética aditiva (σ_u^2) y la varianza residual (σ_e^2). Además, se obtiene tambien para las razones (y correlaciones asociadas) de la varianza total debida a los efectos aditivos (heredabilidad genética aditiva, h^2) y a los efectos residuales (heredabilidad residual, h_e^2). Analizamos, a continuación, cada medida por separado.

La varianza genética aditiva (σ_u^2) es la fracción de variabilidad genética que se debe a efectos aditivos de los genes. Es decir, la suma del efecto de cada uno de los alelos que ayudan a producir el fenotipo (características observables) y no dependen de las combinaciones específicas de los alelos, sino de cada alelo en sí. Esta variabilidad es transmitida desde los padres a sus hijos.

En este caso, se dispone de una estimación de la varianza genetica aditiva media de 24.371 kg^2, con una error típico de 0.414 kg^2. Lo que se puede interpretar como que parte de las diferencias observadas en el peso al nacimiento son debidas a diferencias genéticas que están relacionadas de manera aditiva.

En cuanto a la varianza residual (σ_e^2), es una medida de la variabilidad fenotípica que no puede ser explicada por los factores que están incluidos en el modelo. En el modelo utilizado se tiene una estimación que vale 12.578 kg^2 (con desviación típica 0.265 kg^2). Esto indica que parte de la variabilidad del peso al nacimiento no puede ser explicada por el efecto fijo y el efecto aleatorio que se están considerando.

La heredabilidad (h^2) es la fracción de la varianza fenotípica que se explica por la varianza genética aditiva. Se puede ver como la proporción de la variabilidad genética que se transmite desde los padres a sus hijos, por lo que tomará un valor entre 0 y 1.

En este caso, toma valor medio 0.6596, lo que significa que el 65.96 % de la variabilidad fenotípica se explica por la variabilidad genética aditiva. En otras palabras, el 65.96 % de la variabilidad del peso al nacimiento se debe a diferencias genéticas entre individuos y no a diferencias ambientales. Es decir, el peso al nacimiento se hereda en 2/3.

Precisamente por esto último, y como no se tienen efectos ambientales en el modelo, el 34.04 % restante de la varianza fenotípica se explica por la varianza genética residual. Esto es la heredabilidad residual (h_e^2), es decir, la proporción de variación fenotípica que no puede ser explicada por los efectos genéticos aditivos.

Con esto se puede dar por finalizada la explicación del fichero *results.txt*, aunque también muestra otros datos como el nombre del archivo de parámetros o el número de iteraciones que se han realizado.

samples.txt

En este fichero se guardan las secuencias markovianas de las varianzas (σ_u^2, σ_e^2) y de la heredabilidad (h^2). Se pueden entender como las distribuciones a posteriori de las varianzas, de manera que cada línea de la salida es una muestra.

En este archivo, se observa que los valores de las tres medidas se van estabilizando con el paso de las iteraciones, es decir, convergen. Además, se observa como los valores de convergencia son muy similares a las medias de *results.txt*, aunque los primeros valores están muy alejados de ellos. Esto es debido a que, como ya se comentó, las primeras 30 000 ite-

raciones no se tienen en cuenta y, posteriormente, las muestran ya toman valores similares.

Si se centra el estudio en cada una de las tres cantidades (σ_u^2, σ_e^2, h^2) por separado, se puede entender la procedencia de las medias y desviaciones típicas proporcionadas por *results.txt*. Además, se podrán estudiar las correlaciones existentes entre iteraciones, planteándose si se pueden utilizar todas para los cálculos o no (en el archivo de parámetros se consideraban todas, `THIN INTERVAL 1`).

Empezamos por la **varianza genética aditiva**, de la que se tienen 100 000 estimaciones, una por cada iteración. Además, se sabe que las primeras 30 000 iteraciones son de calentamiento, por lo que se trabaja con las 70 000 restantes. Mientras que considerando las 100 000 estimaciones se tienen valores entre 16.399 kg^2 y 26.070 kg^2, al eliminar las primeras iteraciones, el rango se reduce, quedando entre 23.031 kg^2 y 25.817 kg^2. Observando la Figura 5.3, se comprueba como, tras un primer tramo con estimaciones más bajas, se termina estabilizando en valores cercanos al 24 kg^2.

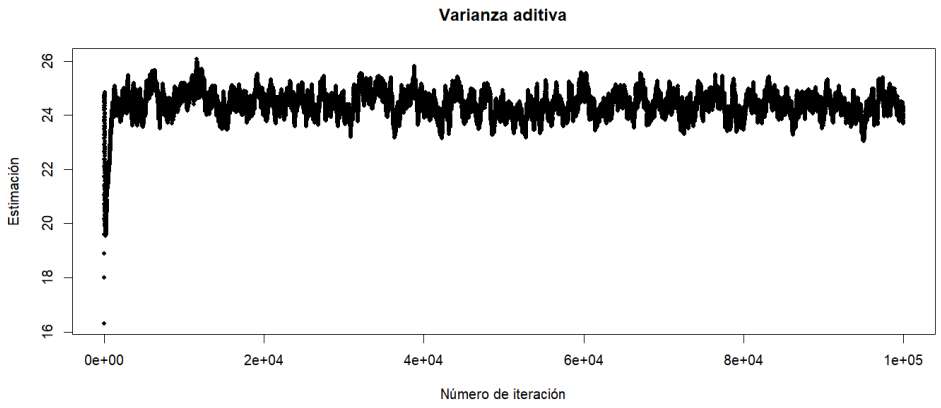

Figura 5.3: Evolución de la varianza genética aditiva.

Otra forma de observar las diferencias entre las estimaciones de calentamiento y las posteriores es a traves de un histograma (Figura 5.4), donde claramente las primeras iteraciones son inferiores a las demás. Además, si calculamos la media y la desviación típica de las 70 000 estimaciones finales se recuperan los valores proporcionados por *results.txt*:

$\mu(\sigma_u^2) = 24.371$ kg^2 y $\sigma(\sigma_u^2) = 0.4139$ kg^2.

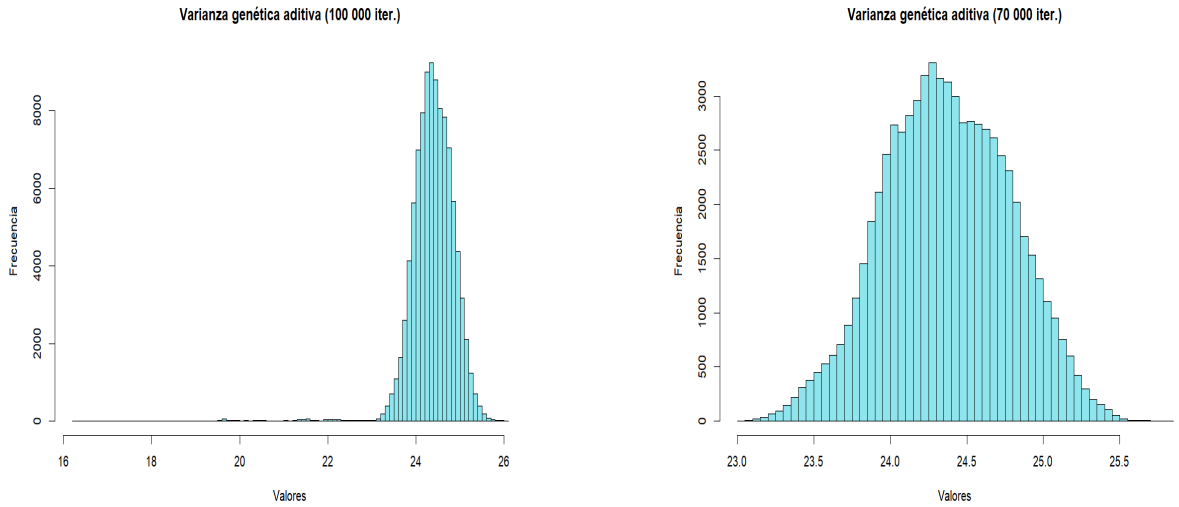

Figura 5.4: Histogramas de las distribuciones a posteriori de la varianza genética aditiva.

Para ver la dependencia entre las diferentes estimaciones se trabaja con correlogramas, tanto simple como parcial. La autocorrelación simple (ACF) mide la correlación de un punto de datos de una serie temporal con los puntos anteriores a diferentes retrasos. Por otro lado, las correlaciones parciales (PACF) muestran la correlación directa entre dos puntos de datos, una vez que se han eliminado las correlaciones indirectas a través de otros puntos de datos, es decir, mide la correlación con los residuos. Valores cercanos a 1 o -1 indican correlación, positiva o negativa, fuerte. Valores cercanos a cero indican correlación débil o nula.

En la Figura 5.5 se observan los correlogramas correspondientes a la varianza genética aditiva. En ACF, para 50 estimaciones, se tienen valores muy cercanos a 1, lo que indica una fuerte correlación positiva. Sin embargo, en el PACF, se observa un claro decrecimiento en los valores, siendo, a partir de 4 retrasos, correlación casi nula, es decir, existiendo independencia. Es decir, la correlación directa es muy débil a partir del cuarto retraso. Por lo tanto, podrían considerarse extracciones casi independientes de la distribución a posteriori si usamos `THIN INTERVAL 5`.

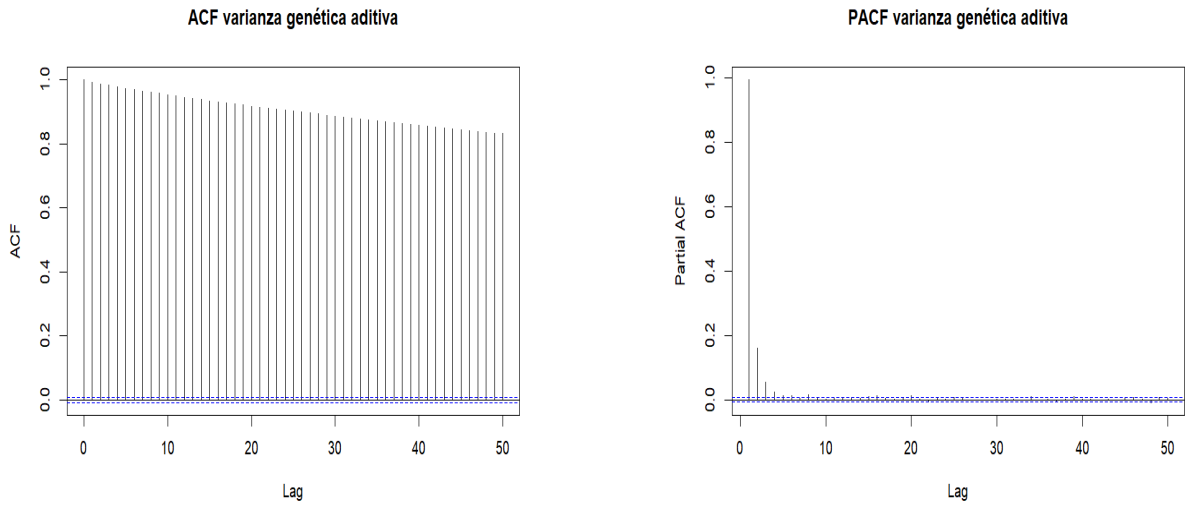

Figura 5.5: Correlogramas varianza genética aditiva.

En el caso de la **varianza residual** se tienen resultados similares (Figura 5.6), tomando valores entre 11.659 kg^2 y 249.690 kg^2 cuando se consideran las 100 000 estimaciones. Sin embargo, este rango se reduce considerablemente al eliminar las iteraciones de calentamiento, quedando entre 11.675 kg^2 y 13.467 kg^2. Por tanto, es un claro ejemplo de la influencia que podría llegar a tener no despreciar los primeros valores.

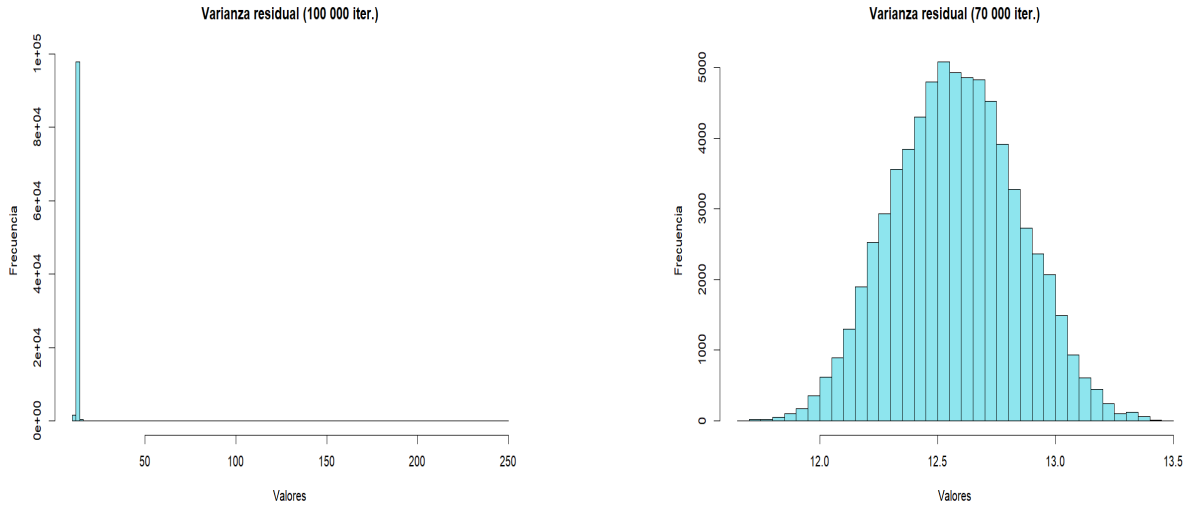

Figura 5.6: Histogramas de las distribuciones a posteriori de la varianza residual.

Como ocurría en la varianza genética aditiva, se puede comprobar como se han obtenido los resultados ofrecidos en *results.txt*. Para ello se calculan la media y la desviación típica de las estimaciones de la varianza residual y se comprueba que coinciden con las anteriores, ya que se tiene $\mu(\sigma_e^2) = 12.578$ kg^2 y $\sigma(\sigma_e^2) = 0.2653$ kg^2.

En cuanto a los correlogramas (Figura 5.7), se tiene que hay una fuerte correlación positiva entre más de 50 estimaciones (todas por encima de valor 0.8), lo que indica una fuerte dependencia. En cambio, para las correlaciones parciales se observa como los valores decrecen rápidamente, llegando a correlaciones casi nulas a partir del cuarto retardo.

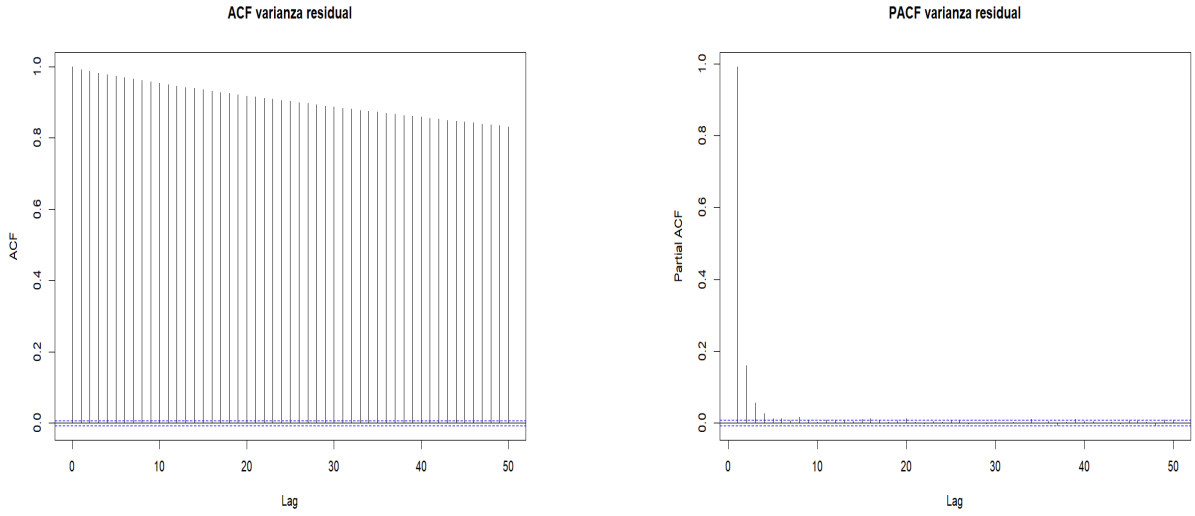

Figura 5.7: Correlogramas varianza residual.

Por último, con la **heredabilidad** se tiene algo muy parecido a las dos magnitudes anteriores. Teniendo en cuenta las 100 000 iteraciones, los valores se encuentran entre 0.082 y 0.685, pero eliminando los valores de calentamiento, el rango se ve reducido a 0.633 y 0.683. Esto se ve claramente en los histogramas correspondientes (Figura 5.8), donde hay gran diferencia entre trabajar con 70 000 iteraciones o con 100 000. Además, como en los casos anteriores, calculado la media ($\mu(h^2) = 0.660$) y la desviación típica ($\sigma(h^2) = 0.008$), se verifican los valores proporcionados por *results.txt*.

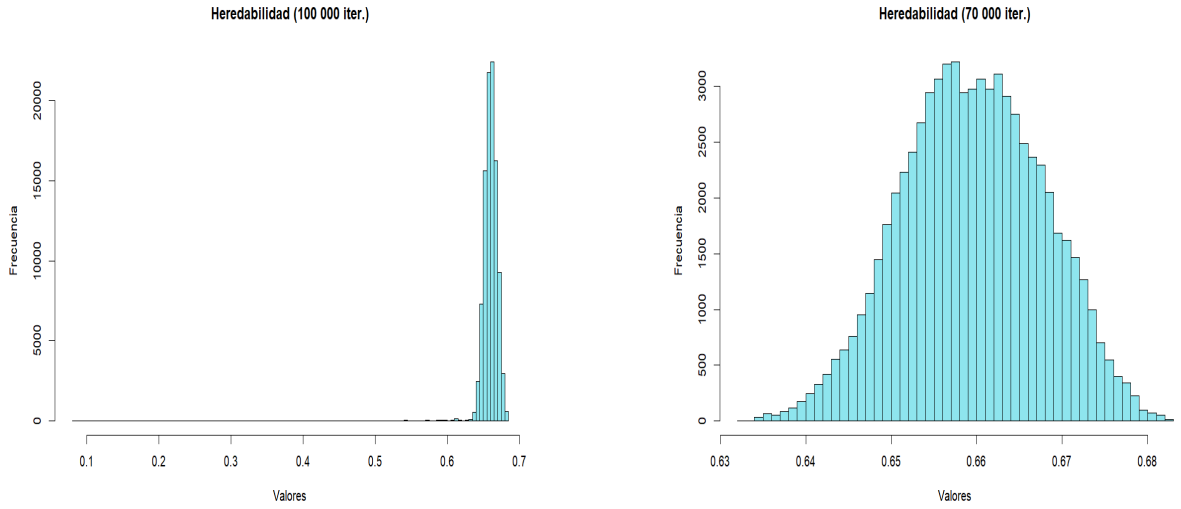

Figura 5.8: Histogramas heredabilidad.

En el caso de las correlaciones (Figura 5.9), se observa un gráfico ACF igual al que tenían las varianzas, con correlaciones fuertes para 50 estimaciones. En cambio, en las correlaciones parciales, aunque también descienden rápidamente, ya se podría considerar que no existe dependencia directa a partir del tercer retardo. Es decir, la correlación es débil en menos retrasos que para las varianzas.

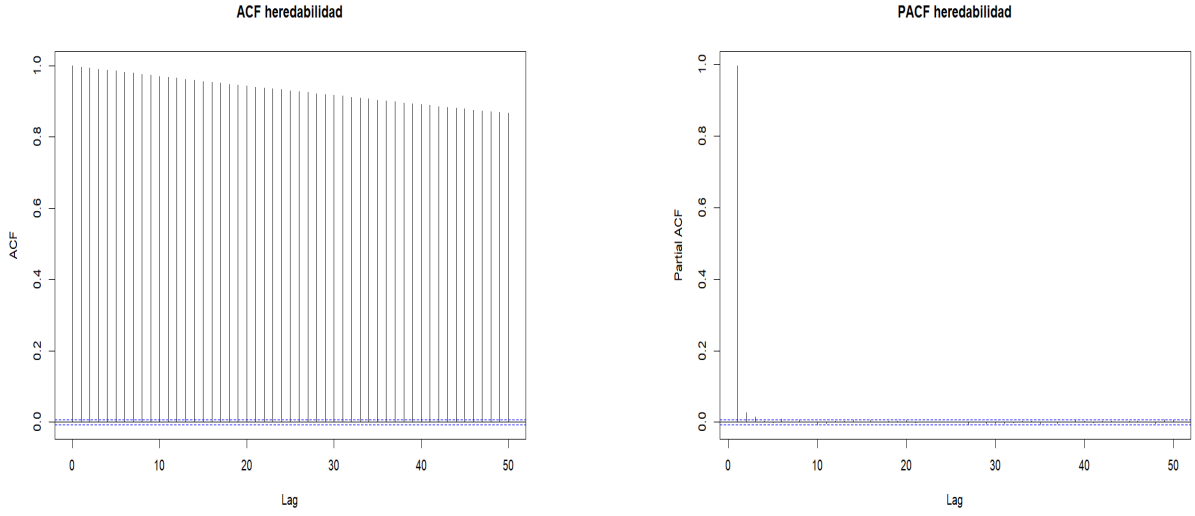

Figura 5.9: Correlogramas heredabilidad.

Por tanto, podemos concluir de todo lo anterior que, como era de esperar, las primeras iteraciones se alejan mucho del resultado final y es importante considerarlas de calentamiento. Además, se comprueba como *results.txt* proporciona directamente las medias y las desviaciones típicas calculadas a partir de los valores de *samples.txt* (eliminando las 30 000 estimaciones consideradas de calentamiento).

Por otro lado, con los correlogramas, se ha observado gran dependencia entre unas iteraciones y otras, en la mayoría de casos positiva. Se puede encontrar una solución a esto utilizando un intervalo para considerar las soluciones mayor, es decir, modificando el archivo de parámetros de forma que: `THIN INTERVAL` tome un valor mayor (10, 100,...). Con esta forma se podría realizar inferencia con mayor seguridad. Por ejemplo, con `THIN INTERVAL 10` se tendrían 7000 valores de las distribuciones a posteriori, lo que seguiría permitiendo estimar la media correspondiente.

Además de estos tres ficheros, se generan otros dos que no tienen interés, pues para nuestro modelo se pueden considerar vacíos. Por un lado, *thresholds.txt* que contiene información de caracteres que no participan en este modelo. Por otro lado estaría *samplesFE.txt*, que proporciona las cadenas markovianas de las estimaciones de los coeficientes de los efectos fijos, pero para obtenerlas sería necesario modificar el código fuente Fortran de `tm` (ver [12] 4.6.10).

Conclusión

Para estudiar la resolución de problemas de mejora genética con métodos MCMC se han explorado cinco posibilidades: implementación propia, librerías `rstan` y `MCMCglmm` de R, programa `loki` en C y programa `tm` en Fortran. Para evaluar la rapidez y fiabilidad de ellas se ha utilizado un conjunto de datos real con 931 689 individuos, buscando determinar qué tipo de individuos proporcionan mayor peso al nacimiento en su descendencia.

El programa propio es el que ha dado peor resultado, ya que para 8010 individuos el tiempo de ejecución era demasiado elevado y la realización de cada iteración se prolongaba durante más de 15 minutos, lo cual hizo imposible obtener resultados fiables o trabajar con el conjunto de datos completos.

Con las dos librerías de R se obtuvieron resultados similares. Aunque ninguna de las dos permitía trabajar con el conjunto total de los datos, `MCMCglmm` proporcionaba resultados con mayor rapidez y no necesitaba la imputación manual por parte del usuario de los pesos al nacimiento desconocidos. Algo similar ocurría con el programa loki, en este caso no se disponía de memoria suficiente para trabajar con los más de 900 000 individuos.

Por tanto, tan solo con el programa `tm` de Andrés Legarra se obtuvieron resultados para toda la población. Con él se ha determinado la influencia del sexo en el peso al nacimiento, obteniendo que el de los machos será mayor que el de las hembras. De manera similar, se han obtenido los valores mejorantes de cada uno de los animales, donde se ve

la influencia en el peso al nacimiento de cada individuo sobre sus hijos.

Tratando de estudiar estas últimas estimaciones del peso se ha llegado a la conclusión de que aquellos individuos que tienen pocos familiares y, además, estos no disponen de datos del peso al nacimiento, son los que van a tener menor influencia. Por otro lado, se ha visto como las estimaciones aumentan su error a medida que pierden datos, es decir, tendrán menor error aquellas correspondientes a individuos que posean hijos de los cuales se conozca el peso al nacimiento. Además, a nivel numérico se puede decir que el 65.96 % del peso al nacimiento se hereda.

Apéndice A

Códigos

Se explican a continuación los códigos utilizados para la realización de los ejemplos de los distintos paquetes de R. Comenzando por la parte común de todos ellos. Es decir, con la carga de librerías y de los datos correspondientes, así como con la eliminación de datos duplicados. El conjunto de datos `avanac` es aquel que contiene los datos reales disponibles, sin eliminar aquellos que están duplicados.

```
library(ggroups) # Para buildA.
library(Matrix)
library(errors)
library(tidyverse)
library(pedigree) # Para add.Inds, orderPed.
library(kinship2)
library(pedigreemm) # Para predigree, getA.
library(gap)

avanac <- readRDS("avanac.rds")
attach(avanac)
avanac.nuevo <- avanac[!duplicated(avanac), ] # Conjunto de datos.
```

A.1. Librería `rstan`

En este Apéndice se muestra, para el paquete `rstan`, el código completo de R del Caso 1 de los ejemplos estudiados, siendo el del Caso 2 análogo. Se darán pequeñas explicaciones para facilitar su interpretación. Lo primero sería seleccionar los individuos (10 padres y 10 madres con mayor número de hijos) del conjunto de datos y crear uno nuevo:

```
padres.maxhijos <- sort(table(avanac.nuevo$pa), decreasing = T)[1:10]
madres.maxhijos <- sort(table(avanac.nuevo$ma), decreasing = T)[1:10]
# Conjunto de datos (nuevo).
maxhijos <- subset(avanac.nuevo, id %in%
                   c(names(padres.maxhijos), names(madres.maxhijos)))
```

A continuación, se incluyen en el nuevo conjunto de datos los individuos que faltan, es decir, padres y madres de los ya seleccionados. Además se ordenan los datos de manera que los pesos al nacimiento conocidos se sitúen al final y los padres/madres tengan un índice menor que sus hijos.

```
# Completamos el conjunto de datos.
maxhijos2 <- add.Inds(maxhijos[,2:6])


# Ordenamos los datos para tener los pesos conocidos al final.
maxhijos2 <- rbind(subset(maxhijos2, is.na(pesonac)),
                   subset(maxhijos2, !is.na(pesonac)))
maxhijos2 <- maxhijos2[orderPed(maxhijos2),]
```

Se continúa añadiendo el sexo de los padres y madres. Para ello, si son padres se les asigna M (macho) y si son madres se les asigna H (hembra)

```
for(i in 1:nrow(maxhijos2)){
    if(maxhijos2$id[i] %in% maxhijos2$ma){
       maxhijos2$sexo[i] <-  "H"
```

```
    } else if(maxhijos2$id[i] %in% maxhijos2$pa){
       maxhijos2$sexo[i] <-  "M"
    }
}
```

Por último, en cuanto al tratamiento del conjunto de datos, quedaría imputar los pesos al nacimiento desconocidos, para ello se utilizarán diferentes métodos, tratando de comprobar si los resultados son estables con todos ellos.

- **Opción 1:** imputar los pesos al nacimiento con todos los individuos de la muestra considerada.
- **Opción 2:** imputar los pesos al nacimiento con todos los individuos del conjunto de datos.
- **Opción 3:** imputar los pesos al nacimiento con todos los individuos del conjunto de datos teniendo en cuenta el sexo.
- **Opción 4:** imputar los pesos al nacimiento con todos los individuos de la muestra considerada teniendo en cuenta el sexo.

```
# Adaptamos el conjunto de datos.
# Opción 1: con los datos de la muestra.
maxhijos2 <- within(maxhijos2, {
        pesonac[is.na(pesonac)] <- rnorm(length(pesonac[is.na(pesonac)]),
                                         mean(pesonac, na.rm = TRUE),
                                         sd(pesonac, na.rm = TRUE))
        g1 <- ifelse(sexo == "M", 1, 0)
        g2 <- ifelse(sexo == "H", 1, 0)
        ID <- id
        id <- ifelse(!is.na(id), id, 0)
        pa <- ifelse(!is.na(pa), pa, 0)
```

```
        ma <- ifelse(!is.na(ma), ma, 0)
})


# Opción 2: con todos los datos.
maxhijos2 <- within(maxhijos2, {
        pesonac[is.na(pesonac)] <- rnorm(length(pesonac[is.na(pesonac)]),
                                         mean(avanac.nuevo$pesonac,
                                              na.rm = TRUE),
                                         sd(avanac.nuevo$pesonac,
                                            na.rm = TRUE))
        g1 <- ifelse(sexo == "M", 1, 0)
        g2 <- ifelse(sexo == "H", 1, 0)
        ID <- id
        id <- ifelse(!is.na(id), id, 0)
        pa <- ifelse(!is.na(pa), pa, 0)
        ma <- ifelse(!is.na(ma), ma, 0)
})

# Opción 3: con todos los datos y el sexo .
for(i in 1:nrow(maxhijos2)){
    if(is.na(maxhijos2$pesonac[i])){
        if(maxhijos2$sexo[i] == "M"){
            maxhijos2$pesonac[i] <- rnorm(1,
                mean(avanac.nuevo$pesonac[avanac.nuevo$sexo=="M"],
                     na.rm = TRUE),
                sd(avanac.nuevo$pesonac[avanac.nuevo$sexo=="M"],
                   na.rm = TRUE))
        }else{
```

```
            maxhijos2$pesonac[i] <- rnorm(1,
                mean(avanac.nuevo$pesonac[avanac.nuevo$sexo=="H"],
                     na.rm = TRUE),
                sd(avanac.nuevo$pesonac[avanac.nuevo$sexo=="H"],
                   na.rm = TRUE))
        }}}
maxhijos2 <- within(maxhijos2, {
        g1 <- ifelse(sexo == "M", 1, 0)
        g2 <- ifelse(sexo == "H", 1, 0)
        ID <- id
        id <- ifelse(!is.na(id), id, 0)
        pa <- ifelse(!is.na(pa), pa, 0)
        ma <- ifelse(!is.na(ma), ma, 0)
})

# Opción 4: con los datos de la muestra y el sexo.
for(i in 1:nrow(maxhijos2)){
    if(is.na(maxhijos2$pesonac[i])){
        if(maxhijos2$sexo[i] == "M"){
            maxhijos2$pesonac[i] <- rnorm(1,
                mean(maxhijos2$pesonac[maxhijos2$sexo=="M"],
                na.rm = TRUE), 0) # Solo hay un dato (sd = 0)
        }else{
            maxhijos2$pesonac[i] <- rnorm(1,
                mean(maxhijos2$pesonac[maxhijos2$sexo=="H"],
                na.rm = TRUE),
                sd(maxhijos2$pesonac[maxhijos2$sexo=="H"],
                na.rm = TRUE))
        }}}
```

```
maxhijos2 <- within(maxhijos2, {
        g1 <- ifelse(sexo == "M", 1, 0)
        g2 <- ifelse(sexo == "H", 1, 0)
        ID <- id
        id <- ifelse(!is.na(id), id, 0)
        pa <- ifelse(!is.na(pa), pa, 0)
        ma <- ifelse(!is.na(ma), ma, 0)
})
```

Para finalizar, se construye la matriz de parentescos y se adaptan los datos al lenguaje Stan.

```
# Construimos la matriz de parentescos.
G <- buildA(maxhijos2)
N <- dim(maxhijos2)[1]

# Conjunto de datos adaptado al lenguaje Stan.
data <- with(maxhijos2, list(N = N, y = pesonac, g1 = g1, g2 = g2, G = G))
```

Con todo esto se tienen los datos adaptados al lenguaje de programación Stan, faltaría escribir el programa correspondiente (explicado en detalle en 3.2) y aplicarlo a dichos datos con la función `stan`.

```
# Programa en lenguaje Stan.
library("rstan")
datos.stan <- "
data {
      int N;
      vector[N] y;
```

```
      vector[N] g1;
      vector[N] g2;
      matrix[N, N] G;
      }
transformed data {
      matrix[N, N] C;
      C = cholesky_decompose(G);
      }
parameters {
      vector[2] b;
      vector[N] z;
      real sigma_p2;
      real sigma_r2;
      }
transformed parameters {
      real sigma_p;
      real sigma_r;
      vector[N] g;
      sigma_p = sqrt(sigma_p2);
      sigma_r = sqrt(sigma_r2);
      g = sigma_p * C * z;
      }
model {
      b ~ normal(0, 10);
      sigma_p2 ~ inv_gamma(1,1);
      sigma_r2 ~ inv_gamma(1,1);
      z ~ normal(0, 1);
      y ~ normal(b[1] * g1 + b[2] * g2 + g, sigma_r);
      }
```

```
generated quantities {
      real h2;
      real p;
      real r;
      p = sigma_p2;
      r = sigma_r2;
      h2 = p / (p + r);
      }
"
```

Con lo anterior, se dispone del programa con el algoritmo que se quiere aplicar. Se completaría utilizando la función `stan` con los argumentos correspondientes.

```
parms <- c("b", "p", "r", "h2")
f1 <- stan(model_code = datos.stan, data = data, chains = 1, iter = 10,
           verbose = 0)
print(f1, pars = parms, digits_summary = 3)
```

A.2. Librería `MCMCglmm`

En este Apéndice se muestra el código completo de R del Caso 1 de los ejemplos estudiados, siendo el del Caso 2 y el del Caso 3 análogos. Lo primero sería seleccionar, ordenar y completar los datos de los individuos de la misma manera que se hizo en el paquete `rstan`:

```
padres.maxhijos <- sort(table(avanac.nuevo$pa), decreasing = T)[1:10]
madres.maxhijos <- sort(table(avanac.nuevo$ma), decreasing = T)[1:10]
# Conjunto de datos (nuevo).
maxhijos <- subset(avanac.nuevo, id %in%
                   c(names(padres.maxhijos), names(madres.maxhijos)))
```

```
# Completamos el conjunto de datos.
maxhijos2 <- add.Inds(maxhijos[,2:6])

# Ordenamos los datos para tener los pesos conocidos al final.
maxhijos2 <- rbind(subset(maxhijos2, is.na(pesonac)),
                   subset(maxhijos2, !is.na(pesonac)))
maxhijos2 <- maxhijos2[orderPed(maxhijos2),]

# Añadimos el sexo a los padres y madres.
for(i in 1: nrow(maxhijos2)){
    if(maxhijos2$id[i] %in% maxhijos2$ma){
       maxhijos2$sexo[i] <-  "H"
    }else if(maxhijos2$id[i] %in% maxhijos2$pa){
       maxhijos2$sexo[i] <-  "M"
    }}

# Adaptamos el conjunto de datos.
maxhijos2 <- within(maxhijos2, {id <- ifelse(!is.na(id), id, 0)
                                pa <- ifelse(!is.na(pa), pa, 0)
                                ma <- ifelse(!is.na(ma), ma, 0)
                    })
```

A continuación se busca calcular la matriz de parentescos inversa en formato "sparse". Para ello se utiliza la función `getA` del paquete `pedigreemm`, a la cual hay que introducirle los datos en formato "pedigree".

```
# Convertimos las datos a fomato pedrigree.
pedigree_mat <- pedigree(label = maxhijos2$id, # Individuo
                         dam = maxhijos2$ma, # Madre
                         sire = maxhijos2$pa) # Padre
```

```
# Calculamos la inversa de A y le damos nombre a las filas.
Ainv <- solve(getA(pedigree_mat))
rownames(Ainv) <- maxhijos2$id
```

El siguiente paso es determinar las distribuciones a priori según sea residuos (`R`), efectos aleatorios (`G`) o efectos fijos (`B`).

```
prior <- list(R = list(V = 1, nu = 1),
              G = list(G1 = list(V = 1, nu = 1)),
              B = list(mu = c(0,0),
                       V = matrix(c(1,0,0,1), ncol = 2) * 1000))
```

Por último, se procede a realizar el modelo, para ello se ejecuta la función `MCMCglmm` con los argumentos correspondientes al modelo considerado. Hay que tener en cuenta que la matriz de parentescos inversa debe llevar el nombre del efecto aleatorio, de manera que cada uno de los posibles valores del efecto aleatorio (en este caso, identificadores de los individuos) se corresponda con el nombre de alguna fila de matriz.

```
modelo <- MCMCglmm(pesonac ~ sexo, # Variable respuesta y efecto fijo.
                   random = ~ id, # Efectos aleatorios.
                   data = maxhijos2, # Conjunto de datos.
                   nitt = 100000, # Número de iteraciones de la cadena.
                   prior = prior, # Distribuciones a priori.
                   ginverse = list(id = Ainv)) # Matriz inversa.
```

A.3. TM Andrés Legarra

A.3.1. Archivos de entrada

En este Apéndice se muestra el código completo de R utilizado para la creación de los dos primeros archivos necesarios para aplicar el programa implementado por Andrés

Legarra. Como en casos anteriores lo primero es cargar los datos disponibles y eliminar los individuos duplicados.

```
avanac <- readRDS("avanac.rds")
avanac.nuevo <- avanac[!duplicated(avanac), ]
```

A continuación, se crea un archivo que contenga únicamente los datos de pedigrí, es decir, el identificador del individuo (id), el identificador de su padre (pa) y el identificador de su madre (ma).

```
avanac.ped <- with(avanac.nuevo, data.frame(id, ma, pa))
```

Se añaden a este conjunto de datos de pedigrí aquellos individuos que no están en él pero que son madre o padre. Además, se ordenan de manera que los hijos tengan un índice mayor que el de sus padres.

```
avanac.ped2 <- add.Inds(avanac.ped) # Pedigrí completo.
orden.ped <- orderPed(avanac.ped2) # Orden de los inidviduos.
avanac.ped3 <- avanac.ped2[orden.ped,] # Pedigrí completo y ordenado.
```

Hay que pasar el identificador de cada individuo a formato numérico, ya que está guardado como variable factor. Además, se necesita una función que haga la recodificación de los id, ya que tienen que ir desde 1 hasta el número de individuos del conjunto de datos.

```
avanac.ped3$id <- as.numeric(avanac.ped3$id) # id a numérico.


# Función: De índice original a índice renumerado.
idori.a.renum <- function(id){
  aux <- match(id, avanac.ped3$id) # Posición de cada id en avanac.ped3.
  aux[is.na(aux)] <- length(avanac.ped3$id)+1  # NA = nº individuos + 1.
  aux
}
```

Se crea el correspondiente archivo de **pedigrí** con los individuos correctamente renumerados y se guarda con formato "csv".

```
pedigri <- data.frame(lapply(avanac.ped3, idori.a.renum))


# Guardamos el pedigrí.
write.table(pedigri, "pedigri.csv", sep = " ", row.names = FALSE,
            col.names = FALSE, quote = FALSE)
```

Por último, para el archivo de **datos**, se trabaja con el archivo inicial de individuos, pero teniendo en cuenta pequeñas modificaciones. Hay que renumerar los identificadores de los individuos, considerar el sexo como una variable numérica en la que los valores desconocidos son un nivel más y considerar 0 los valores desconocidos del peso al nacimiento.

```
# Renumerar.
avanac.nuevo$id <- idori.a.renum(avanac.nuevo$id)
avanac.nuevo$pa <- idori.a.renum(avanac.nuevo$pa)
avanac.nuevo$ma <- idori.a.renum(avanac.nuevo$ma)


# Modificamos la variable sexo.
avanac.nuevo$sexo <- as.numeric(avanac.nuevo$sexo)
avanac.nuevo$sexo[is.na(avanac.nuevo$sexo)] <- 3 # NA = 3.


# Modificamos la variable pesonac.
avanac.nuevo$pesonac[is.na(avanac.nuevo$pesonac)] <- 0 # NA = 0.
```

A la hora de guardar el archivo correspondiente a los datos, es importante tener en cuenta que el orden de las variables debe ser el correcto. La primera en aparecer debe ser el *sexo*, que se corresponde con la sexta columna del fichero de datos. A continuación aparece el *id*, es decir, la columna número 2 de los datos. Y la última variable será el peso al nacimiento (*pesonac*), la quinta variable del conjunto *avanac.nuevo*.

```
# Guardamos el archivo de datos.
write.table(avanac.nuevo[,c(6,2,5)], "datos.csv", sep = " ",
            row.names = FALSE, col.names = FALSE, quote = FALSE)
```

A.3.2. Análisis de soluciones

En este apartado se proporciona el código utilizado para analizar la soluciones del modelo `tm`, concretamente, lo obtenido a partir del resultado del fichero *solutions.txt* y *samples.txt*. Empezando por las soluciones, lo primero es cargar el archivo correspondiente y dividir las estimaciones en efectos fijos ($\vec{\beta}$), calculando los intervalos de confianza correspondientes, y en efectos aleatorios ($\vec{u}$).

```
solutions <- read.table("solutions.txt")

beta <- solutions[1:3,] # Estimaciones por sexo (efecto fijo).
rownames(beta) <- c("Hembra", "Machos", "Desconocidos")

# Intervalos de confianza.
inferior.beta <- beta[,1] - 1.96 * beta[,2] # Extremo inferior.
superior.beta <- beta[,1] + 1.96 * beta[,2] # Extremo superior.
longitud.IC <- superior.beta - inferior.beta # Longitud del intervalo.

u <- solutions[4:dim(solutions)[1],] # Valores genéticos.
rownames(u) <- as.character(1:932712)
```

Una vez separadas, interesa analizar los valores genéticos, ya que se dispone de uno por cada individuo. Primeramente, se realiza un análisis global de los resultados (valores mejorantes y residuos).

```
# Análisis global de los valores mejorantes.
hist(u[,1], breaks = 100, main = "Valores mejorantes",
```

```
     xlab = "Estimaciones", ylab = "Frecuencia", col = "cadetblue2")
summary(u[,1]) # Valores desde -23.93 hasta 20.71.
table(sign(u[,1])) # 545867 negativos y 386844 positivos.

# Análisis global de los errores típicos.
hist(u[,2], breaks = 50, main = "Errores típicos de los valores mejorantes",
     xlab = "Errores típicos", ylab = "Frecuencia", col = "cadetblue2")
summary(u[,2]) # Valores entre 0 y 6.39.
```

Para simplificar el análisis individual, se buscan los extremos, es decir, los valores mayores y menores. Además se debe tener especial cuidado cuando se nombra el individuo al que pertenecen, ya que es necesario renumerar el *id* al original.

```
# Mayor (positivo).
mayor <- which.max(u[,1]) # Individuo con id = 598545 (recodificado).
max(u[,1]) # 20.79047.
se.mayor <- u[mayor,2] # 1.28 error típico.
renum.a.idori(mayor) # Individuo con id = 624998 (original).
avanac.nuevo[avanac.nuevo$id == 624998, ] # Macho.
IC.mayor <- c(max(u[,1]) - 1.96*se.mayor, max(u[,1]) + 1.96*se.mayor)
longitud.mayor <- IC.mayor[2] - IC.mayor[1] # Longitud 5.01.

# Menor (negativo).
menor <- which.min(u[,1]) # Individuo con id = 412414 (recodificado).
min(u[,1]) # -23.926.
se.menor <- u[menor,2] # 1.70 error típico.
renum.a.idori(menor) # Individuo con id = 535533 (original).
avanac.nuevo[avanac.nuevo$id == 535533, ] # Macho.
IC.menor <- c(min(u[,1]) - 1.96*se.menor, min(u[,1]) + 1.96*se.menor)
longitud.menor <- IC.menor[2] - IC.menor[1] # Longitud 6.64.
```

```
# Menor (valor absoluto).
menor.abs <- which.min(abs(u[,1])) # Individuos desconcidos.
min(abs(u[,1])) # Valor 0.
se.menor.abs <- u[menor.abs,2] # 0 error típico.
renum.a.idori(menor.abs) # Comprobación (NA).

# Menor (valor absoluto, sin NA).
menor.abs2 <- which.min(abs(u[1:932711,1])) # Individuo con id = 808026
                                            # (recodificado).
min(abs(u[1:932711,1])) # 2e-07 (practicamente 0).
se.menor.abs2 <- u[menor.abs2,2] # 4.93 error típico.
renum.a.idori(menor.abs2)  # Individuo con id = 435992 (original).
IC.menor.abs2 <- c(min(abs(u[1:932711,1])) - 1.96*se.menor.abs2,
                   min(abs(u[1:932711,1])) + 1.96*se.menor.abs2)
longitud.menor.abs2 <- IC.menor.abs2[2] - IC.menor.abs2[1] # 19.32.
```

Por último, se busca analizar los individuos que proporcionan mayores y menores errores típicos en las estimaciones de los valores genéticos.

```
# Mayor error típico.
mayores.se <- head(order(u[,2], decreasing = TRUE), 10)
mayores.se.ori <- renum.a.idori(mayores.se) # id original.

# Estudio de los hijos y el peso al nacimiento.
mayores.se.hijos <- data.frame()
for(i in 1:length(mayores.se)){
  individuo <- mayores.se.ori[i]
  datos <- avanac.nuevo[which(avanac.nuevo$pa == individuo),]
```

```
  hijos <- sum(avanac.nuevo$pa == mayores.se.ori[i], na.rm = TRUE)
  pes.des <- sum(is.na(datos$pesonac))
  pes.con <- hijos - pes.des
  mayores.se.hijos <- rbind(mayores.se.hijos,
                      cbind(individuo, hijos, pes.des, pes.con))
}
mayores.se.hijos

# Menor error típico.
menores.se <- head(order(u[,2], decreasing = FALSE),20)
menores.se.ori <- renum.a.idori(menores.se)  # id original.

# Estudio de los hijos y el peso al nacimiento.
menores.se.hijos <- data.frame()
for(i in 1:length(menores.se)){
  individuo <- menores.se.ori[i]
  datos <- avanac.nuevo[which(avanac.nuevo$pa == individuo),]
  hijos <- sum(avanac.nuevo$pa == menores.se.ori[i], na.rm = TRUE)
  pes.des <- sum(is.na(datos$pesonac))
  pes.con <- hijos - pes.des
  menores.se.hijos <- rbind(menores.se.hijos,
                      cbind(individuo, hijos, pes.des, pes.con))
}
menores.se.hijos

# Errores típicos intermedios
intermedios.se <- which(u[,2] >= 2.665 & u[,2] <= 2.67)
intermedios.se.ori <- renum.a.idori(intermedios.se) # id original.
```

```
# Estudio de los hijos y el peso al nacimiento.
for(i in 1:length(intermedios.se)){
  individuo <- intermedios.se.ori[i]
  datos <- avanac.nuevo[which(avanac.nuevo$pa == individuo),]
  hijos <- sum(avanac.nuevo$pa == intermedios.se.ori[i], na.rm = TRUE)
  pes.des <- sum(is.na(datos$pesonac))
  pes.con <- hijos - pes.des
  intermedios.se.hijos <- rbind(intermedios.se.hijos,
                                cbind(individuo, hijos, pes.des, pes.con))
}
intermedios.se.hijos
intermedios.se.hijos[which(intermedios.se.hijos$hijos != 0),]
```

A continuación, de manera similar, se analizan los resultados proporcionados por *samples.txt*. Lo primero es cargar el conjunto de datos que contiene las estimaciones de las distintas cantidad y comprobar que valores toman cada una de ellas.

```
samples <- read.table("samples.txt", header = TRUE)
summary(samples)
```

A partir de aquí se realizan las gráficas de cada una de las cantidades para todos los valores disponibles (100 000 iteraciones). Además, para cada una de ellas se hace el histograma correspondiente tras eliminar las iteraciones de calentamiento, se calculan media y desviación típica y se realizan los correlogramas.

```
# Varianza genética aditiva.
hist(samples[,1], breaks = 100,
     main = "Varianza genética aditiva (100 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")

plot(samples[,1], lwd = 1, pch = 20, main = "Varianza aditiva",
```

```
     xlab = "Número de iteración", ylab = "Estimación") # Evolución.

var_a <- samples[30001:100000,1] # Estimaciones.
hist(var_a, breaks = 50,
     main = "Varianza genética aditiva (70 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")
summary(var_a) # Valores entre 23.03 y 25.82.
               # Media 24.37072.
sd(var_a) # Desviación típica 0.4138772.
acf(var_a, lag.max = 50, main = "ACF varianza genética aditiva")
pacf(var_a, lag.max = 50, main = "PACF varianza genética aditiva")

# Varianza residual.
hist(samples[,2], breaks = 100,
     main = "Varianza residual (100 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")

plot(samples[,2], lwd = 1, pch = 20, main = "Varianza residual",
     xlab = "Número de iteración", ylab = "Estimación") # Evolución.

var_e <- samples[30001:100000,2] # Estimaciones.
hist(var_e, breaks = 50,
     main = "Varianza residual (70 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")
summary(var_e) # Valores entre 11.68 y 13.47.
               # Media 12.57804.
sd(var_e) # Desviación típica 0.2653092.
acf(var_a, lag.max = 50, main = "ACF varianza residual")
pacf(var_a, lag.max = 50, main = "PACF varianza residual")
```

```
# Heredabilidad.
hist(samples[,3], breaks = 100, main = "Heredabilidad (100 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")

plot(samples[,3], lwd = 1, pch = 20, main = "Heredabilidad",
     xlab = "Número de iteración", ylab = "Estimación") # Evolución.

h <- samples[30001:100000,3] # Estimaciones.
hist(h, breaks = 50, main = "Heredabilidad (70 000 iter.)",
     xlab = "Valores", ylab = "Frecuencia", col = "cadetblue2")
summary(h) # Valores entre 0.6327 y 0.6829.
           # Media 0.6595512.
sd(h) # Desviación típica 0.008215004.
acf(h, lag.max = 50, main = "ACF heredabilidad")
pacf(h, lag.max = 50, main = "PACF heredabilidad")
```

Bibliografía

[1] López Álvarez, M., "Cadenas de Márkov para modelos lineales", 2022, Trabajo Fin de Grado (Matemáticas), Universidad de Oviedo, Facultad de Ciencias. https://drive.google.com/file/d/1ZKFZKdezhvnLwBNxxJ4A2551dQXdkQmB/view?usp=sharing

[2] Zhao, J. H., Luan, J., Congdon, P., "Bayesian Linear Mixed Models with Polygenic Effects", 2018, Journal of Statistical Software, 85(6), 1–27. https://doi.org/10.18637/jss.v085.i06

[3] Package pedigree. https://cran.r-project.org/web/packages/pedigree/pedigree.pdf (Consulta: 10 de noviembre de 2023).

[4] Package ggroups. https://cran.r-project.org/web/packages/ggroups/ggroups.pdf (Consulta: 10 de noviembre de 2023).

[5] Package pedigreemm. https://cran.r-project.org/web/packages/pedigreemm/pedigreemm.pdf (Consulta: 10 de noviembre de 2023).

[6] Stan Reference Manual. https://mc-stan.org/docs/reference-manual/ (Consulta: 7 de diciembre de 2023).

[7] Package rstan. https://cran.r-project.org/web/packages/rstan/rstan.pdf (Consulta: 22 de diciembre de 2023).

[8] Chen T., Fox E.B., Guestrin C., "Stochastic Gradient Hamiltonian Monte Carlo", 2014, https://arxiv.org/pdf/1402.4102.pdf

[9] Package MCMCglmm https://cran.r-project.org/web/packages/MCMCglmm/MCMCglmm.pdf (Consulta: 25 de enero de 2024).

[10] Hadfield, J.D., "MCMC Methods for Multi-Response Generalized Linear Mixed Models: The MCMCglmm R Package", 2010, Journal of Statistical Software,33(2), 1–22. https://doi.org/10.18637/jss.v033.i02

[11] General Commands Manual Loki. https://manpages.debian.org/jessie/loki/prep.1.en.html (Consulta: 20 de marzo de 2024).

[12] Legarra A., "Threshold Model", 2011, https://genoweb.toulouse.inra.fr/~alegarra/tm_folder/manualtm.pdf

[13] Misztal I., Tsuruta S., Lourenco D., Masuda Y., "Manual for BLUPF90 family of programs", 2014, https://nce.ads.uga.edu/wiki/lib/exe/fetch.php?media=blupf90_all.pdf