UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE CIENCIAS

WinBUGS: un *software* para Inferencia Bayesiana

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

A C T U A R I A

P R E S E N T A

E L I Z A B E T H M O N R O Y C R U Z

FACULTAD DE CIENCIAS

UNAM

Tutor:

Dr. Eduardo Arturo Gutiérrez Peña

2006

Hoja de Datos del Jurado

1. Datos del alumno
   Monroy
   Cruz
   Elizabeth
   53 06 95 22
   Universidad Nacional Autonoma de México
   Facultad de Ciencias
   Actuaría
   098320080

2. Datos del tutor
   Dr
   Eduardo Arturo
   Gutierrez
   Peña

3. Datos del sinodal 1
   Dr
   Ramses Humberto
   Mena
   Chavez

4. Datos del sinodal 2
   Act
   Hortensia
   Cano
   Granados

5. Datos del sinodal 3
   Dr
   Alberto
   Contreras
   Cristan

6. Datos del sinodal 4
   Dr
   Raul
   Rueda
   Diaz del Campo

7. Datos del trabajo escrito
   WinBUGS: un software para Inferencia Bayesiana
   147 p
   2006

Mi más sincero y profundo agradecimiento al Dr. Eduardo Arturo Gutiérrez Peña *por el apoyo en la elaboración de este trabajo, así como los comentarios de los profesores:*

Dr. Ramsés Humberto Mena Chávez
Act. Hortensia Cano Granados
Dr. Alberto Contreras Cristán
Dr. Raúl Rueda Díaz del Campo

II

El hombre se encontraría indeciso en la mayor parte de las acciones de su vida si no tuviera nada para conducirse cuando le falta un conocimiento certero. Con frecuencia es necesario contentarse con un simple crepúsculo de probabilidad.

Dutens

Índice general

Prefacio

Una parte importante en la práctica actuarial implica la construcción de modelos y específicamente la solución a problemas por medio de modelos estadísticos. El presente trabajo propone el uso de WinBUGS para el análisis de una gran variedad de modelos abordados desde el punto de vista Bayesiano. WinBUGS es un *software* especializado en la implementación de Métodos de Monte Carlo basados en Cadenas de Markov, que se ha convertido en un *software* muy popular debido a su sencillez y al tipo de problemas que se pueden modelar. De hecho, existen algunas publicaciones (Congdon; 1998, 2001, 2005) dedicadas a realizar aplicaciones más elaboradas de las que se podrán encontrar en este documento.

Uno de los objetivos de este trabajo es presentar las herramientas suficientes para conocer el funcionamiento de WinBUGS, por lo que en los primeros capítulos se exponen conceptos muy generales que permiten introducir al lector a la descripción del *software* en sí.

En el Capítulo 2 se discute el enfoque Bayesiano, pues resulta de suma importancia para cualquier problema estadístico que se quiera analizar utilizando WinBUGS. Posteriormente, en el Capítulo 3 se exponen el algoritmo de Metropolis-Hastings y el muestreo de Gibbs, como fundamentos de WinBUGS, dado que hacer inferencias con base en la distribución final del parámetro de interés en un modelo Bayesiano de manera analítica generalmente resulta muy difícil o imposible, e incluso generar una muestra de la distribución final del parámetro a través del algoritmo de Gibbs resulta complicado. Sin embargo WinBUGS provee implementaciones del muestreo de Gibbs y Metropolis-Hastings relativamente sencillas.

El Capítulo 4 comprende la parte central de este trabajo. Se presenta la descripción del paquete que incluye una explicación de los principales menús y comandos de WinBUGS, así como la especificación de datos, la forma de construir un modelo gráficamente o por medio del lenguaje BUGS y cómo

llevar a cabo el análisis de un modelo. También se incluye una parte dedicada a explicar de forma muy concreta el funcionamiento de los paquetes *boa* y R2WinBUGS, los cuales conforman un complemento de WinBUGS y tienen el propósito de ayudar a determinar la convergencia de los algoritmos de Monte Carlo vía Cadenas de Markov. Al final de ese capítulo, se incluye una sección de aplicaciones en la que se exponen problemas basados en datos reales analizados a través de WinBUGS.

Por último, en el Capítulo 5 se dan las conclusiones de este trabajo.

Con este trabajo se incluye un CD que contiene animaciones, en las cuales podrá observar como llevar a cabo algunas rutinas en WinBUGS.

Capítulo 1

Introducción

1.1. Desarrollo de las técnicas de simulación en los años recientes

Las técnicas de Monte Carlo basadas en cadenas de Markov (MCMC por sus siglas en inglés) se han convertido en una herramienta muy poderosa desde la década pasada. A través de ellas, se resuelven muchos problemas de cálculo e inferencia que surgen en contextos multidimensionales y que son intratables analíticamente. El desarrollo del método de Monte Carlo se remonta al año 1944; la primera aplicación fue en el desarrollo de la bomba atómica (estudios de reacción nuclear) durante la Segunda Guerra Mundial. Sin embargo, el desarrollo sistemático de esta herramienta se realizó en trabajos de Harris y Khan en 1948 y Fermi, Metropolis y Ulam en ese mismo año. Posteriormente, Metropolis et al. (1953) propusieron un algoritmo básico que fue generalizado por Hastings (1970), dando lugar al algoritmo de Metropolis-Hastings, que constituye la versión más general de la familia de algoritmos MCMC. Más tarde, Geman y Geman (1984) elaboraron un método de simulación que también genera una cadena de Markov y que, tras ser difundido en Gelfand y Smith (1990), fue conocido como muestreo de Gibbs. El algoritmo básico de Metropolis-Hastings y el muestreo de Gibbs forman los dos esquemas fundamentales de la metodología MCMC a partir de los cuales se han creado otros, con fines más específicos y con distintas propiedades. El muestreo de Gibbs puede verse como un caso particular del algoritmo de Metropolis-Hastings. Sin embargo, como destacan Robert y Casella (1999), tiene algunas características particulares que le dan entidad propia.

Durante muchos años se han hecho aplicaciones específicas de los algoritmos MCMC en áreas de la mecánica estadística y la reconstrucción de imágenes digitales, pero ha sido recientemente, con el desarrollo de las herramientas computacionales, que se han introducido estos métodos en ámbitos como la inferencia Bayesiana aplicada y la estimación máximo verosímil entre otras áreas.

WinBUGS es parte del desarrollo computacional que se ha dado en estos últimos años. WinBUGS es un *software* que permite realizar inferencia Bayesiana y lo hace a través del algoritmo de Gibbs.

1.2. WinBUGS: un *software* para Inferencia Bayesiana

WinBUGS es un *software* de libre acceso en la red [1]. Fue diseñado por Spiegelhalter, Thomas y Best y es parte del proyecto BUGS que desarrollaron estos investigadores para el análisis Bayesiano de modelos estadísticos a través de métodos Monte Carlo basados en Cadenas de Markov.

WinBUGS permite que métodos complejos de simulación sean asequibles para los usuarios de la estadística Bayesiana aplicada en diversas disciplinas. Además, WinBUGS es una herramienta muy flexible ya que le permite al usuario construir su propio modelo y, una vez construido, puede realizar un análisis Bayesiano de éste. Lo anterior representa una gran ventaja para el usuario, ya que puede invertir más tiempo en la construcción de su modelo y en la interpretación de los resultados del mismo, que en la codificación del análisis MCMC. Por otra parte, la ejecución del modelo usualmente es rápida aún cuando los modelos sean complicados y manejen grandes cantidades de información.

El *software* ofrece una interfaz con el usuario basada en cuadros de diálogo y comandos a través de los cuales se analiza el modelo, por lo que el ambiente de WinBUGS se vuelve más amigable. Además, también es posible realizar una interfaz con R[2] o S-Plus, los cuales manejan una sintaxis muy similar a la que usa WinBUGS, aunque WinBUGS no cuenta con tantos comandos como R o S-Plus. Por otro lado, existen aplicaciones especializadas, como PKBUGS a través de las cuales es posible analizar aplicaciones estadísticas relacionadas

[1] http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml
[2] http://ww.r-project.org

con la medicina. En el sitio http://www.winbugs-development.org.uk/, se pueden consultar nuevas extensiones del *software*.

En WinBUGS los modelos pueden ser especificados textualmente por medio del lenguaje BUGS, o a través de gráficos al usar una interfaz gráfica llamada *DoodleBUGS*.

El concepto del diseño del *software* se basa en la representación interna del modelo de probabilidad, que es análogo al que se construye gráficamente. En el modelo gráfico, los parámetros o las variables se representan por un nodo y los nodos se conectan por flechas, las cuales muestran una dependencia directa. Este tipo de propiedades permiten una representación abstracta del modelo, lo cual resulta natural para la filosofía orientada a objetos en el diseño del *software*.

La forma de operar de WinBUGS está fundamentada en el muestreo de Gibbs; es decir, dada un función de verosimilitud y una distribución inicial, el propósito es muestrear valores de los parámetros del modelo a partir de la distribución final. Una vez obtenidos estos valores es posible obtener estimadores de los parámetros y hacer todo tipo de inferencias sobre ellos.

Capítulo 2

Inferencia Bayesiana

2.1. Interpretación subjetiva de la probabilidad y teorema de Bayes

La teoría de probabilidad se ha desarrollado desde el siglo XVII y es ampliamente aplicada en diversas áreas. El concepto de probabilidad no sólo aparece en aplicaciones formales, sino que es usada frecuentemente en la vida cotidiana. A menudo se oyen o se dicen expresiones tales como: "es probable que llueva en la tarde", "probablemente gane mi equipo de futbol favorito", o "tiene posibilidades de pasar el examen". Cada una de estas expresiones refleja cierto conocimiento sobre la posible ocurrencia de algún suceso específico.

Existen diversas formas de interpretar la probabilidad. Una que se basa en la frecuencia, es decir, cuantifica la probabilidad tomando en cuenta las repeticiones de un experimento aleatorio (probabilidad frecuentista) y otra fundamentada en la interpretación subjetiva que se refiere al grado de creencia que tiene un sujeto acerca de la ocurrencia de algún suceso de acuerdo a la información que tenga sobre el mismo (probabilidad subjetiva). Esta última forma de medir la probabilidad suele ser criticada debido a que los juicios que emite una persona respecto a la ocurrencia de un suceso pueden ser poco consistentes y contradictorios. Sin embargo, cuando se hace una evaluación subjetiva, ésta puede estar parcialmente basada en la interpretación frecuentista de la probabilidad, ya que el sujeto puede tener en cuenta la frecuencia relativa de la ocurrencia del suceso en el pasado; no obstante, la asignación final de probabilidades númerica es responsabilidad del propio sujeto y refleja

su estado de conocimiento.

La metodología Bayesiana es un enfoque alternativo para el análisis estadístico de datos, que propone el uso de la probabilidad subjetiva. Esto la distingue de la inferencia estadística clásica, que cuantifica la probabilidad a través de la interpretación frecuentista. Otro argumento que apoya a la teoría Bayesiana es que existen experimentos que no se pueden repetir bajo las mismas condiciones o que no se pueden cuantificar bajo el esquema del enfoque frecuentista.

Los factores que incrementan el nivel de información con relación a un fenómeno o experimento aleatorio, van desde la información que proveen datos históricos observados hasta la apreciación de especialistas sobre ciertas características de dicho fenómeno. Esto le permite al enfoque Bayesiano hacer inferencias sobre el suceso en cuestión por medio del teorema de Bayes, por lo que la estadística Bayesiana toma su nombre de este teorema. Por ejemplo, al inicio de esta sección se mencionaba que en la vida cotidiana surgen expresiones como: "probablemente gane mi equipo de futbol favorito", si este suceso se abordara desde el punto de vista Bayesiano los factores que se pueden tomar en cuenta son: datos históricos que corresponden a los partidos que ya ha jugado el equipo y, además, podrían estar involucrados factores subjetivos como el clima o el apoyo del público, los cuales pueden afectar el resultado del juego.

A continuación se enuncia el teorema de Bayes para el caso de eventos. La versión para densidades se explica más adelante.

Sea (Ω, Φ, P) un espacio de probabilidad; si $A, B \in \Phi$ y $P(A) > 0$ entonces la probabilidad condicional del evento B dado A, se define de la siguiente forma

$$P(B|A) := \frac{P(B \cap A)}{P(A)}.$$

Si $B \in \Phi$, entonces $P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$ y, por lo tanto, se tiene la siguiente expresión para $P(B|A)$:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A|B)P(B) + P(A|B^c)P(B^c)}.$$

Generalizando el teorema de Bayes, si $\{B_i\} \in \Phi$, una partición del espacio muestral Ω tal que $B_i \in \Phi$ y $P(B_i) > 0$ para todo i, entonces

$$P(A) = \sum_i P(A|B_i)P(B_i).$$

Por lo tanto,

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_i P(A|B_i)P(B_i)}.$$

En resumen, el teorema de Bayes permite utilizar información de un evento A, produciendo una descripción conjunta de la incertidumbre del evento B_i a través de $P(B_i|A)$.

2.2. El enfoque Bayesiano

La contribución del teorema de Bayes en la estadística Bayesiana es que provee de un procedimiento inferencial mediante el cual, a partir de una distribución de probabilidad "inicial", se obtiene la distribución "final" que describe un estado de conocimiento a la luz de toda la información disponible. Más específicamente, en estadística Bayesiana, θ, la cantidad de interés en el problema de estudio, se modela probabilísticamente; es decir, se considera al valor de θ como una variable aleatoria (o vector aleatorio) con una función de distribución inicial (o *a priori*), $p(\theta)$. Esta función contiene la información subjetiva sobre el valor de θ o información basada en datos históricos.

A medida que se observan los datos, $\mathbf{x} := (x_1, x_2, ..., x_n)$, la función de distribución que modela a θ se modifica. El teorema de Bayes permite incorporar estos datos observados a través de la función de distribución final (o *a posteriori*), la cual se expresa como

$$p(\theta|\mathbf{x}) = \frac{p(\mathbf{x}, \theta)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\theta)p(\theta)}{\int_\Theta p(\mathbf{x}|\tilde{\theta})p(\tilde{\theta})d\tilde{\theta}} \ .$$

La distribución final de θ es la base para hacer inferencias sobre θ.

Por otro lado, $p(\mathbf{x})$ es la probabilidad conjunta de la muestra $\mathbf{x} = (x_1, \ldots, x_n)$ a partir del vector aleatorio $\mathbf{X} = (X_1, \ldots, X_n)$ y es una constante respecto a θ, por lo que

$$p(\theta|\mathbf{x}) \propto p(\mathbf{x}|\theta)p(\theta),$$

donde $p(\mathbf{x}|\theta)$ es la función de verosimilitud. En el caso de que los componentes del vector aleatorio $\mathbf{X} = (X_1, \ldots, X_n)$, sean independientes dado θ, se tiene que

$$p(\mathbf{x}|\theta) = \prod_{i=1}^{n} p(x_i|\theta).$$

Al igual que en estadística clásica, se puede realizar la estimación puntual de θ, por ejemplo a partir de alguna medida de tendencia central

$$\hat{\theta} := E(\theta|\mathbf{x}) = \int_{\Theta} \theta p(\theta|\mathbf{x}) d\theta.$$

Aún cuando no se cuente con información muestral, se puede calcular $\hat{\theta}$ mediante $p(\theta)$ en lugar de la distribución de probabilidad final. Mas aún, en estadística Bayesiana la *estimación puntual* de θ se plantea como un problema de decisión. Es decir, se construye una función de utilidad $u(\hat{\theta}^*, \theta)$, donde $\hat{\theta}^*$ representa el valor propuesto del estimador de θ. El estimador $\hat{\theta}$ es entonces el valor de $\hat{\theta}^*$ que maximiza la utilidad esperada final

$$\bar{u}(\hat{\theta}^*) = \int u(\hat{\theta}^*, \theta) p(\theta|\mathbf{x}) d\theta.$$

Además se puede realizar el contraste de k hipótesis,

$$H_1 : \theta \in \Theta_1$$

$$H_2 : \theta \in \Theta_2$$

$$\vdots$$

$$H_k : \theta \in \Theta_k$$

y usualmente se escoge la que tenga la probabilidad más alta. La probabilidad final de la hipótesis H_j se calcula de la siguiente forma

$$P(H_j|\mathbf{x}) = \int_{\Theta_j} p(\theta|\mathbf{x}) d\theta.$$

Como en el caso de la estimación puntual, es posible plantear el problema de contraste de hipótesis como un problema de decisión.

En muchos casos, más que interesarse en el valor de un parámetro desconocido θ, se pretende describir el comportamiento de observaciones futuras del fenómeno aleatorio en cuestión. Dado el valor de θ, la función que describe el comportamiento de la observación futura $Y = y \equiv x_{n+1}$ es $p(y|\theta)$, pero por lo general el valor de θ es desconocido. La estadística frecuentista aborda este problema estimando puntualmente a θ con base en la muestra observada y sustituyendo dicho estimador $\hat{\theta}$ en $p(y|\theta)$, es decir, se utiliza $p(y|\hat{\theta})$. Desde la

perspectiva Bayesiana el modelo $p(y|\theta)$ junto con la distribución inicial $p(\theta)$ inducen una distribución conjunta para el vector aleatorio (Y, θ) mediante

$$p(y, \theta) = p(y|\theta)p(\theta),$$

por lo tanto,

$$p(y) = \int_{\Theta} p(y, \theta)d\theta,$$

la cual es conocida como la distribución predictiva inicial (o *a priori*) y describe el comportamiento de la observación futura Y basado en la información contenida en $p(\theta)$. Lo anterior también se puede expresar de la siguiente forma

$$p(y) = \int_{\Theta} p(y|\theta)p(\theta)d\theta.$$

A medida que se obtienen datos sobre el fenómeno es necesario incorporarlos al modelo, lo cual se logra mediante

$$p(y|\mathbf{x}) = \int_{\Theta} p(y, \theta|\mathbf{x})d\theta,$$

que describe el conocimiento de la observación futura basado en la información que se adquiere de los datos y la que contiene $p(\theta)$. Esta función corresponde a la distribución predictiva final (o *a posteriori*) y es equivalente a

$$\begin{aligned} p(y|\mathbf{x}) &= \int_{\Theta} p(y|\theta)p(\theta|\mathbf{x})d\theta \\ &= \int_{\Theta} p(y|\theta, \mathbf{x})p(\theta|\mathbf{x})d\theta. \end{aligned}$$

Si y y $\mathbf{x}$ son condicionalmente independientes dado θ

$$p(y|\mathbf{x}) = \int_{\Theta} p(y|\theta)p(\theta|\mathbf{x})d\theta.$$

Para hacer predicción puntual sobre observaciones futuras del fenómeno aleatorio es posible utilizar,

$$\hat{y} := E(Y|\mathbf{x}) = \int_{\mathcal{Y}} y\, p(y|\mathbf{x})dy.$$

El problema del constraste de hipótesis se resuelve de manera análoga al problema de estimación puntual de θ.

$$P(H_j|\mathbf{x}) = P(Y \in \mathcal{Y}_j|\mathbf{x}) = \int_{\mathcal{Y}_j} p(y|\mathbf{x})dy \quad j = 1, \ldots, k,$$

donde $\mathcal{Y}$ es el soporte de Y y $\mathcal{Y}_1, \mathcal{Y}_2, ..., \mathcal{Y}_k$ es una partición de $\mathcal{Y}$. Una vez calculada cada probabilidad, se puede elegir la hipótesis con mayor probabilidad.

Las distribuciones de probabilidad definidas anteriormente constituyen el modelo general de la estadística Bayesiana. Cualquier problema estadístico bajo el enfoque Bayesiano implica el uso de ellas.

Capítulo 3

Métodos de simulación

3.1. Simulación

Aunque la construcción de modelos comienza en el Renacimiento, la simulación científica se remonta a los trabajos de Student para determinar la distribución de la variable "t" que lleva su nombre. Esta disciplina apareció más tarde como una técnica numérica gracias a los científicos Von Neuman y Ulam que trabajaron en el proyecto de Monte Carlo durante la Segunda Guerra Mundial. Resolvieron problemas de reacciones nucleares en los que la solución experimental era muy clara, mientras que el análisis matemático era demasiado complicado. Con el uso de la computadora y, por consiguiente, el mejoramiento de los lenguajes de programación en los experimentos de simulación, surgieron incontables aplicaciones y, con ello, una mayor cantidad de problemas teóricos y prácticos. Tal es el caso de los modelos Bayesianos.

La simulación es una técnica numérica en la que se pretende imitar un proceso del mundo real o de un sistema a través del tiempo, principalmente por medio de la computadora. Para simular estos procesos o sistemas es necesario establecer relaciones matemáticas y lógicas, que permitan describir y predecir su comportamiento. Este capítulo se enfoca en la simulación estocástica cuya característica principal es que involucra procesos aleatorios para estudiar el comportamiento de un problema. Dentro de la simulación estocástica, se encuentra la simulación basada en las técnicas de Monte Carlo, las cuales son útiles para estudiar procesos estocásticos, sistemas determinísticos que involucran modelos analíticos muy complicados y problemas determinísticos de muchas dimensiones cuya discretización no es factible, como ocurre con pro-

blemas de integración. Como se ha mencionado, los avances computacionales, así como el mejoramiento de los lenguajes de programación han permitido el desarrollo de las técnicas de simulación y por ende, la simulación Monte Carlo basada en cadenas de Markov.

3.1.1. Método Monte Carlo

El método Monte Carlo se utiliza para obtener una aproximación numérica de integrales cuyo valor no es inmediato. La idea básica es calcular una integral en términos del valor esperado de alguna función con respecto a alguna distribución de probabilidad. Para ilustrar el método, suponga que desea calcular la integral de una función $f(x)$, continua en el intervalo $[a, b]$; es decir

$$I = \int_a^b f(x)dx.$$

Para encontrar un valor aproximado a I se requiere de una función de densidad $g(x)$, definida en $[a, b]$. Entonces

$$I = \int_a^b f(x)dx = \int_a^b f(x)\frac{g(x)}{g(x)}dx.$$

A través de esta expresión se reconoce el valor esperado de la función $h(x) = f(x)/g(x)$, por medio del cual se obtiene el valor de I, es decir

$$I = \int_a^b f(x)\frac{g(x)}{g(x)}dx = \int_a^b h(x)g(x)dx = E\left[h(x)\right].$$

Para obtener este valor esperado es posible simular variables aleatorias independientes e idénticamente distribuidas $X_1, X_2, ..., X_n$, con función de densidad $g(x)$. Por la ley fuerte de los grandes números,

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} h(X_i) = I \quad \text{c.s.}$$

Por lo tanto,

$$\hat{I} = \frac{1}{n}\sum_{i=1}^{n} h(x_i)$$

es un estimador insesgado y consistente de I.

3.2. Métodos Computacionales vía Cadenas de Markov

Una cadena de Markov es un proceso estocástico $\left\{\boldsymbol{\theta}^t : t \in T\right\}$ tal que, dado el estado presente, el pasado y el futuro del proceso son independientes. En otras palabras,

$$\Pr\left(\boldsymbol{\theta}^{(n+1)} \in E \mid \boldsymbol{\theta}^{(n)} \in E_n, \boldsymbol{\theta}^{(n-1)} \in E_{n-1}, \ldots, \boldsymbol{\theta}^{(0)} \in E_0\right)$$

$$= \Pr\left(\boldsymbol{\theta}^{(n+1)} \in E \mid \boldsymbol{\theta}^{(n)} \in E_n\right)$$

para todos los conjuntos $E_0, \ldots, E_{n-1} \subset \sigma(S)$, donde S denota el espacio de estados de la cadena y $\sigma(S)$ es un σ-álgebra de subconjuntos de S. Para consultar más detalles respecto a la teoría de Cadenas de Markov, ver el apéndice A.1.

Los métodos Monte Carlo vía cadenas de Markov (MCMC) permiten generar, de manera iterativa, observaciones de distribuciones multivariadas que difícilmente podrían simularse utilizando métodos directos. La idea básica es construir una cadena de Markov que sea fácil de simular y cuya distribución de equilibrio corresponda a la distribución final de interés; de tal forma que los métodos MCMC requieren que una cadena sea

- Homogénea, es decir que $\Pr(\boldsymbol{\theta}^{(n+1)} \in E \mid \boldsymbol{\theta}^{(n)} \in E_n)$ no depende de n.
- Irreducible, desde cualquier estado se puede acceder a otro, todos los estados se comunican entre sí.
- Aperiódica.
- Reversible.

Es importante mencionar que la propiedad de reversibilidad asegura la existencia de una cadena ergódica. Cuando el espacio de estados E es numerable, esta propiedad se sigue del hecho de que la cadena sea homogénea en el tiempo, irreducible y aperiódica. Sin embargo, cuando el espacio de estados E no es numerable que es el caso más común en aplicaciones del algoritmo de Metropolis-Hastings y muestreo de Gibbs existen otras consideraciones, consultar Robert y Casella (1999).

Sea $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}$,... una cadena de Markov homogénea, irreducibe y aperiódica, con espacio de estados Θ y distribución de equilibrio $p(\boldsymbol{\theta}|\boldsymbol{x})$. Entonces, conforme $t \to \infty$

(i) $\boldsymbol{\theta}^t \to^{\mathcal{D}} \boldsymbol{\theta}$, donde $\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\boldsymbol{x})$;

(ii) $\frac{1}{t}\sum_{i=1}^{t} g(\boldsymbol{\theta}^{(i)}) \to E(g(\boldsymbol{\theta})|\boldsymbol{x})$ c.s.

Una cuestión importante de orden práctico es que, a pesar de que los valores iniciales influyen en el comportamiento de la cadena, conforme el número de iteraciones aumenta, los valores iniciales pierden importancia y el proceso converge a una distribución de equilibrio. Por eso, en la aplicación es común que las iteraciones iniciales sean descartadas, a este proceso se le llama *periodo de calentamiento*. El problema por lo tanto consiste en construir algoritmos que generen cadenas de Markov cuya distribución de equilibrio coincida con la distribución de interés. A continuación se presentan el algoritmo de Metropolis-Hastings y el algoritmo de Gibbs, los cuales abordan este problema.

3.2.1. Metropolis Hastings

Este algoritmo permite construir una cadena de Markov al definir las probabilidades de transición de la siguiente manera.

Sea $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta})$ una distribución de transición (arbitraria) y se define

$$\alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \text{mín}\left\{\frac{p(\boldsymbol{\theta}^*|\boldsymbol{x})Q(\boldsymbol{\theta}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}|\boldsymbol{x})Q(\boldsymbol{\theta}^*|\boldsymbol{\theta})}, 1\right\}.$$

La idea es generar un valor de una distribución auxiliar y aceptarla con una probabilidad dada. Este mecanismo de corrección garantiza la convergencia de la cadena a su distribución de equilibrio. Suponga que la cadena está en el estado $\boldsymbol{\theta}$ y se genera un valor $\boldsymbol{\theta}^*$ de una distribución propuesta $Q(\cdot|\boldsymbol{\theta})$. Un nuevo valor $\boldsymbol{\theta}^*$ es aceptado con probabilidad α. La cadena puede permanecer en un mismo estado a lo largo de muchas iteraciones. En la práctica, se acostumbra monitorear esto para calcular el porcentaje medio de iteraciones para los cuales los nuevos valores se aceptan.

El algoritmo de Metropolis-Hastings se especifica de la siguiente forma:
Dado un valor inicial $\boldsymbol{\theta}^{(0)}$, la t-ésima iteración consiste en:

1. generar una observación $\boldsymbol{\theta}^*$ de $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{(t)})$;
2. generar una variable $u \sim U(0,1)$
3. si $u \leq \alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta}^{(t)})$, hacer $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^*$; en caso contrario, hacer $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)}$.

Este procedimiento genera una cadena de Markov con distribución de transición

$$P(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) = \alpha(\boldsymbol{\theta}^{(t+1)}, \boldsymbol{\theta}^{(t)})Q(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}).$$

La probabilidad de aceptación $\alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta})$ sólo depende de $p(\boldsymbol{\theta}|x)$ a través de un cociente, por lo que la constante de normalización no es necesaria.[1] En la práctica, frecuentemente se utiliza alguno de los dos siguientes casos:

- *Caminata aleatoria*. Sea $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}) = Q_1(\boldsymbol{\theta}^* - \boldsymbol{\theta})$, donde $Q_1(\cdot)$ es una densidad de probabilidad simétrica centrada en el origen. Entonces,

$$\alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \text{mín}\left\{\frac{p(\boldsymbol{\theta}^*|\boldsymbol{x})}{p(\boldsymbol{\theta}|\boldsymbol{x})}, 1\right\}.$$

- *Independencia*. Sea $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}) = Q_0(\boldsymbol{\theta}^*)$, donde $Q_0(\cdot)$ es una densidad de probabilidad sobre Θ. Por lo tanto,

$$\alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \text{mín}\left\{\frac{w(\boldsymbol{\theta}^*)}{w(\boldsymbol{\theta})}, 1\right\},$$

  con $w(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|x)Q_0(\boldsymbol{\theta})$.

En la práctica es común utilizar, después de una reparametrización apropiada, distribuciones de transición normales ó t de Student ligeramente sobredispersas,

$$Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}) = N_d(\boldsymbol{\theta}^*|\boldsymbol{\theta}, \kappa V(\hat{\boldsymbol{\theta}})) \qquad \text{(caminata aleatoria)}$$

$$Q_0(\boldsymbol{\theta}^*) = N_d(\boldsymbol{\theta}^*|\hat{\boldsymbol{\theta}}, \kappa V(\hat{\boldsymbol{\theta}})) \qquad \text{(independencia)},$$

donde $\hat{\boldsymbol{\theta}}$ y $V(\hat{\boldsymbol{\theta}})$ denotan a la media y a la matriz de varianzas-covarianzas de la aproximación asintótica normal para $p(\boldsymbol{\theta}|x)$, respectivamente, y $\kappa \geq 1$ es un factor de sobredispersión. Este algoritmo se ilustra con el siguente ejemplo.

[1] La versión original del algoritmo de Metropolis toma a Q tal que $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}) = Q(\boldsymbol{\theta}|\boldsymbol{\theta}^*)$, en cuyo caso:

$$\alpha(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \text{mín}\left\{\frac{p(\boldsymbol{\theta}^*|\boldsymbol{x})}{p(\boldsymbol{\theta}|\boldsymbol{x})}, 1\right\}$$

3.2.2. Ejemplo

Suponga que desea simular una distribución gamma con parámetros $\alpha > 0$ y $\beta > 0$.

$$p(\theta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)}\theta^{\alpha-1}\exp\left\{-\beta\theta\right\}, \quad \text{si} \quad \theta > 0.$$

Como se mencionó anteriormente, se acostumbra hacer una reparametrización, en este caso con el fin de lograr una mejor aproximación normal. Se aplica el cambio de variable $\eta = \log(\theta)$

$$p(\eta) \propto \exp\left\{\alpha\eta - \beta e^{\eta}\right\}.$$

A continuación se obtienen los estimadores para los parámetros de la distribución normal.

$$\log p(\eta) = \alpha\eta - \beta e^{\eta}.$$

Estimador para la media:

$$\frac{\partial \log p(\eta)}{\partial \eta} = \alpha - \beta e^{\eta}$$

$$\frac{\partial \log p(\eta)}{\partial \eta} = 0 \qquad \Leftrightarrow \qquad \hat{\eta} = \log\left(\frac{\alpha}{\beta}\right).$$

Estimador para la varianza:

$$\frac{\partial^2 \log p(\eta)}{\partial \eta^2} = \beta e^{\eta} \qquad \Rightarrow \qquad \sum(\hat{\eta}) = \frac{1}{\alpha}.$$

El siguiente código de R corresponde a la simulación de una distribución gamma con parámetros $\alpha = 2$, $\beta = 5$, utilizando el caso de independencia discutido anteriormente.

1. Especificar un vector inicial $\boldsymbol{\eta}_0$, cada $\eta_{0_i} \sim N(\hat{\eta}, \sum(\hat{\eta}))$, $i = 1, \ldots, n$.
2. Generar un vector $\boldsymbol{\eta}_1$, cada $\eta_{1_i} \sim N(\hat{\eta}, \sum(\hat{\eta}))$, $i = 1, \ldots, n$.
3. Calcular $\boldsymbol{\alpha} = \frac{\boldsymbol{p(\eta_1)Q(\eta_0)}}{\boldsymbol{p(\eta_0)Q(\eta_1)}}$.
4. Generar un vector $\boldsymbol{u}$, donde cada $u_i \sim U(0,1)$, $i = 1, \ldots, n$
   $\eta_{0i}^{(t+1)} = \eta_{0i}^{(t)}$ si $u_i < p_i$
   $\eta_{0i}^{(t+1)} = \eta_{1i}^{(t)}$ si $u_i > p_i$

5. Incrementar el contador de t para $t+1$ y volver al paso 2.

6. Vector de observaciones de la distribución gamma, $\boldsymbol{\theta} = e^{\boldsymbol{\eta}_0^{(t+1)}}$ donde $e^{\boldsymbol{\eta}_0^{(t+1)}} = (e^{\eta_{0,1}^{(t+1)}}, \ldots, e^{\eta_{0,n}^{(t+1)}})$.

```
# Metropolis-Hastings para la distribución Gamma

# Parámetros de la distribución Gamma

alfa = 2
beta = 5

# Kernel de la densidad de eta

deneta = function(eta,a,b){  exp(a*eta - b * exp(eta) )  }

#  Algoritmo de Metropolis-Hastings
N = 10000
IT = 11000

# Estimador para la media y la varianza
etah = log(alfa/beta)
setah = 1/alfa

eta0 = rnorm(N,etah,sqrt(setah))
etam = 1:IT

# Iteraciones del algoritmo de Metropolis-Hastings
for(i in 1:IT)
{
 eta1 = rnorm(N,etah,sqrt(setah))
 w1 =deneta(eta1,alfa,beta)/dnorm(eta1,etah,sqrt(setah))
 w0 =deneta(eta0,alfa,beta)/dnorm(eta0,etah,sqrt(setah))
 prob = w1/w0
 u = runif(N,0,1)
 aux = ifelse(u < prob,eta1,eta0)
 eta0 = aux
 etam[i] = mean(eta0)
}
theta = exp(eta0)
```

```
# Gráficas y diagnóstico
 c(mean(theta),var(theta))
#[1] 0.40110350 0.07964198

 ITB = trunc(IT/10)
 etame = etam[(ITB+1):IT]
 etameg = cumsum(etame)/(1:(IT-ITB))

 win.graph()
 par(mfrow=c(2,2))
 plot(etameg,type="l",font.main=6, xlab=IT-ITB,main="Promedios ergódicos")
 frame()
 hist(theta, font.main=6, ylab="Frecuencia", main="Histograma de theta")
 plot(density(theta,width=0.5,from=0, to=(mean(theta)+3*sqrt(var(theta)))),
 + xlab="theta",ylab="p",type="l",font.main=6,main="Distribución de theta")
```

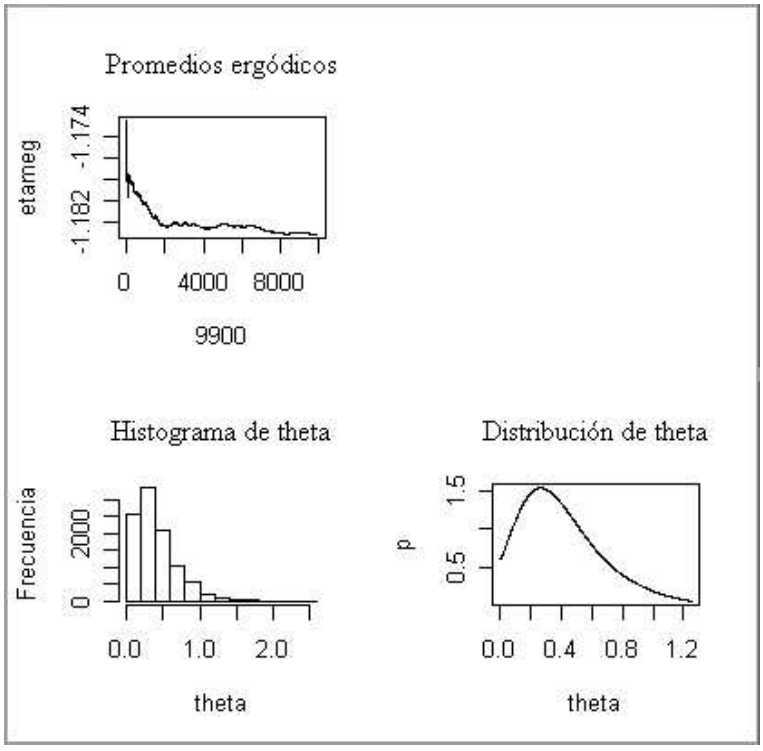


Figura 3.1: *Gráfica de la distribución gamma*

3.2.3. Algoritmo de Gibbs

El algoritmo de Gibbs permite simular una cadena de Markov $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ con distribución de equilibrio $p(\boldsymbol{\theta} \mid x)$. Cada valor nuevo de la cadena se obtiene al generar muestras de distribuciones cuya dimensión es menor que d y que en la mayoría de los casos tiene una forma más sencilla que la de $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

Sea $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$ una partición del vector $\boldsymbol{\theta}$, donde $\boldsymbol{\theta}_i \in \Re^{d_i}$ y $\sum_{i=1}^{n} d_i = d$. En este caso $\boldsymbol{\theta}_i$ es un vector, pero en general cada componente $\boldsymbol{\theta}_i$ puede ser un escalar, un vector o una matriz. Las siguientes densidades para cada $\boldsymbol{\theta}_i$ son conocidas como *densidades condicionales completas*

$$
\begin{aligned}
& p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n, \boldsymbol{x}) \\
& \qquad \vdots \\
& p(\boldsymbol{\theta}_i \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_{i+1}, \ldots, \boldsymbol{\theta}_n, \boldsymbol{x}) \qquad i = 2, \ldots, n-1 \\
& \qquad \vdots \\
& p(\boldsymbol{\theta}_n \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{n-1}, \boldsymbol{x})
\end{aligned}
$$

Estas densidades pueden identificarse fácilmente al inspeccionar la forma de la distribución final $p(\boldsymbol{\theta} \mid \boldsymbol{x})$. De hecho para cada $i = 1, 2, ..., n$,

$$
p(\boldsymbol{\theta}_i \mid \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}_{i+1}, \ldots, \boldsymbol{\theta}_n, \boldsymbol{x}) \propto p(\boldsymbol{\theta} \mid \boldsymbol{x}),
$$

donde $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ es vista sólo como función de $\boldsymbol{\theta}_i$.

Dado un valor inicial $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\theta}_1^{(0)}, \ldots, \boldsymbol{\theta}_n^{(0)})$, el algoritmo de Gibbs simula una cadena de Markov en la que $\boldsymbol{\theta}^{(t+1)}$ se obtiene a partir de $\boldsymbol{\theta}^{(t)}$ de la siguiente manera:

generar una observación $\boldsymbol{\theta}_1^{(t+1)}$ de $p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_2^{(t)}, \ldots, \boldsymbol{\theta}_n^{(t)}, \boldsymbol{x})$;
generar una observación $\boldsymbol{\theta}_2^{(t+1)}$ de $p(\boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_1^{(t+1)}, \boldsymbol{\theta}_3^{(t)}, \ldots, \boldsymbol{\theta}_n^{(t)}, \boldsymbol{x})$;
generar una observación $\boldsymbol{\theta}_3^{(t+1)}$ de $p(\boldsymbol{\theta}_3 \mid \boldsymbol{\theta}_1^{(t+1)}, \boldsymbol{\theta}_2^{(t+1)}, \boldsymbol{\theta}_4^{(t)}, \ldots, \boldsymbol{\theta}_n^{(t)}, \boldsymbol{x})$;

$\vdots$

generar una observación $\boldsymbol{\theta}_n^{(t+1)}$ de $p(\boldsymbol{\theta}_n \mid \boldsymbol{\theta}_1^{(t+1)}, \boldsymbol{\theta}_2^{(t+1)}, \boldsymbol{\theta}_3^{(t+1)}, \ldots, \boldsymbol{\theta}_{n-1}^{(t+1)}, \boldsymbol{x})$.

La sucesión así obtenida $\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots$ es una realización de una cadena de Markov cuya distribución de transición está dada por:

$$p\left(\boldsymbol{\theta}^{(t+1)} \mid \boldsymbol{\theta}^{(t)}\right) = \prod_{i=1}^{n} p\left(\boldsymbol{\theta}_i^{(t+1)} \mid \boldsymbol{\theta}_1^{(t+1)}, \ldots, \boldsymbol{\theta}_{i-1}^{(t+1)}, \boldsymbol{\theta}_{i+1}^{(t)}, \ldots, \boldsymbol{\theta}_n^{(t)}, \boldsymbol{x}\right).$$

Para ilustrar el funcionamiento de este algoritmo se realizará la simulación de una normal bivariada.

3.2.4. Ejemplo

Sea la distribución de (X, Y) una normal bivariada, cuya distribución condicional para X dado un valor fijo de $Y = y$ es

$$p(x|y) = \frac{1}{\sqrt{2\pi}\sigma_x\sqrt{1-\rho^2}} e^{-\frac{1}{2\sigma_x^2(1-\rho^2)}[x-\mu_x-\frac{\rho\sigma_x}{\sigma_y}(y-\mu_y)]^2},$$

la cual representa una función de densidad normal univariada, con media $\mu_x - \frac{\rho\sigma_x}{\sigma_y}(y-\mu_y)$ y varianza $\sigma_x^2(1-\rho^2)$. La densidad condicional de Y dado X es análoga.

El procedimiento consiste en generar un nuevo valor de X ó Y a partir de sus respectivas funciones de distribución condicional. Dado un valor inicial de $X = x_0$, el algoritmo simula una cadena de Markov que se obtiene de la siguiente forma:

genera una observación y_1 de $p(y|x_0)$;
genera una observación x_1 de $p(x|y_1)$;

$$\vdots$$

genera una observación y_n de $p(y|x_{n-1})$;
genera una observación x_n de $p(x|y_n)$.

A continuación se muestra el código de R que permite simular la distribución normal bivariada y producir las gráficas de los promedios ergódicos. Note que después de un número suficientemente grande de iteraciones éstos convergen a la media de X y Y, respectivamente. Observe la Figura 3.2.

```
gibbs<-function (n, mx,my,sx,sy,rho)
{
# n: número de simulaciones
# mx,my: media de cada distribución condicional
# sx,sy: varianza de la distribución condicional de x e y
```

```
# rho. coeficiente de correlación

    mat <- matrix(ncol = 2, nrow = n)
    xb <- 0
        for (i in 1:n) {
        x <- rnorm(1, mx+rho *(sx/sy)*( y-my),sx* sqrt(1 - rho^2))
        y <- rnorm(1,my+ rho *(sy/sx)*( x-mx),sy* sqrt(1 - rho^2))
  mat[i, ] <- c(x,y)
        }
        mat
}

 bvn<-gibbs(10000,2,5,6,5,0.5)
 IT=10000
 ITB = trunc(IT/10)
 nx=bvn[,1]
 ny=bvn[,2]

# Se descartan las primeras 1000 iteraciones (periodo de calentamiento)
 Nx = nx[(ITB+1):IT]
 Ny= ny[(ITB+1):IT]
 normx = cumsum(Nx)/(1:(IT-ITB))
 normy = cumsum(Ny)/(1:(IT-ITB))
 par(mfrow=c(3,2))
 plot(normx,type="l", font.main=6, main="Promedios ergódicos de x",
 + xlab="IT-ITB")

 plot(normy,type="l", font.main=6, main="Promedios ergódicos de y",
 + xlab="IT-ITB")

 hist(Nx, font.main=6, main="Función de densidad de x", xlab="x",
 + ylab="Frecuencia")

 hist(Ny,font.main=6,main="Función de densidad de y",xlab="y",
 + ylab="Frecuencia")

 plot(density(Nx,width=1.5,from=( mean(Nx)-3*sqrt(var(Nx)) ),
 + to=( mean(Nx)+3*sqrt(var(Nx)) )), xlab="x",ylab="p", type="l",
 + font.main=6, main="Función de densidad de x")
```

```
plot(density(Ny,width=1.5,from=( mean(Ny)-3*sqrt(var(Ny)) ),
+ to=( mean(Ny)+3*sqrt(var(Ny)) )), xlab="y",ylab="p", type="l",
+ font.main=6, main="Función de densidad de y")
```

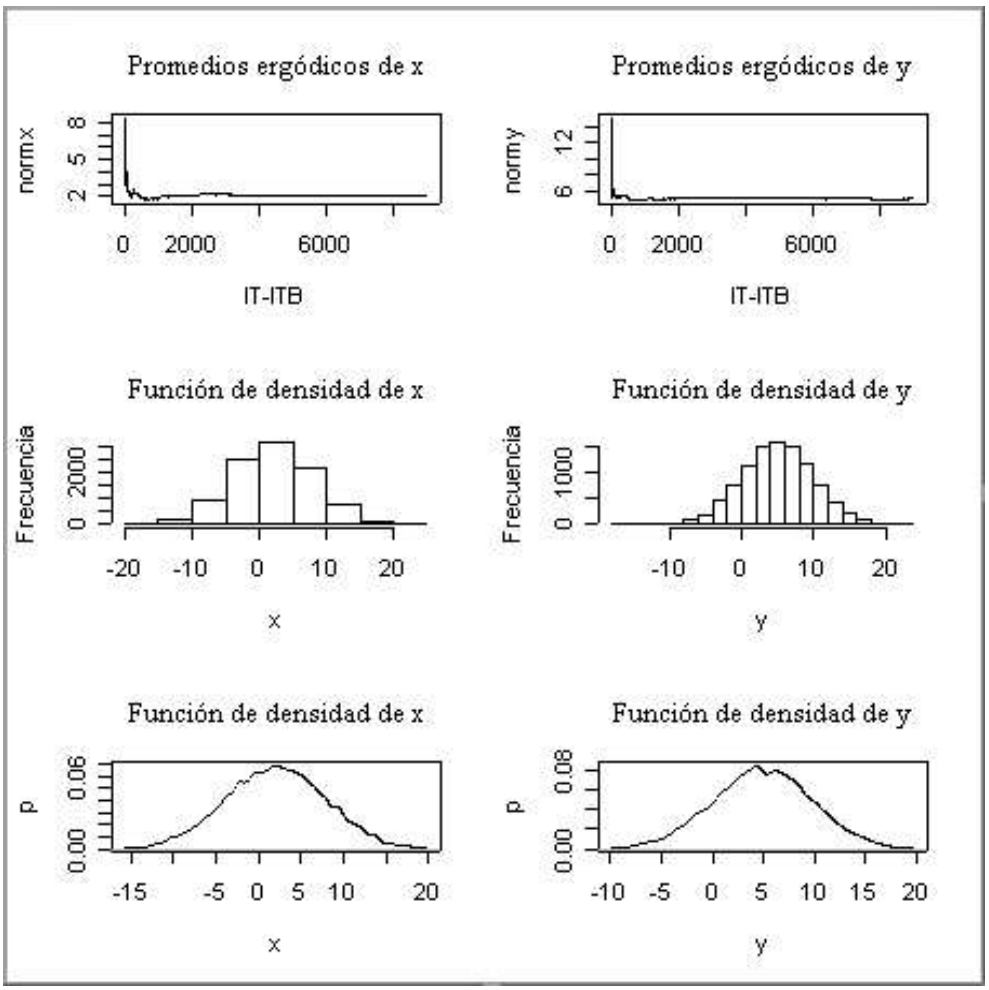


Figura 3.2: *Gráfica normal bivariada*

Capítulo 4

WinBUGS

Una de las dificultades del análisis Bayesiano es determinar la distribución final de los parámetros de interés; para algunos modelos hay cantidades que se pueden aproximar o incluso calcular analíticamente, pero son muy pocos y resultan ser los modelos más simples. El desarrollo y mejoramiento de las técnicas Monte Carlo basadas en cadenas de Markov ha hecho posible construir la distribución final de modelos Bayesianos muy complejos. WinBUGS proporciona herramientas simples para llevar a cabo la simulación Monte Carlo; a través de WinBUGS, es posible implementar el muestreo de Gibbs para una gran variedad de modelos.

Algunos de los modelos que se pueden analizar en WinBUGS son aquellos que incluyen variables latentes, las cuales son variables que no son directamente observadas por el investigador y que usualmente constituyen un artificio probabilístico para estructurar un modelo. Además, en WinBUGS todos los parámetros son tratados como variables aleatorias, lo que significa que en un modelo se define la distribución conjunta de todas las cantidades no observadas (parámetros y datos faltantes) y las cantidades observadas (los datos). Posteriormente se condiciona en los datos observados para obtener la distribución final de los parámetros y de los datos no observados. Por otra parte, en WinBUGS se pueden analizar algunos modelos no paramétricos, problemas típicos como modelos mixtos con efectos temporales y espaciales, modelos lineales, modelos lineales generalizados y modelos jerárquicos. Estos últimos se adaptan perfectamente al algoritmo de Gibbs, debido a que las distribuciones condicionales necesarias presentan usualmente formas conocidas y sencillas de implementar.

Es importante resaltar que WinBUGS provee herramientas para construir

una gran variedad de modelos y que resulta relativamente simple llevar a cabo esta construcción para los usuarios de la estadística Bayesiana, debido a la sencillez del lenguaje y a la forma de implementar los modelos.

4.1. Necesidad de contar con un programa genérico

Como se ha mencionado anteriormente, generar una muestra a través del algoritmo de Gibbs para obtener la distribución final de un modelo Bayesiano puede ser una tarea muy compleja, especialmente para el caso en el que se introducen muchos parámetros en un modelo. Por otro lado, implementar algoritmos de Gibbs en algún lenguaje convencional como C o FORTRAN puede llevar mucho tiempo. Por fortuna, WinBUGS provee implementaciones del muestreo de Gibbs relativamente sencillas.

Los modelos en WinBUGS se pueden traducir, casi literalmente, de la descripción escrita usual al lenguaje BUGS. Por otra parte, existe una gran variedad de ejemplos disponibles, algunos de los cuales se encuentran en el paquete y otros en línea en el sitio de *web* BUGS, que pueden servir de base para modelar su propio problema.

El *software* tiene ciertas limitaciones para detectar la convergencia, resumir las muestras y realizar diagnósticos sobre el ajuste de los modelos. Sin embargo, es posible usar los paquetes *boa* o *coda* de R junto con WinBUGS para realizar un mejor análisis con relación a estos puntos; en la sección 4.4.1 se habla más al respecto de estos paquetes.

Una característica de WinBUGS es que funciona como un sistema experto; es decir, intenta usar el método de muestreo más apropiado para cada nodo estocástico[1]. Además, WinBUGS puede ser extensible, es decir usted puede incorporar tanto nuevas distribuciones o funciones al sistema, como técnicas de muestreo MCMC e interfaces que le permitirán realizar el muestreo más eficiente de modelos específicos. Estas son algunas razones por las que WinBUGS alivia la necesidad de contar con un programa genérico.

[1] WinBUGS examina las circunstancias y elige una método de muestreo apropiado.

Distribución objetivo	Método de muestreo
Discreta	Inversión de una función de distribución acumulativa
Forma cerrada	Muestreo directo usando algoritmos estándar
Log-cóncava	Muestreo de rechazo adaptativo (Gilks 1992)
Rango restringido	Muestreo slice (Neal 1997)
Rango sin restricción	Metropolis-Hastings (Metropolis et al. 1953, Hastings 1970)

4.2. Descripción del paquete

4.2.1. Introducción

WinBUGS es un *software* para Windows y se puede obtener libremente en la página `http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml`.

La descripción de los modelos en WinBUGS se hace por medio del lenguaje *BUGS*, o a través de *Doodles*, que es la representación gráfica de los modelos. Sin embargo, el lenguaje *BUGS* es más flexible que el modo gráfico.

Es importante tomar en cuenta que la simulación en WinBUGS puede ser altamente ineficiente o la convergencia puede ser muy lenta para aquellos modelos cuyos parámetros estén altamente correlacionados bajo su correspondiente distribución final. De ahí la importancia de elegir una parametrización adecuada del modelo antes de analizarlo con WinBUGS. En principio, los modelos que se pueden construir en WinBUGS soportan menos de cien nodos. Sin embargo, puede obtenerse una clave al registrarse en el sitio de BUGS, mencionado anteriormente, la que le permitirá emplear modelos con más nodos. También es posible obtener un parche para actualizar el *software* a la versión 1.4.1.

En las siguentes secciones se describen los componentes y los comandos principales que ofrece WinBUGS. El la sección 4.2.11 se describe con detalle una sesión de WinBUGS para analizar un problema descrito en el Volumen I de ejemplos del manual.

4.2.2. Documentos Compuestos

Los documentos de WinBUGS contienen varios tipos de información (texto, fórmulas, tablas, gráficas, etc.) y las herramientas necesarias para crearlos y manipularlos están disponibles en el paquete. En WinBUGS un documento es la descripción de un análisis estadístico, la interacción del usuario con el *software*, y los resultados que arroje éste. Al guardar un documento se crea un archivo con extensión *.odc*, es posible especificar otra extensión.

¿Cómo se trabaja con un documento compuesto?

Un documento compuesto es similar al de un procesador de texto que contiene recuadros o elementos, los cuales se pueden manipular de manera

independiente; cada recuadro se comporta como un caracter simple y puede ser seleccionado, copiado, eliminado, etcétera.

WinBUGS trabaja con diferentes tipos de elementos, los más interesantes son los gráficos (*Doodles*) ya que permiten describir modelos estadísticos a través de nodos, placas y enlaces. Estos y otros elementos se describen por completo en la sección 4.2.9.

Editar un documento compuesto

WinBUGS contiene herramientas de un procesador de texto, las cuales pueden ser usadas para manipular cualquier resultado producido por el *software*. Si es necesario un editor con más herramientas, se pueden introducir formatos de otro procesador de texto a WinBUGS; por ejemplo, puede manejar un editor de ecuaciones a través del comando *Insert OLE Object...* del menú *Tools* (Figura 4.1).

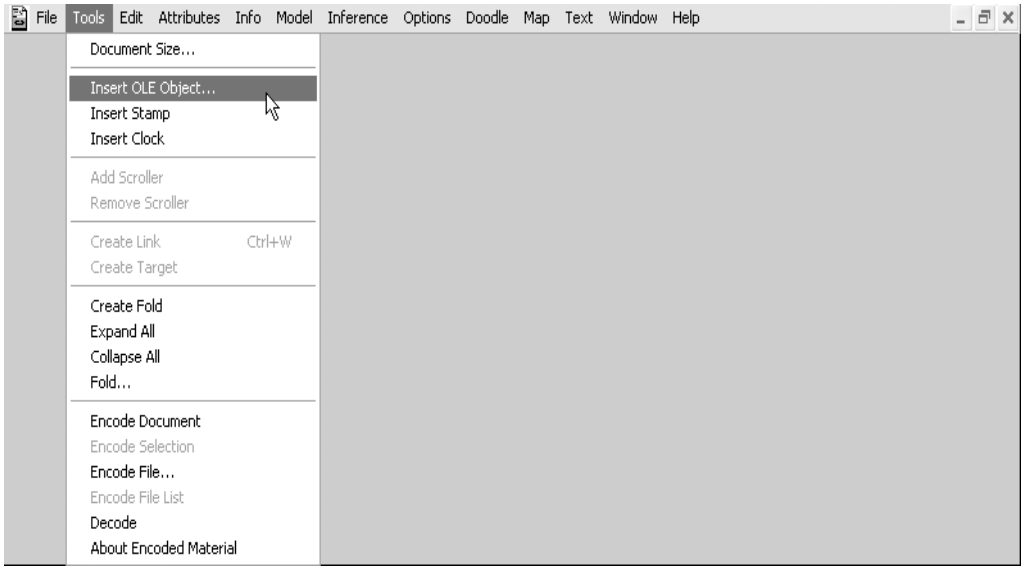


Figura 4.1: *Al dar click en Insert OLE Object... se despliega un recuadro donde aparecen las aplicaciones ejecutables, que se encuentran instaladas en su computadora.*

Si da un *click* con el botón izquierdo del *mouse* sobre un elemento, ya sea un cuadro o una gráfica, aparece un borde alrededor de éste, lo cual indica que puede modificar su tamaño; además, si ubica el *mouse* dentro de la región

rectangular puede mover el elemento. Para hacer una copia del mismo oprima la tecla *CTRL* y arrástrelo con el *mouse* al lugar donde deseé copiar, esto lo puede realizar en la ventana que se encuentra el elemento o en otra ventana de WinBUGS. La especificación del problema y el resultado del análisis del modelo los puede poner en un sólo documento, el cual puede copiar a un procesador de texto o a cualquier otro programa para diseñar presentaciones.

El estilo, tamaño, fuente y color del texto seleccionado los puede cambiar a través el menú *Attributes*. Por medio de *Text→Insert Ruler* puede desplegar una regla (Figura 4.2) que le permitirá modificar los márgenes y la alineación del texto.

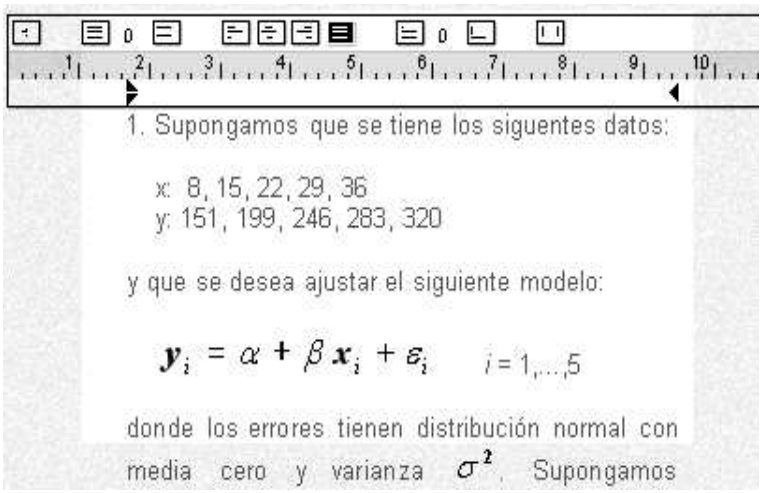


Figura 4.2: *Por medio de los iconos puede modificar la alineación del texto posterior a la regla y el margen a través de los triángulos.*

Documentos Compuestos y Correo Electrónico

Los documentos de WinBUGS contienen caracteres no-ASCII, sin embargo en el menú *Tools* se encuentra el comando *Encode Document* que produce una representación ASCII del documento. El documento original puede ser recuperado a través el comando *Decode*. Por ejemplo, esta herramienta puede ser útil para enviar los gráficos de WinBUGS a través del correo electrónico.

Imprimir un documento o un gráfico WinBUGS

Es posible imprimir directamente desde el menú *File*. Otra alternativa es copiar los documentos o gráficos a un procesador de texto e imprimirlos desde ahí.

Leer archivos de texto

Desde el menú *File* se pueden abrir archivos con formato de texto. Estos archivos pueden copiarse en un documento de WinBUGS o almacenarse como tales.

4.2.3. Distribuciones

La Tabla 4.1 presenta la lista de distribuciones disponibles en el paquete. Como puede observarse, WinBUGS permite trabajar con la mayoría de las distribuciones usadas comúnmente en las aplicaciones.

Si el modelo de interés requiere el uso de alguna distribución distinta a las presentadas en la Tabla 4.1, es posible representarla de manera aproximada; para más detalles consulte la sección 4.3.

Cuadro 4.1: Funciones de Distribución

Distribuciones Discretas Univariadas		
Distribución	Función WinBUGS	Especificación
Bernoulli	$r \sim dbern(p)$	$p^r(1-p)^{1-r}; \quad r = 0, 1$
Binomial	$ace{-}1exr \sim dbin(p, n)$	$\frac{n!}{r!(n-r)!}p^r(1-p)^{n-r}; \quad r = 0, 1, \ldots, n$
Categorial	$r \sim dcat(p[\,])$	$p[r];\ r = 1, 2, \ldots, dim(p); \quad \sum_i p[i] = 1$
Binomial Negativa	$x \sim dbin(p, n)$	$\frac{(x+r-1)!}{x!(r-1)!}p^r(1-p)^x; \quad x = 0, 1, \ldots$
Poisson	$r \sim dpois(\lambda)$	$e^{\lambda}\frac{\lambda^r}{r!}; \quad r = 0, 1, \ldots$
Distribuciones Continuas Univariadas		
Distribución	Función WinBUGS	Especificación
Beta	$r \sim dbeta(a, b)$	$p^{a-1}(1-p)^{b-1}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}; \quad 0 < p < 1$
Ji-cuadrada	$x \sim dchisqr(k)$	$\frac{2^{-\frac{k}{2}}x^{\frac{k}{2}-1}e^{\frac{-x}{2}}}{\Gamma(\frac{k}{2})}; \quad x > 0$

Distribución	Función WinBUGS	Especificación
Doble Exponencial	$x \sim ddexp(\mu, \tau)$	$\frac{\tau}{2}e^{-\tau\|x-\mu\|}; \quad -\infty < x < \infty$
Exponencial	$x \sim dexp(\lambda)$	$\lambda e^{-\lambda x}; \quad x > 0$
Gamma	$x \sim dgamma(r, \mu)$	$\frac{\mu^r x^{r-1} e^{-\mu x}}{\Gamma(r)}; \quad x > 0$
Gamma Generalizada	$x \sim gen.gamma(r, \mu, \beta)$	$\frac{\beta}{\Gamma(r)}\mu^{\beta r} x^{\beta r-1} e^{-(\mu x)^\beta}; \quad x > 0$
Log-normal	$x \sim dnorm(\mu, \tau)$	$\sqrt{\frac{\tau}{2\pi}}\frac{1}{x} exp(\frac{-\tau}{2}(\log x - \mu)^2); \quad x > 0$
Logistic	$x \sim dlogistic(\mu, \tau)$	$\frac{\tau e^{\tau(x-\mu)}}{(1+e^{\tau(x-\mu)})^2}; \quad -\infty < x < \infty$
Normal	$x \sim dnorm(\mu, \tau)$	$\sqrt{\frac{\tau}{2\pi}} exp(-\frac{\tau}{2}(x-\mu)^2); \quad -\infty < x < \infty$
Pareto	$x \sim dpar(\alpha, c)$	$\alpha c^\alpha x^{-(\alpha+1)}; \quad x > c$
t-Student	$x \sim dt(\mu, \tau, k)$	$\frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\sqrt{\frac{\tau}{k\pi}}[1 + \frac{\tau}{k}(x-\mu)^2]^{-\frac{k+1}{2}};$ $-\infty < x < \infty; \quad k \geq 2$
Uniform	$x \sim dunif(a, b)$	$\frac{1}{b-a}; \quad a < x < b$
Weibull	$x \sim dweib(v, \lambda)$	$v\lambda x^{v-1} e^{-\lambda x^v}; \quad x > 0$
Distribuciones Discreta Multivariada		
Distribución	Función WinBUGS	Especificación
Multinomial	$x[\,] \sim dmulti(p[\,], N)$	$\frac{(\sum_i x_i)!}{\prod_i x_i!}\prod_i p_i^{x_i}; \sum_i x_i = N;$ $0 < p_i < 1; \quad \sum_i p_i = 1$
Distribuciones Continuas Multivariadas		
Distribución	Función WinBUGS	Especificación
Dirichlet	$p[\,] \sim ddirch(\alpha[\,])$	$\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\prod_i p_i^{\alpha_i - 1}; \quad 0 < p_i < 1;$ $\sum_i p_i = 1$
Normal Multivariada	$x[\,] \sim dmnorm(\mu[\,], T[,])$	$(2\pi)^{\frac{-d}{2}}\|T\|^{\frac{1}{2}} exp(-\frac{1}{2}(x-\mu)'T(x-\mu));$ $-\infty < x < \infty$
t-Student Multivariada	$x[\,] \sim dmt(\mu[\,], T[,], k$	$\frac{\Gamma(\frac{k+d}{2})}{\Gamma(\frac{k}{2})k^{\frac{d}{2}}\pi^{\frac{d}{2}}}\|T\|^{\frac{1}{2}}[1 + \frac{1}{k}(x-\mu)'T(x-\mu)]^{-\frac{k+d}{2}};$ $-\infty < x < \infty$
Wishart	$x[,] \sim dwish(R[,], k)$	$\|R\|^{\frac{k}{2}}\|x\|^{\frac{k-p-1}{2}} e^{-\frac{1}{2}Tr(Rx)};$ x simétrica y definida positiva

4.2.4. Menú *Model*

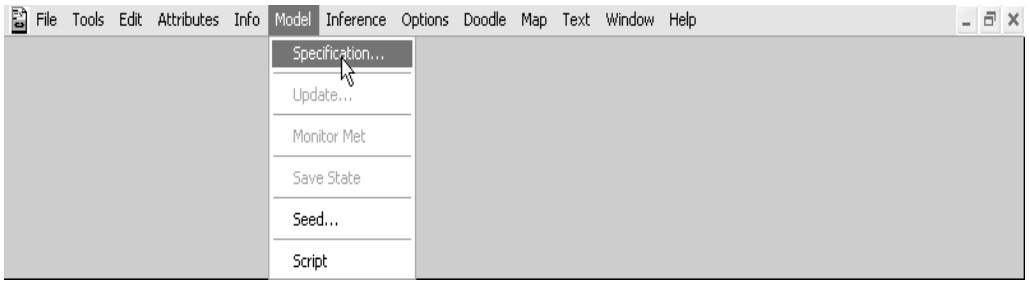


Figura 4.3: *Este menú contiene comandos que permiten verificar y compilar el modelo.*

Propiedades Generales

Los comandos de este menú podrán ser usados para especificar un modelo gráfico o en lenguaje BUGS. En la Figuras 4.4 y 4.5 puede observar la estructura de un modelo, el cual podrá ser analizado con los comandos que se describen en las siguientes secciones.

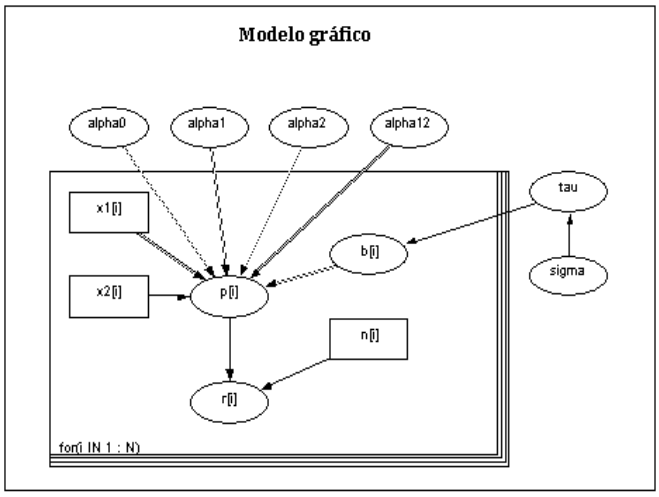


Figura 4.4: *Modelo gráfico del ejemplo* seeds *del Volumen I de ejemplos de WinBUGS.*

Modelo en lenguaje BUGS

```
model
{
        for( i in 1 : N ) {
                r[i] ~ dbin(p[i],n[i])
                b[i] ~ dnorm(0.0,tau)
                logit(p[i]) <- alpha0 + alpha1 * x1[i] + alpha2 * x2[i] +
                        alpha12 * x1[i] * x2[i] + b[i]
        }
        alpha0 ~ dnorm(0.0,1.0E-6)
        alpha1 ~ dnorm(0.0,1.0E-6)
        alpha2 ~ dnorm(0.0,1.0E-6)
        alpha12 ~ dnorm(0.0,1.0E-6)
        # Choice of priors for random effects variance
        # Prior 1: uniform on SD
        sigma~ dunif(0,100)
        tau<-1/(sigma*sigma)

        #Prior 2:
        #tau ~ dgamma(1.0E-3, 1.0E-3);
        #sigma <- 1/sqrt(tau); # s.d. of random effects
    }
Data
    list(r = c(10, 23, 23, 26, 17, 5, 53, 55, 32, 46, 10,  8, 10,  8, 23, 0,  3, 22, 15, 32, 3),
         n = c(39, 62, 81, 51, 39, 6, 74, 72, 51, 79, 13, 16, 30, 28, 45, 4, 12, 41, 30, 51, 7),
         x1 = c(0,  0, 0,  0,  0, 0,  0,  0, 0,  0,  0, 1,  1,  1,  1, 1,  1,  1,  1,  1, 1),
         x2 = c(0,  0, 0,  0,  0, 1,  1,  1, 1,  1,  1, 0,  0,  0,  0, 0,  1,  1,  1,  1, 1),
         N = 21)

Inits1      list(alpha0 = 0, alpha1 = 0, alpha2 = 0, alpha12 = 0, sigma = 1)
Inits2      list(alpha0 = 0, alpha1 = 0, alpha2 = 0, alpha12 = 0, tau = 1)
```

Figura 4.5: *Modelo en lenguaje BUGS del ejemplo* seeds *del Volumen I de ejemplos de WinBUGS.*

Specification...

check model: Permite revisar la sintaxis del modelo en cuestión. Si el modelo se encuentra especificado en lenguaje *BUGS*, seleccione la palabra `model` en la especificación del modelo (ver sección 4.2.10) y dé *click* en el botón *check model*. Si existe un error de sintaxis, el cursor muestra dónde se encuentra éste, y en la parte inferior de la pantalla, en la barra de estado, aparece la descripción del error.

Si la ventana activada contiene el gráfico del modelo y el gráfico ha sido seleccionado, WinBUGS asume que el modelo ha sido especificado gráficamente. Si existe un error de sintaxis, el nodo que presenta el error aparece sombreado y se muestra un mensaje de error en la barra de estado.

En ambos casos, si el modelo se ha especificado correctamente, se despliega el mensaje "`model is syntactically correct`". Únicamente

si el modelo está escrito en lenguaje BUGS se activan los botones *load data* y *compile*. Si el modelo es gráfico, sólo se activa el botón *compile*. El siguiente paso consiste en cargar los datos a través del botón *load data*.

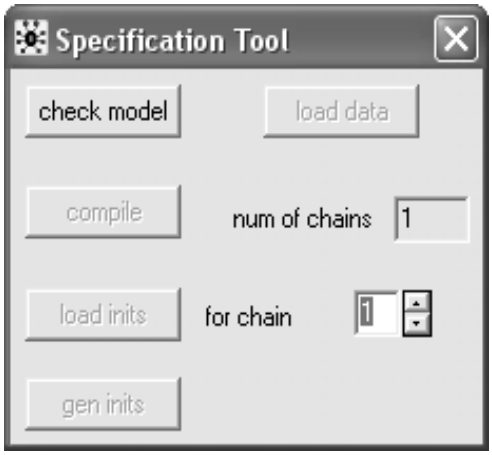


Figura 4.6: Specification*; contiene comandos para verificar, compilar, especificar datos y los valores iniciales del modelo.*

load data: Para cargar los datos que utiliza el modelo se procede de dos maneras:

1. Si los datos se encuentran en otro documento, abra los archivos que contengan los datos y dé *click* en el botón *load data*.
2. Si los datos se encuentran en el documento, éstos deben estar contenidos en una lista, `list`, seleccione la palabra `list` y dé *click* en el botón *load data*.

Ante cualquier error sintáctico o inconsistencias en los datos, se despliega un mensaje de error en la barra de estado. Después de haber realizado las correcciones pertinentes, nuevamente revise el modelo a través del botón *check model* e intente cargar los datos. Cuando éstos hayan sido cargados aparece el mensaje "`data loaded`".

num of chains: Indica el número de cadenas que desea simular. Dé *click* en el campo *num of chains* y proporcione el número de cadenas. Este cuadro se activa una vez revisado el modelo y se desactiva cuando se haya compilado el modelo. WinBUGS simula de entrada una cadena.

compile: Construye las estructuras de datos necesarias para realizar el muestreo de Gibbs. En esta parte se verifica por completo el modelo y se revisa que los datos sean consistentes con el mismo.

Se crea automáticamente un nodo llamado *deviance*[2]. Este nodo puede ser monitoreado al teclear *deviance* en la caja de diálogo *Samples* del menú *Inference* (ver sección 4.2.5). Una vez que el modelo ha sido revisado y compilado exitosamente en la barra de estado aparece el mensaje "`model compiled`".

load inits: Para cargar los valores iniciales se procede de la misma forma como lo hizo para cargar los datos del modelo, salvo que deberá presionar el botón *load inits* en lugar de *load data*. Si algunos de los elementos del arreglo de valores iniciales son desconocidos (porque son restricciones en una parametrización), estos elementos deberán ser especificados como `NA`, no disponibles.

Si WinBUGS ha cargado los valores iniciales, se despliega el mensaje "`model is initialized`" en la barra de estado; de lo contrario, se muestra el mensaje "`model contains uninitialized nodes`"; este último puede tener varios significados:

1. Si sólo se simula una cadena esto significa que la cadena contiene algunos nodos que no han sido iniciados.
2. Cuando se simulan varias cadenas significa que algunos valores iniciales no han sido cargados para alguna de ellas.

Load inits puede ser ejecutado mientras el muestreo de Gibbs se lleva a cabo, pero la muestra de salida tendrá una nueva trayectoria. Si el comando se utiliza en este contexto aparece un mensaje de advertencia "`the MCMC will start at a new point`".

gen inits: Este botón genera valores iniciales y lo hace a través de muestrear la distribución inicial o una aproximación de ésta. Para las variables discretas no se generan valores que tengan una probabilidad de cero. Este comando produce valores extremos si las distribuciones iniciales son muy vagas. Una vez generados los valores iniciales se despliega el

[2] *Deviance* se define como menos dos veces el logaritmo de la función de verosimilitud para cada iteración. En la sección 4.2.5 se habla con más detalle acerca de este nodo.

mensaje "`initial values generated: model initialized`", si no es así aparece el mensaje "`could not generate initial values`". El botón *gen inits* se activa una vez que el modelo se ha compilado exitosamente y se desactiva cuando el modelo haya sido iniciado.

Update...

Figura 4.7: *Este comando se activa cuando el modelo ha sido compilado e iniciado.*

updates: Número de iteraciones de la(s) cadena(s) simulada(s).

refresh: En *refresh* se indica el intervalo en el cual se reporta el estado de la simulación. Por ejemplo, suponga que desea simular 1000 valores; si teclea 100 en *refresh*, podrá observar en *iteration*, iteraciones de 100 en 100.

thin: Sólo se almacenan las muestras de cada *k-ésima* iteración, donde k es el valor de *thin*; $k > 1$ ayuda a reducir la autocorrelación en la muestra. Sin embargo, esta opción no proporciona una ventaja real, salvo que reduce los requerimientos de almacenamiento y el costo de manejar simulaciones cuando se llevan a cabo muchas corridas.

update: Permite ejecutar el proceso de simulación. Si da un *click* durante el muestreo, éste se detiene. Para continuar con el muestreo dé nuevamente *click*. Al interrumpir el proceso de simulación se puede cambiar el número de actualizaciones si es necesario.

iteration: Muestra el número de iteraciones simuladas. Note que no es el número de iteraciones que aparecen como muestreadas. Más bien,

iteration comprende las iteraciones que se realizan incluyendo el *periodo de calentamiento*. Sólo cuando se haya especificado que los nodos deberán ser monitoreados sin previa etapa de calentamiento, se considerarán como iteraciones muestreadas (ver sección 4.2.5). Si se generan 100 muestras a través de *update* y *thin* es igual a 10, entonces se actualizan 10*100 iteraciones y se almacena 1 cada 10 iteraciones hasta obtener las 100 muestras requeridas.

over relax: Marque esta casilla para seleccionar la forma de sobre-relajamiento del algoritmo (Neal 1998). Este método genera muestras múltiples en cada iteración y después selecciona una que está negativamente correlacionada con el valor actual. El tiempo por iteración se incrementará, pero se reducirá la correlación en la cadena y se necesitarán menos iteraciones. Sin embargo, este método no siempre es efectivo, por lo que deberá usarse con mucha precaución. La función de autocorrelación puede emplearse para verificar si la cadena que se produjo ha mejorado.

adapting: Esta casilla aparece seleccionada cuando los algoritmos de Metropolis o *slice-sampling* se encuentran en una etapa de adaptación para optimizar el cálculo de algunos parámetros. Ignore los resúmenes de estadísticas o todos los resultados que pueda producir de los parámetros mientras WinBUGS se encuentre en esta etapa de adaptación. El algoritmo de Metropolis y *slice-sampling* tiene una fase de adaptación de 4000 y 500 iteraciones, respectivamente, las cuales deberán ser descartadas para el cálculo de estadísticas.

Monitor Metropolis

Este comando se activa sólo cuando se está utilizando el algoritmo de Metropolis. Este proporciona la tasa de aceptación mínima, máxima y media para determinar el promedio de más de 100 iteraciones cuando el algortimo de Metropolis se adapta para las primeras N iteraciones. A través de este comando se puede observar una gráfica en la que la tasa debe encontrarse entre dos líneas horizontales. Las primeras N iteraciones no pueden ser usadas para hace inferencia estadística (el valor predeterminado de N es igual a 4000 iteraciones).

Save State

Muestra una ventana en la que aparece el estado actual de todas las variables estocásticas y permite guardar sus valores correspondientes. Estos resultados pueden ser usados como valores iniciales para las futuras corridas.

Seed...

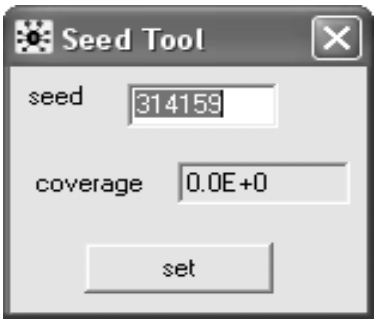


Figura 4.8: *El número que se introduce en el campo* seed *corresponde a una nueva semilla del generador de números aleatorios.*

coverage: El seudogenerador usado por WinBUGS genera una sucesión finita de números que, aunque muy larga, eventualmente se puede repetir al muestrear durante un periodo muy largo de tiempo. El campo *coverage* muestra el porcentaje que la sucesión ha cubierto durante el proceso que lleva a cabo WinBUGS.

set: Dé *click* en este botón para especificar la semilla. Puede introducir una semilla después de haber revisado el modelo, con el propósito de especificar un nuevo valor al proceso de simulación.

Script...

Genera una bitácora de la sesión de WinBUGS (para más detalles consulte la sección 4.3.1).

4.2.5. Menú *Inference*

Propiedades Generales

Este menú contiene herramientas que permiten hacer inferencia sobre los parámetros del modelo. Los comandos están divididos en tres secciones: los primeros tres están relacionados con el monitoreo de los valores de los parámetros; los siguientes dos comandos proporcionan algunas estadísticas, algunas de las cuales están incluidas en los primeros comandos; y el último comando concierne a la evaluación del *Deviance Information Criterion* (*DIC*). Antes de utilizar estas herramientas debe asegurarse que la simulación ha convergido (para más detalles acerca de convergencia consulte la sección 4.2.13). Note que la simulación MCMC comprende una fase de adaptación por lo que no es posible hacer inferencia usando los valores muestreados antes de que termine esta fase.

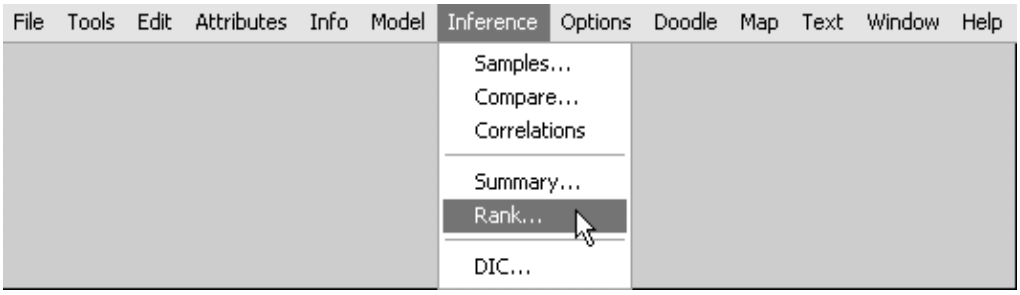


Figura 4.9: *A través de este menú se obtiene estadísticas de los parámetros simulados; además contiene herramientas para realizar gráficas.*

Sample Monitor Tool...

node: En este campo debe teclear el parámetro de interés. Si el parámetro es un vector debe introducir su nombre y especificar sus entradas. Por ejemplo, suponga que desea hacer inferencia sobre el vector $v = (v_1, v_2, ..., v_5)$; si sólo desea obtener resultados para las entradas v_1 a v_3, tecleé $v[1 : 3]$; si requiere obtener los cálculos de todos los elementos del vector introduzca $v[1 : 5]$ o simplemente v.

Cuando el parámetro de interés es una matriz, por ejemplo, suponga que $(m_{11}, m_{12}, m_{13}, m_{14}, ..., m_{44})$ son entradas de la matriz m de dimensiones 4×4, si desea obtener todas las componentes debe teclear

$m[1:4, 1:4]$ o únicamente m. Si requiere sólo una coordenada, por ejemplo m_{12} escriba $m[1, 2]$ o $m[1:2]$; por otro lado si quisiera sólo las entradas $m_{12}, m_{13}, m_{22}, m_{23}, m_{32}, m_{33}$ debe teclear $m[1:2, 2:3]$.

WinBUGS genera automáticamente un nodo determinístico para medir una cantidad identificada como *deviance*. Para conocer las estadísticas respecto a este nodo tecleé *deviance* en el campo *nodo*. *Deviance* se define como $-2\log(p(y|\theta))$, donde $p(y|\theta)$ corresponde a la función de verosimilitud, y comprende todos los nodos estocásticos observados (datos), y θ los nodos padres de y (parámetros). Los nodos padre son nodos estocásticos de los cuales depende la distribución de y.

Figura 4.10: *Tecleé " * " para obtener la información de todos los parámetros que hayan sido introducidas en el campo node.*

beg and end: Número de iteraciones que son usadas para seleccionar un subconjunto de la muestra almacenada para el análisis. Suponga que se realizan 5000 iteraciones. Si introduce *beg*=2000 y *end*=5000, la muestra para hacer inferencias va de la iteración 2000 a la 5000.

thin: Le permite seleccionar un subconjunto de la cadena generada, es decir, si genera una cadena de 1000 valores y *thin* es 10 implica que un valor cada 10 iteraciones contribuye al cálculo de las estadísticas, por lo que el número total de valores que contribuyen al cálculo es 100. Esta opción es análoga al comando *Update Tool→thin*, sin embargo no produce ningún impacto en los requerimientos de almacenamiento como al utilizar *Update Tool→thin*.

chain to: A través de este comando puede seleccionar las cadenas que deseé para realizar el cálculo de las estadísticas.

clear: Remueve el parámetro almacenado de la memoria de la computadora.

set: Registra el parámetro para el cual se construye una cadena de valores.

trace: Grafica el valor del parámetro contra el número de iteraciones. Esta gráfica es dinámica, es decir, a medida que corren las iteraciones del parámetro, los valores generados aparecen en la gráfica.

history: Muestra una gráfica del parámetro contra las iteraciones, sólo muestra las iteraciones que fueron seleccionadas en las cajas *beg* y *end*.

Los comandos que a continuación se describen aparecen desactivados si las simulación MCMC se encuentra en la fase de adaptación.

density: Dibuja una curva suavizada que estima una función de densidad si la variable es continua, o un histograma si la variable es discreta.

auto cor: Despliega la gráfica de la autocorrelación de la variable hasta un rezago (*lag*) de orden 50. Los valores que producen la gráfica aparecen en una ventana, al dar doble *click* sobre el gráfico y presionar *CTRL* más *click* con el botón derecho del *mouse*. Si se simulan muchas cadenas aparecen los valores para cada una de ellas.

stats: Produce un resumen de las estadísticas de la variable de interés, a través de las cadenas seleccionadas. Si desea calcular otro(s) percentil(es) en particular, selecciónelo(s) por medio del recuadro *percentiles*. WinBUGS proporciona por defecto la mediana y los cuantiles al 2.5 % y 97.5 % de probabilidad. La cantidad reportada en *MC error* proporciona una estimación de $\frac{\sigma}{N^{\frac{1}{2}}}$, la desviación estándar Monte Carlo para la media.

coda: Despliega dos archivos, cuyos valores podrán ser usados por S-Plus o R, a través de los paquetes *coda* o *boa*[3]. En una ventana aparecen

[3] *Coda* y *boa* son paquetes que permiten llevar a cabo un análisis de convergencia con base en la(s) cadena(s) simulada(s), ver sección 4.4. Estos paquetes se pueden encontrar en los sitios *web*:
Boa: http://cran.r-project.org/src/contrib/Descriptions/boa.html
Coda: http://cran.r-project.org/src/contrib/Descriptions/coda.html

los valores generados y el número de iteraciones (archivo *.out*). La otra ventana (archivo *.ind*) contiene información que permite identificar a qué nodo corresponde cada uno de los valores del archivo *.out*. Para que los paquetes mencionados puedan manejar estos archivos debe guardalos con la extensión *.out* y *.ind*, respectivamente. Si generó más de una cadena se despliega una ventana para cada una de ellas.

quantiles: Despliega la gráfica del intervalo al 95 % de probabilidad para las medias generadas en cada iteración.

Compare...

Figura 4.11: *Comparison Tool, facilita la comparación de las variables o nodos, con respecto a sus distribuciones finales.*

node: Introduzca el nodo que desea graficar. No olvide monitorear el nodo con anticipación, ya que la gráfica se realiza con base en la distribución final.

other: Permite definir un vector en el que cada uno de sus elementos es graficado a la par de los elementos del nodo introducido en el campo *node*, en la misma escala. Por ejemplo, *other*, pueden ser los datos observados en la gráfica producida por el botón *model fit*. Los elementos de *other* pueden ser también variables monitoreadas (en tal caso se grafican las medias finales). Esta opción permite obtener algunas gráficas similares a las que se usan para el análisis de regresión, como son las gráficas de residuales.

axis: Define un conjunto de valores contra los cuales los elementos del nodo y *other* son graficados. Cada elemento de *axis* debe ser conocido o, en

su defecto, la variable monitoreada; en este caso se usa la media final para producir la gráfica.

Recuerde que *node, other* y *axis* deben contener el mismo número de elementos.

beg y end: Permite seleccionar un subconjunto de la muestra, con el cual se realiza las gráficas deseadas.

box plot: A través de este comando puede realizar una gráfica de cajas y brazos de la distribución final de los elementos del nodo, cada uno de ellos se dibuja en el orden en el que aparecen en el nodo y también se muestra su índice. Las cajas representan los rangos intercuartiles y las líneas que las dividen son las medias. Los brazos de cada caja cubren el 95 % de la distribución, por lo que se ubican en los cuantiles 2.5 % y 97.5 %; observe la Figura 4.12.

La línea que cruza todas las cajas representa la media global de las medias finales. Una propiedad especial del editor para las gráficas de cajas es que, además de generarlas vía *Comparison Tool*, puede interactuar con ellas. Por ejemplo, es posible representar las distribuciones a través de las medias o las medianas, o bien, graficar en escala logarítmica.

caterpillar: Una gráfica *caterpillar* es similar a la gráfica de cajas. La única diferencia significativa es que no se muestra el rango intercuartil y la escala está sobre el eje de las abcisas; vea la Figura 4.12. Cada línea representa un intervalo al 95 % de probabilidad y el punto muestra la ubicación de la media. La línea vertical representa la media global. Es preferible la gráfica de *caterpillar* que la de cajas y brazos debido a su sencillez, sobre todo cuando el número de distribuciones a comparar es muy grande.

model fit: Los elementos del nodo (y *other*, si se especifica) son considerados como series de tiempo. La distribución final de cada elemento es resumida por los cuantiles 2.5 %, 50 % y 97.5 %. La línea roja representa la mediana y las líneas azules punteadas corresponden a un intervalo del 95 % de probabilidad. Asimismo, la curva corresponde al modelo ajustado; observe la Figura 4.12. En el caso de que *other* sea especificado, también se grafican sus valores.

Ambos ejes pueden expresarse en escala logarítmica a través de *property editor*.

scatterplot: Despliega una gráfica que dibuja por defecto la media final contra los correspondientes valores de *axis* y un ajuste no paramétrico basado en pesos exponenciales; observe la Figura 4.12. Para modificar algunas propiedades de esta gráfica use el editor descrito en *Scartterplot* de la sección 4.2.8. Por ejemplo, puede cambiar la suavidad de la curva que ajusta el parámetro de suavidad o reemplazarla por otro tipo de línea; también puede modificar el tamaño del intervalo.

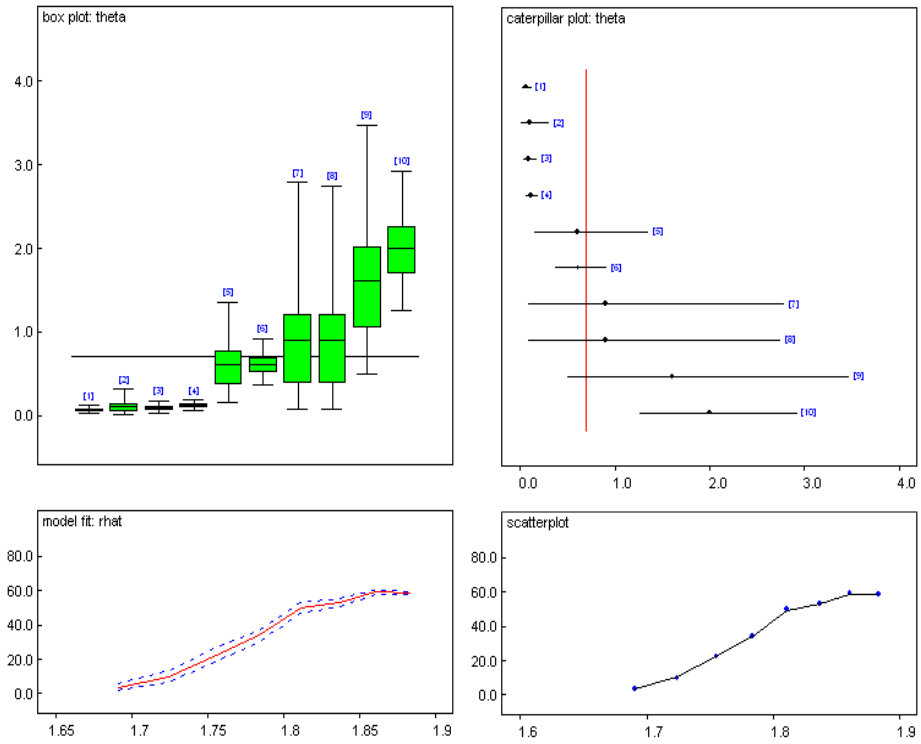


Figura 4.12: *Consulte la sección 4.2.8 si desea modificar las propiedades de cada tipo de gráfica.*

Correlations...

nodes: En este campo puede introducir escalares o arreglos y todas las combinaciones de los parámetros que seleccione. Si proporciona sólo un

arreglo, se dibujan todas las correlaciones entre sus elementos.

scatter: Produce una gráfica de dispersión de puntos para cada pareja de nodos.

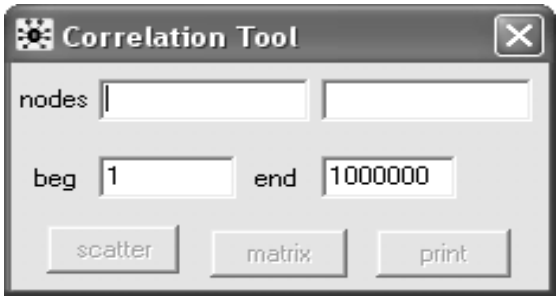


Figura 4.13: *Este comando es útil para graficar la relación entre los valores de los parámetros previamente monitoreados.*

matrix: Despliega una gráfica de la matriz de correlaciones.

print: Abre una ventana que contiene los coeficientes de todas las posibles correlaciones entre los parámetros seleccionados.

Summary...

Figura 4.14: *No es posible obtener únicamente v[1], la primera entrada de un vector v. Para obtener las estadísticas de este elemento, debe especificar el vector (v), por completo.*

A través de este cuadro de diálogo puede calcular las medias, las desviaciones estándar y los cuantiles. Los comandos que se encuentran en este recuadro son los más sencillos y también los puede encontrar en *Sample Monitor Tool*. El objetivo de considerarlos en otro apartado es que, en el

momento que haya muchos parámetros y el proceso de simulación demore, se pueda contar rápidamente con estas estadísticas ya que, a diferencia de *Sample Monitor Tool* este comando no almacena todas las iteraciones. Es importante aclarar que no es posible obtener resultados sólo para algunas entradas de un vector o de una matriz.

node: Tecleé el parámetro de interés en este campo.

set: Registra las corridas de la distribución del nodo.

stats: Despliega la media, la desviación estándar, y los cuantiles 2.5 %, 50 %(mediana) y 97.5 % del nodo.

means: Despliega las medias para el nodo, separadas por comas. Esto puede ser útil para transferir los resultados a otros paquetes estadísticos.

clear: Borra el nodo que se muestra en la caja y la muestra generada del mismo.

Rank...

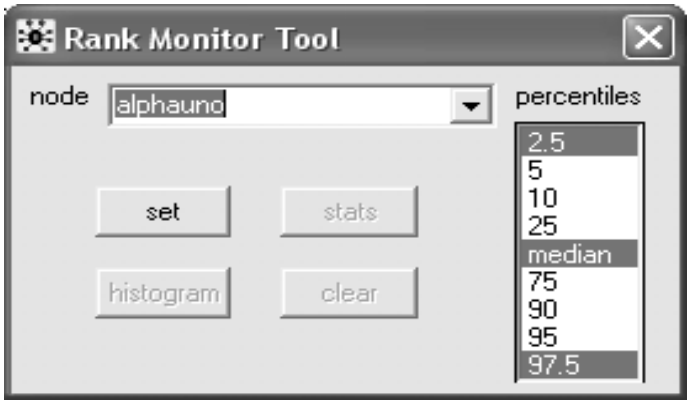


Figura 4.15: *Este comando se utiliza para almacenar los rangos de los valores simulados.*

Al igual que en el comando *Summary Monitor Tool*, no es posible obtener resultados para una parte del vector o de la matriz; debe especificarlos por completo.

node: En este campo se introduce el parámetro y a partir de éste se calculan los rangos (el parámetro debe ser un arreglo).

set: Construye los histogramas que representarán el rango de cada componente del nodo. Almacena una cantidad proporcional al cuadrado del número de componentes del nodo disponible. Aún cuando el nodo tenga miles de componentes, esto requiere menos almacenaje que calcular los rangos explícitamente en el modelo especificado y almacenar sus muestras, por lo que es más eficiente.

stats: Resume la distribución de los rangos de cada componente del parámetro *node*; además se despliegan los cuantiles marcados en la sección *percentiles*. Si desea marcar varios cuantiles presione *CTRL* y dé clik en los cuantiles que deseé.

histogram: Despliega la distribución empírica de los rangos simulados para cada componente del parámetro (*node*).

clear: Borra el nodo que se muestra en la caja y la muestra generada del mismo.

En el apéndice B se muestra una ilustración correspondiente al *Volumen II* de ejemplos de WinBUGS, *Schools: multivariate hierarchical model of examination results*, donde podrá encontrar algunas observaciones adicionales respecto al uso de la opción *Rank*.

DIC...

Figura 4.16: *Hay circunstancias, como es el caso de los modelos de mezclas, en los que WinBUGS no permite el cálculo de DIC y el comando aparece desactivado.*

La herramienta *DIC* que aparece en el cuadro de diálogo es utilizada para evaluar el *Deviance Information Criterion* y estadísticas relacionadas. Estas últimas, son empleadas para evaluar modelos complejos y comparar

diferentes modelos. Es importante tomar en cuenta que el *DIC* asume que la media final es un buen estimador para los parámetros, si esto no ocurre debido a un sesgo o a que la distribución final es bimodal, entonces el *DIC* no es apropiado.

set: Permite calcular el *DIC* y estadísticas relacionadas. Deberá asegurarse que la(s) cadena(s) ha(n) convergido antes de presionar *set* para indicar que todas los subsecuentes iteraciones serán utilizadas para el cálculo.

clear: Si ya realizó cálculos para el *DIC*, a través de esta opción puede borrarlos de la memoria. Para reiniciar los cálculos presione *set* nuevamente.

DIC: Despliega el cálculo de las estadísticas que se describen a continuación.

Dbar: Es la media final de la devianza, que es exactamente la misma que puede monitorear a través de la herramienta *Sample Monitor Tool*. La devianza es definida como $-2\log$(función de verosimilitud). La función de verosimilitud se especifica como $p(y|\theta)$, donde y comprende todos los nodos estocásticos dados los valores observados es decir, una lista de datos, y θ comprende los nodos padres de y, que son aquellos de los que depende la distribución de y.

Dhat: Este es un estimador puntual de la devianza, el cual se obtiene sustituyendo la media final, $\theta.bar$, en lugar de θ, esto es: $-2\log(p(y|\theta.bar))$.

pD: Proporciona "el número efectivo de parámetros", y está dado por $pD = Dbar - Dhat$. Así que pD es la media final de la devianza menos la devianza de la media final.

DIC: *Deviance Information Criterion*, se define como:

$$DIC = Dbar + pD = Dhat + 2pD.$$

El modelo con el menor *DIC* es considerado como el modelo que mejor podría predecir una réplica de los datos con la misma estructura que las observaciones[4].

[4]Para más detalles ver Spiegelhalter et al. (2002).

4.2.6. Menú *Info*

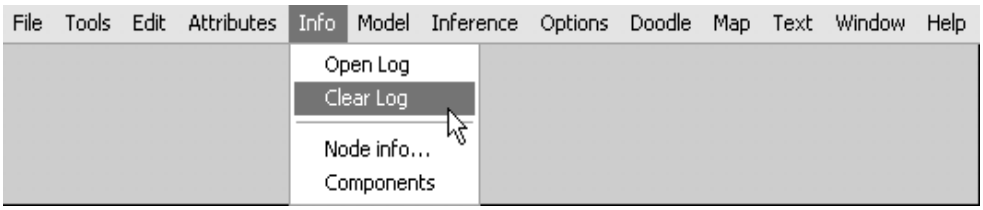


Figura 4.17: *Components despliega una lista de todas las acciones que ha llevado a cabo. Así como la fecha y la hora en la que las ejecutó.*

Propiedades Generales

Este menú le permite al usuario obtener más información acerca del funcionamiento del *software*.

Open log: Esta opción abre una ventana con registros de los errores y guarda una bitácora de la sesión.

Clear log: Permite borrar toda la información desplegada en la ventana de *log*.

Node info...

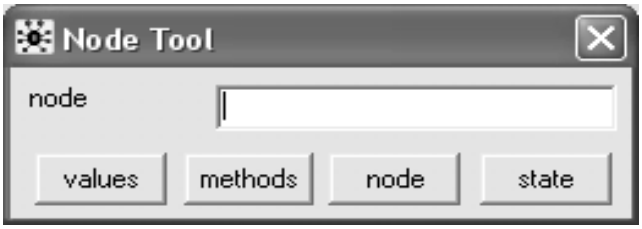


Figura 4.18: *Node Tool permite obtener información de algún nodo en particular.*

node: Tecleé el nombre del parámetro cuya información es requerida.

values: Despliega el valor actual del nodo (para cada cadena), en la ventana *log*.

methods: Despliega el tipo de algoritmo utilizado para la simulación del nodo en una ventana *log*.

node: Despliega el tipo de nodo usado para representar el parámetro en la ventana *log*.

state: Abre una ventana *trap* que muestra el estado actual de la estructura de datos usada internamente para representar el nodo. Esto es sólo para ver el proceso y no tiene ningún uso práctico para el usuario.

4.2.7. Menú *Options*

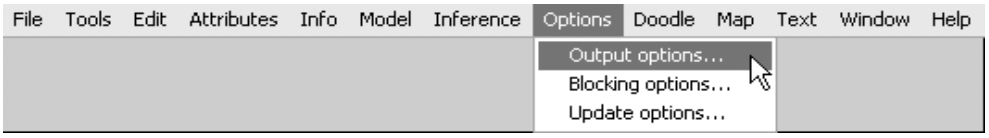


Figura 4.19: *Menú* Options

Output options

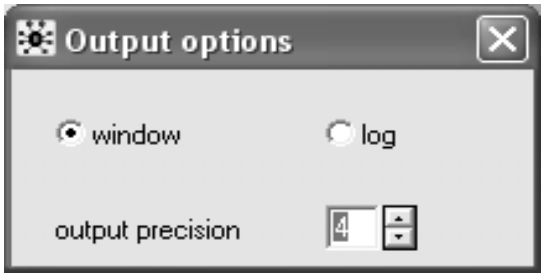


Figura 4.20: *Output precision permite elegir el número de dígitos que se mostrarán en los resultados que arroje WinBUGS.*

window o log: Si selecciona *window* se abre una nueva ventana para cada aplicación ejecutada (estadísticas, gráficas, etc.); por otra parte si selecciona *log* entonces todos los resultados se despliegan en una sola ventana.

Blocking options...

Figura 4.21: *Esta casilla aparece seleccionada por defecto.*

Si la casilla se encuentra seleccionada, WinBUGS usa métodos de simulación multivariados descritos en Gamerman (1997), para generar nuevos valores para los bloques de los ajustes a los parámetros.

Update options...

Una vez que el modelo ha sido compilado, existen varios algoritmos para ejecutar la simulación MCMC del modelo; puede elegir alguno a través del comando *Updater option*[5].

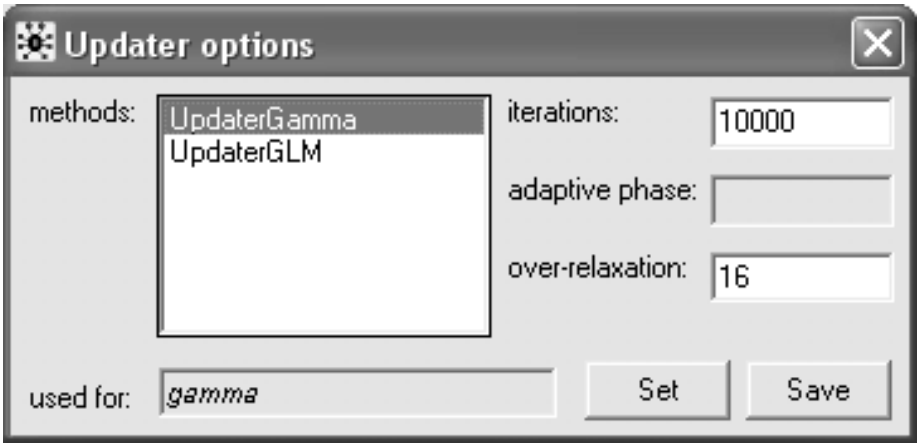


Figura 4.22: *Este cuadro muestra los nombres los métodos para actualizar el modelo en cuestión.*

[5]Es posible cambiar el método de muestreo para cierta clase de distribuciones, sin embargo esto debe hacerse con mucho cuidado.

methods: Todos los componentes del cuadro de diálogo *Updater options*, (campos y comandos) pertenecen al método que esté seleccionado.

used for: Describe para qué tipo de nodo el método seleccionado se utiliza comúnmente.

4.2.8. Gráficas en WinBUGS

Propiedades Generales

Todas las gráficas que produce WinBUGS tienen propiedades básicas que pueden ser modificadas a través del cuadro de diálogo *Plot Properties*. Para abrirlo, dé *click* sobre la gráfica y vaya al menú *Edit→Object Properties...* Otra alternativa es dar *click* con el botón derecho del *mouse* sobre la gráfica, para desplegar un menú, en el cual podrá observar *Properties...* El cuadro de diálogo correspondiente a *Plot Properties* comprende varias secciones y dos botones, *Apply* y *Special...* Cada sección contiene diferentes campos a través de los cuales puede realizar distintas modificaciones a las gráficas que podrá aplicar por medio del botón *Apply*. *Special...* despliega un cuadro en el que aparecen propiedades especiales que le permitirán interactuar con la gráfica.

Margins

Esta pestaña muestra los márgenes (expresados en milímetros) de la gráfica. Los márgenes izquierdo (*left*) e inferior (*bottom*) son usados para dibujar los ejes y y x, respectivamente. El margen superior proporciona un espacio para el título de la gráfica y el margen de la derecha es usado para las gráficas que requieran una leyenda (Figura 4.23).

Si los valores especificados no son adecuados, por ejemplo, el margen derecho más el izquierdo es mayor que el ancho de la gráfica o algún margen es negativo, no hay ningún cambio y los campos de *Margins* retoman los valores que tenían con anterioridad.

Figura 4.23: *Dé click en* Apply *para efectuar las modificaciones realizadas.*

Axis Bounds

El usuario puede especificar los valores mínimos y máximos de los ejes a través de *Axis Bounds* (Figura 4.24).

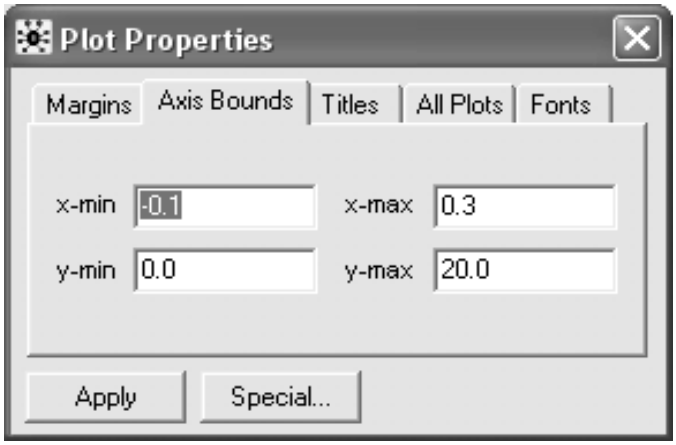


Figura 4.24: *Los valores del mínimo y el máximo no siempre serán los valores especificados porque WinBUGS siempre trata de dar una buena escala a los ejes.*

Note que si especifica el máximo con un valor menor que el mínimo, WinBUGS ignora esta especificación y reestablece los valores que tenía anteriormente. Por otra parte, si el rango de valores del modelo no se encuentra contenido en el rango de valores especificado, no aparece ningún valor graficado. Esta sección no está activada para gráficas como *trace*, que generalmente representan series de tiempo.

Titles

WinBUGS no tiene predeterminado un título para el eje vertical, pero puede introducir un título en el campo correspondiente (Figura 4.25). Si el espacio no es suficiente no aparece el título en la gráfica, por lo que tendrá que configurar los márgenes.

Figura 4.25: *WinBUGS proporciona el título de la gráfica por defecto y, en algunos casos, el título del eje horizontal.*

All Plots

Esta pestaña le permite aplicar algunas o todas las propiedades de la gráfica seleccionada a todas las gráficas que se encuentren en la misma ventana, observe Figura 4.26.

Marque las casillas que desee modificar y dé *click* en *Apply all*.

Figura 4.26: *Si comete un error no existe una opción para deshacer los cambios, por lo que tendrá que volver a generar la(s) gráfica(s).*

Fonts

En esta pestaña puede cambiar la fuente del título, de los ejes y de cualquier otro texto que se encuentre presente en la gráfica. Seleccione una casilla, dé *click* en el botón *Modify...*, para desplegar un cuadro de diálogo que le permitirá modificar el tamaño, color, tipo de fuente, entre otras características.

Figura 4.27: *Sólo para algunas gráficas aparece activado other; por ejemplo, para la gráfica de cajas y brazos.*

Propiedades específicas (a través de *Special...*)

Special... esta definido sólo para ciertas gráficas, a continuación se describen los distintos cuadros de diálogo que se despliegan al presionar este botón.

- **Density plot properties**. Al generar la gráfica de la densidad de una variable, si la variable es discreta la gráfica es un histograma, si es continua se dibuja una curva. Para cada gráfica se despliega un cuadro de diálogo diferente. En ambos cuadros aparece un campo numérico que el usuario puede manipular y dos botones: *apply* y *apply all*. En el caso del histograma el campo numérico corresponde al ancho de la barra *bin-size*, mientras que para la curva, *smooth*, corresponde al grado de suavidad de la curva. Si realiza algún cambio respecto a estos parámetros presione *apply*. WinBUGS define el parámetro de suavidad

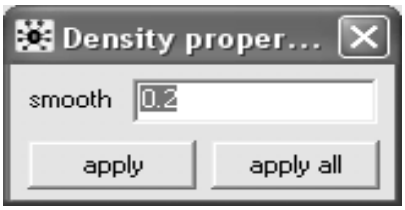


Figura 4.28: *Apply all, aplica las modificaciones que se realizaron a la gráfica seleccionada a todas las gráficas del mismo tipo que se encuentren en la misma ventana.*

para la curva, σ, en el caso de la gráfica de las variables continuas, a través de la definición *band-width*. Supongamos que la muestra *a posteriori* comprende m realizaciones de la variable z. Entonces,

$$band - width = \frac{\sqrt{v}}{m^{\sigma}}; \quad \text{con} \quad v = \frac{\sum_{i=1}^{m} z_i^2}{m} - \left(\frac{\sum_i^m z_i}{m}\right)^2.$$

Por defecto σ vale 0.2.

- **Box plot properties**. Es una herramienta para modificar las propiedades de las gráficas de cajas y brazos.

Figura 4.29: *El botón que se encuentra en la parte inferior derecha permite cambiar el color a las cajas.*

show baseline: Esta casilla aparece activada por defecto, dibuja una línea que cruza la gráfica y representa la media global de la medias finales. Podrá especificar una línea con un valor distinto a la media global en el campo que se encuentra en el lado derecho, observe la Figura 4.29. Si no desea que aparezca la línea desmarque la casilla.

show label: Al marcar esta opción se muestran los índices de cada componente del nodo.

show means o show medians: Al seleccionar alguna de estas opciones se muestra en la gráfica la media final o la mediana final, según la elección que haya hecho, por defecto aparece la media final.

rank: Al marcar esta casilla se grafican los rangos de la distribución. La base para obtener los rangos es la media final o la mediana final, dependiendo de que opción haya marcado *show means* o *show medians*.

vert. box o horiz. box: La elección de alguna de estas opciones le permite dar una orientación vertical u horizontal a la gráfica.

log-scale: Puede cambiar la escala de los ejes a una escala logarítmica al seleccionar esta casilla.

- **Caterpillar plot properties**. Las propiedades que existen para esta gráfica son muy similares a las de la gráfica de cajas y brazos, salvo

que en este cuadro de diálogo no aparece el comando para cambiar de color algún atributo de la gráfica.

- **Model fit properties**. En el caso de que un eje esté definido con un rango estrictamente positivo es posible cambiarlo a una escala logarítmica al marcar las casillas de verificación.

Figura 4.30: *Es posible marcar las dos casillas si el rango para ambos ejes es positivo.*

- **Scartterplot properties**. También existen propiedades especiales para las gráficas de dispersión, en las que se podrán modificar las propiedades que a continuación se explican.

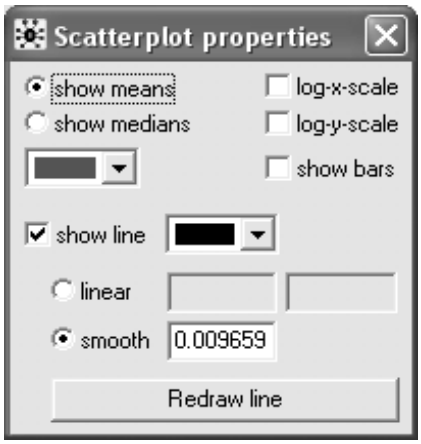


Figura 4.31: *Si la casilla show line no se encuentra seleccionada, las opciones linear y smooth aparecerán desactivadas.*

show means o show medians: Seleccione *show means* si desea que la gráfica de dispersión se realice con base en las medias finales o elija *show medians* para las medianas finales; WinBUGS utiliza por defecto las medias. Abajo de estas casillas aparece un botón

que le permite seleccionar el color de los puntos que se dibujan en la gráfica, observe la Figura 4.31.

log-x/log-y-scale: Marque estas casillas para representar a los ejes en una escala logarítmica (puede seleccionar ambas casillas) siempre y cuando el rango de valores sea estrictamente positivo.

show bars: Muestra un intervalo *a posteriori* al 95 % de probabilidad, construido a partir de los cuantiles 2.5 % y 97.5 %. El intervalo de cada elemento es representado por una línea vertical.

show line: Al activar esta casilla se traza una línea. Al lado aparece un botón que le permite cambiar el color de la línea, observe la Figura 4.31.

linear o smooth: Estas casillas le permiten especificar el tipo de línea que se ajustará a los datos. Si elige una línea recta debe proporcionar la ordenada al origen y la pendiente. Si opta por *smooth* debe proporcionar el grado de "suavidad" de la curva. WinBUGS predetermina valores tanto para la suavidad de la curva como para la ordenada al origen y la pendiente de la recta; sin embargo, si desea realizar cambios, después de introducir los valores en los campos pulse el botón *Redraw line*.

WinBUGS utiliza pesos exponenciales para construir la curva, los cuales se definen de la siguiente forma:

$$s_i = \frac{\sum_{j=1}^{n} w_{ij} y_j}{\sum_{j=1}^{n} w_{ij}}; \qquad i = 1, \ldots, n,$$

donde n es el número de puntos. Los pesos w_{ij} están dados por la siguiente expresión:

$$w_{ij} = \exp(-|x_i - x_j|/\sigma),$$

donde σ es el grado de suavidad definido en el cuadro de diálogo *Scatterplot properties→smooth*. El valor de σ es de alguna manera arbitrario:

$$\sigma = \frac{\text{máx}(x) - \text{mín}(x)}{20}.$$

4.2.9. Modelo Gráfico

El modelo gráfico de WinBUGS consta de tres elementos: nodos, placas y uniones. Para construir los gráficos es necesario utilizar el teclado, el *mouse* y el menú *Doodle*. A continuación se describen algunos comandos que aparecen en este menú.

New: Permite generar y especificar el tamaño de la ventana, también podrá proporcionar el tamaño del nodo.

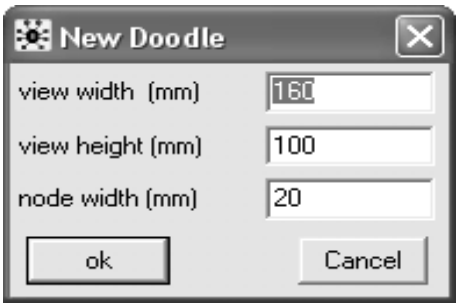


Figura 4.32: *A través de este cuadro de diálogo se especifica el tamaño de la ventana en la que se realizará el gráfico.*

view width: Permite especificar el largo de la ventana.

view heigth: Permite especificar la altura de la ventana.

node width: Permite especificar el ancho del nodo.

Las medidas del largo y la altura de la ventana; así como el ancho del nodo están expresadas en milímetros (mm).

Scale node: Por medio de este comando puede reducir el tamaño del gráfico en un porcentaje, pero no incrementarlo.

Remove Selection: A través de este comando desaparecerán los datos correspondientes a los nodos o a la placa, de esta forma sólo podrá ver la estructura del gráfico.

Write Code: Abre una ventana en la que aparece el código escrito en lenguaje BUGS correspondiente al gráfico.

Crear un nodo

Abra una ventana a través del menú *Doodle→New*, coloque el puntero del *mouse* en la ventana y dé *click* para crear un nodo. En la parte superior de la ventana aparecerá un texto azul referente a las características del nodo que deberá especificar, las cuales se mencionan a continuación:

name: Nombre del nodo. El nombre debe empezar con una letra y puede contener números e incluso un punto, no podrá contener dos puntos sucesivos y no deberá terminar con un punto. Un vector se denota por corchetes y los índices se separan por comas. Para indicar el rango de un nodo o una placa, éste debe ir entre corchetes separado por dos puntos.

type: Tipo de nodo.Un nodo puede ser estocástico (*stochastic*), determinístico (*logical*) o una constante (*constant*).

Stochastic: Si el nodo es estocástico debe indicar en *density* la densidad del nodo creado.

Logical: Un nodo de este tipo está en función de otros. La expresión que representa este nodo se especifica en *value*, y en *link* se indica el tipo de expresión (pueden ser *logit*, *probit*, *cloglog*, *log* o la identidad, sin transformación).

Constant: Si el nodo es una constante, se representa por un rectángulo, a diferencia de los demás nodos que se dibujan como una elipse.

Nota: Al crear un nodo, WinBUGS asume de entrada que es estocástico y que tiene una distribución normal. Para cambiar las características del nodo debe dar *click* sobre el texto azul. En el caso de tipo y densidad del nodo aparece una lista con las opciones que puede seleccionar.

Selecionar un nodo

Para seleccionar un nodo ubique el puntero del *mouse* sobre el nodo y dé *click*.

Eliminar un nodo

Seleccione el nodo y presione *CTRL* más *Suprimir*.

Mover un nodo

Seleccione el nodo, dé *click* sin soltar el *mouse* y coloque el nodo en el área deseada. También puede mover el nodo una vez seleccionado, a través de las teclas de navegación.

Crear una placa

Coloque el puntero del *mouse* en la región en blanco de la ventana que creó a partir del menú *Doodle→New* y presione *CTRL* más *click*. La placa representa el número de iteraciones que se llevarán a cabo respecto a uno o más nodos, por medio de un ciclo *for*, como en un lenguaje de programación. Al igual que el nodo, la placa tiene ciertas características que deberá especificar.

index: La variable que lleva el registro de las iteraciones.

from: Iteración en la que empezará a ejecutarse el ciclo *for*.

upto: Última iteración del ciclo *for*.

Seleccionar una placa

Coloque el puntero del *mouse* en los bordes que aparecen a la derecha de la placa y dé *click*.

Eliminar una placa

Seleccione la placa y presione *CTRL* más *Suprimir*.

Mover una placa

Seleccione la placa y dé *click* en el borde sombreado, sin soltar el *mouse* arrastre la placa. Después de haber seleccionado la placa también podrá moverla por medio de las teclas de navegación.

Cambiar el tamaño de la placa

Seleccione la placa y dé *click* en la esquina inferior derecha, sin soltar el *mouse* modifique el tamaño.

Crear un enlace o unión

Seleccione el nodo al que apuntará el enlace y dé *click* en el nodo padre[6], presione *CTRL* al mismo tiempo.

Borrar un enlace o unión

Siga el mismo procedimiento que hizo para crear un enlace.

Mover el gráfico

Después de haber construido el gráfico, si se requiere, puede moverlo al documento que contenga los datos, los valores iniciales, gráficas o texto. Vaya al menú *Edit→Select Document*. Finalmente, copie y pegue.

Cambiar el tamaño del gráfico

Sitúe el puntero del *mouse* en el área del gráfico y dé *click*. Ubique el puntero del *mouse* sobre los pequeños cuadros del borde que aparecen alrededor del gráfico. El puntero se convertirá en una flecha de dos puntas, dé *click* sin soltar el *mouse* para cambiar el tamaño del gráfico.

Imprimir un gráfico

Puede imprimir el gráfico directamente desde WinBUGS, o copie el gráfico en otro paquete que le permita imprimir imágenes o gráficas.

Ejemplo

Para ilustrar de una mejor manera la construcción de un modelo gráfico se usará un modelo de regresión lineal simple,

$$y_i = \alpha + \beta x_i + \epsilon_i,$$

donde los errores tienen distribución normal con media cero y varianza σ^2. Suponga que α y β tienen distribuciones iniciales *no informativas*, para lo cual se utilizan distribuciones normales con una precisión cercana a cero. Por otra parte, es conveniente trabajar en términos de la precisión en vez de la

[6]Se dice que un nodo v es padre de un nodo w si un arreglo proviene de v y apunta a w y, por lo tanto w es un nodo hijo de v.

varianza. Para este parámetro también se supone que tiene una distribución inicial *no informativa*, dada por una distribución gamma con parámetros cercanos a cero.

Vaya al menú *Dooble*, elija *New* para crear la ventana en la que se construirá el modelo. Dé *click* en la ventana creada, para generar un nodo. Como se puede observar en la Figura 4.33, en la parte superior deberá especificar las características de los nodos. Puesto que β se encuentra seleccionado, sólo se muestran las características para este nodo. En este caso α tiene las mismas características que β, ambos nodos tienen distribución normal y los valores de los parámetros son los que WinBUGS proporciona por defecto.

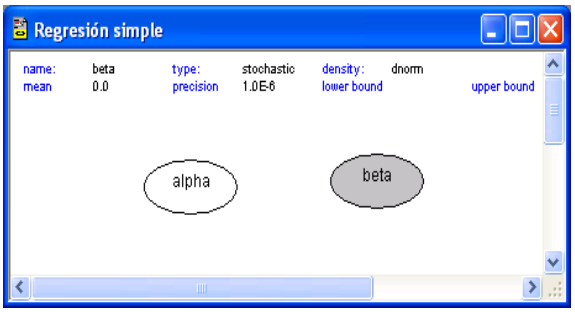


Figura 4.33: *Para especificar las características de los nodos dé click sobre el texto que aparece en la parte superior de la ventana.*

En la Figura 4.34 se agrega el nodo $x[i]$ que representa a los datos observados y que serán evaluados en la recta mediante un ciclo *for*, representado por la placa. Como los valores de cada $x[i]$ son constantes, el nodo se representa por medio de un rectángulo.

El nodo μ_i representa la media para cada y_i. Éste a su vez depende de α, β y x_i; es decir $\mu_i = \alpha + \beta x_i$. Por lo tanto, el nodo μ_i se dibuja dentro de la placa y además debe ser de tipo *logical* (determinístico), por estar en función de los demás nodos. Como se puede observar en la Figura 4.34, en el campo *value* se especifica la ecuación de la recta. En el campo *link* se utiliza la expresión predeterminada por WinBUGS, ya que no se aplica ninguna transformación a la recta.

El siguiente paso es especificar la distribución para cada y_i cuyos parámetros son μ_i y τ. Como ya se había mencionado anteriormente, τ se distribuye gamma y se le asignan parámetros cercanos a cero. En la Figura 4.35 se muestra el modelo gráfico completo para este ejemplo.

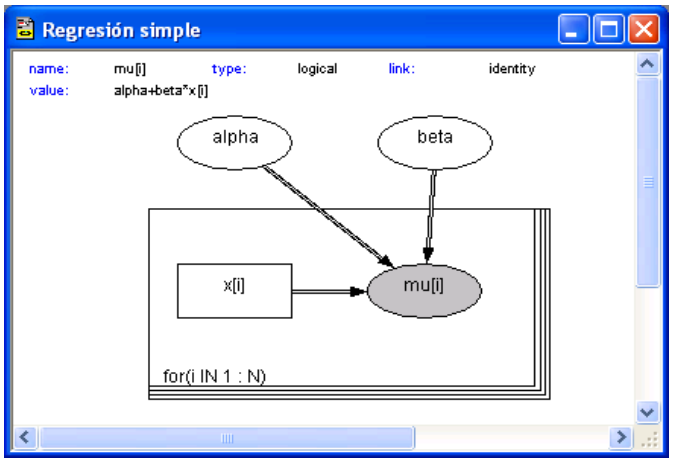


Figura 4.34: *Puesto que μ_i depende de α, β y $x[i]$ los enlaces apuntan hacia μ_i.*

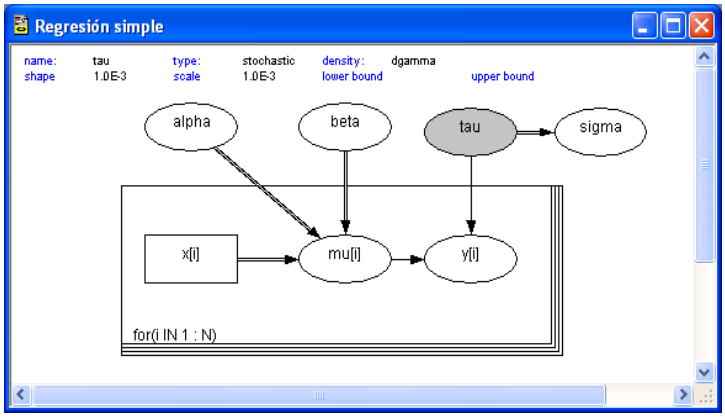


Figura 4.35: *Como y_i es un nodo estocástico los enlaces que apuntan hacia éste son sencillos, a diferencia de los que apuntan hacia μ_i.Para calcular la varianza se crea otro nodo σ.*

Si desea obtener el modelo en lenguaje BUGS a partir de esta gráfica, vaya al menú *Dooble* y dé *click* en *Write Code*, para desplegar una ventana como se muestra la que en la Figura 4.36.

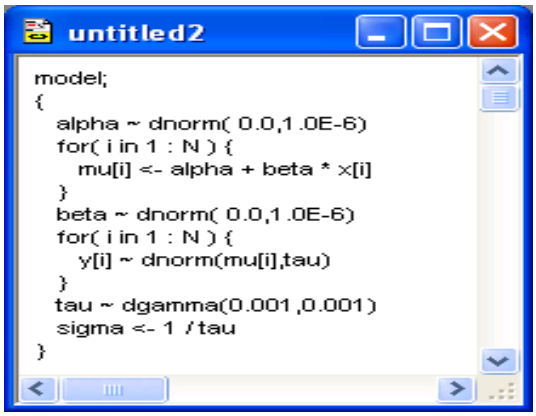


Figura 4.36: *Modelo en lenguaje BUGS*

4.2.10. Especificar un modelo en lenguaje BUGS

Esta estructura permite definir el modelo, puede usar `model;` o `model`.

```
model{
  Lista de declaraciones del código del modelo
}
```

En el lenguaje BUGS el símbolo de una tilde "$\sim$" denota una relación estocástica, y una flecha "$\leftarrow$", una relación determinística. Observe que en el ejemplo de regresión (Figura 4.36) y, α y β tienen una distribución normal y τ una distribución gamma, por lo tanto la relación es estocástica. Por otra parte, μ y σ representan una función de los nodos anteriores y por lo tanto la relación es determinística.

Repetir estructuras

Para repetir estructuras se debe especificar un ciclo *for*; la sintaxis es la siguiente:

```
for (i in a:b) {
  Lista de declaraciones que se repiten para cada iteración "i"
}
```

Note que a y b no son estocásticos. Para el ejemplo de regresión el ciclo itera de 1 a N, donde N es la longitud de los vectores x y y (Figura 4.36).

Arreglos e índices

Los índices de los arreglos van entre corchetes cuadrados. Las cuatro operaciones básicas (+,-,* y /) pueden ser utilizadas para realizar una función de un índice, por ejemplo:

$$Y[(i+j)*k, 1]$$

La expresión del índice izquierdo únicamente se evalúa si las variables son constantes o una función de los datos. El otro índice puede ser un valor o el nombre de un nodo. Las funciones de los nodos no observados no pueden aparecer directamente como un término del índice salvo los nodos determinísticos. WinBUGS utiliza la misma notación que R o S-Plus para el uso de índices y arreglos.

- $n : m$ Representa $n, n+1, \ldots, m$.
- $x\,[\,]$ Todos los elementos del arreglo x.
- $y\,[, 3]$ Indica todos los elementos de la tercera columna de una matrix.

Transformaciones de los datos

Aún cuando la transformación de los datos puede realizarse antes de usar WinBUGS, en ocasiones es conveniente probar varias transformaciones de las variables en la descripción de un modelo. Por ejemplo, se puede probar $sqrt(y)$ sin necesidad de crear una variable por separado ($z = sqrt(y)$) en el archivo de los datos. WinBUGS permite escribir la siguiente estructura.

```
for (i in 1:N) {
    z[i] <- sqrt(y[i])
    z[i] ~ dnorm(mu, tau)
}
```

Estrictamente hablando, esta estructura no va de acuerdo con la especificación del modelo, pues a un nodo se debe hacer sólo una asignación. Sin embargo, al revisar el modelo, cuando se encuentra un nodo determinístico asociado también a un nodo estocástico, como se muestra en la estructura

anterior, el nodo estocástico se crea con los valores que se calcularon a través del nodo determinístico.

Esta construcción es muy útil en modelos que utiliza las transformaciones de Box-Cox y en otras circunstancias donde las transformaciones son muy complejas. En estos casos es recomendable hacer la transformación de los datos en una sección al inicio de la especificación del modelo; así la descripción esencial del modelo puede examinarse por separado.

Anidar índices: mezclas

Suponga que cada uno de N individuos puede estar en uno de G grupos, y $g[1:N]$ es un vector que contiene los miembros del grupo. Entonces, un grupo de coeficientes $beta[i]$ puede ajustarse usando $beta[g[1:N]]$ en una ecuación de regresión.

En el lenguaje BUGS, anidar índices puede ser útil para especificar los parámetros de las distribuciones, como se puede observar en el ejemplo *Eyes*[7], el cual contiene una mezcla de distribuciones normales en la que el i-ésimo caso en un grupo desconocido T_i determina la media λ_{T_i} de y_i y el modelo se especifica de la siguiente forma:

$$T_i \sim Categorical(P)$$

$$y_i \sim Normal(\lambda_{T_i}, t),$$

que puede escribirse en el lenguaje BUGS como:

```
for (i in 1:N) {
    T[i] ~ dcat(P[ ])
    y[i] ~ dnorm(lambda[T[i]], tau)
}
```

Sin embargo, cuando se usan gráficos los parámetros de una distribución deben aparecer como nodos en la gráfica, de manera que se requiere una placa adicional para especificar $\mu_i = \lambda_{T_i}$, como se muestra en la Figura 4.37. El vector de parámetros puede ser identificado de forma dinámica, pero sólo con un máximo de dos dimensiones. Suponga que hay dos placas para una variable categórica x. Para tener un vector de probabilidades con índices de i y j, se puede escribir $x \sim dcat(p[i, j, 1:2])$.

[7] *Eyes:normal mixture model* se encuentra en el *Volumen II* de ejemplos de WinBUGS.

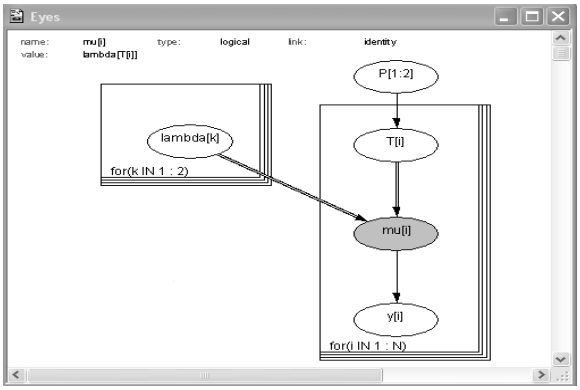


Figura 4.37: *Nodo* y_i

Formato de los datos

Los datos se pueden especificar en un formato como el de S-Plus o R, o como un arreglo de datos en un arreglo rectangular. De preferencia cree un archivo para el arreglo; en este caso no es posible especificar sólo ciertas componentes. Los datos faltantes son representados como `NA`. Todas las variables en un archivo de datos deben especificarse en el modelo.

Formato de S-Plus o R: Los arreglos y escalares se encuentran en una sola estructura, especificados a través de la instrucción `list`. Es importante no dejar un espacio después del comando `list`, de lo contrario se desplegará un mensaje de error. En el siguiente ejemplo, *Rats*, que se encuentra en el *Volumen I* de ejemplos de WinBUGS, se ilustra esta manera de representar los datos. Se especifica un escalar, *xbar*; el número de individuos, N; un vector, x; la dimensión de x, T; y un arreglo de dos dimensiones, Y; de 30 renglones y 5 columnas.

```
list( xbar = 22, N = 30, T = 5,
      x = c(8.0, 15.0, 22.0, 29.0, 36.0),
      Y = structure(
          .Data = c(
            151, 199, 246, 283, 320,
            ...................
            153, 200, 244, 286, 324),
          .Dim = c(30, 5)
        )
     )
```

WinBUGS lee datos de un arreglo en un orden distinto al de S-Plus o R. Específicamente, al leer la cadena de números $c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$ en una matriz de 2×5, WinBUGS utiliza el orden:

```
[i, j]-ésimo elemento
de la matriz          valor
[1, 1]                1
[1, 2]                2
[1, 3]                3
.....                 ..
[1, 5]                5
[2, 1]                6
.....                 ..
[2, 5]                10
```

mientras que S-Plus o R leen la misma cadena de números en el siguiente orden:

```
[i, j]-ésimo elemento
de la matriz            valor
[1, 1]                  1
[2, 1]                  2
[1, 2]                  3
.....                   ..
[1, 3]                  5
[2, 3]                  6
.....                   ..
[2, 5]                  10
```

Por lo tanto, el orden de las dimensiones del arreglo debe invertirse antes de usar el comando *dput* de S-Plus o R para crear un archivo de datos para WinBUGS. Por ejemplo, considere la matriz M de 2×5.

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \end{bmatrix}$$

En S-Plus o R debe especificarse como una matriz de dimensiones 5×2:

```
> M
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
```

El siguiente comando de S-Plus y R

```
> dput(list(M=M), file="matrix.dat")
```

produce el siguiente archivo:

```
list(M = structure(.Data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
.Dim=c(5,2))
```

Se debe hacer un cambio de $.Dim = c(5, 2)$ a $.Dim = c(2, 5)$, para poder ser usado en WinBUGS.

Forma Rectangular: Las columnas de los datos en un formato rectangular necesitan llevar un nombre y deben ser todas del mismo tamaño. Además, al final la matriz deberá escribir `END` como se muestra a continuación:

```
age[ ] sex[ ]
26        0
52        1
..        ..
34        0
  END
```

Puede especificar un arreglo multidimensional al hacer explícitos los índices, por ejemplo,

```
Y[,1] Y[,2] Y[,3] Y[,4] Y[,5]
151   199   246   283   320
145   199   249   293   354
147   214   263   312   328
.......
153   200   244   286  324
      END
```

Es posible cargar los datos a través de una combinación de un arreglo rectangular y un formato de S-Plus o R para el mismo modelo. Es decir, si el arreglo de datos proviene de un archivo que contiene un arreglo rectangular, las constantes pueden ser definidas en una lista por medio del comando `list`. Para el ejemplo *Rats* se crearon dos archivos *Ratsx* y *Ratsy*. El primer archivo (Figura 4.38) contiene un escalar *xbar*, N el número de individuos y T la dimensión de x, y el vector x. El segundo archivo (Figura 4.39) contiene un arreglo Y, de 30 renglones y 5 columnas.

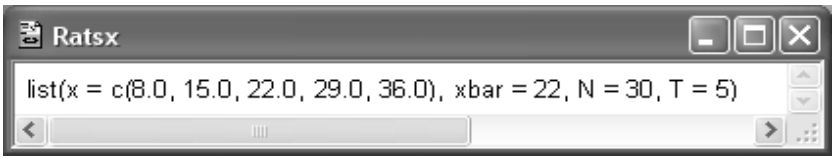


Figura 4.38: *Constantes del modelo*

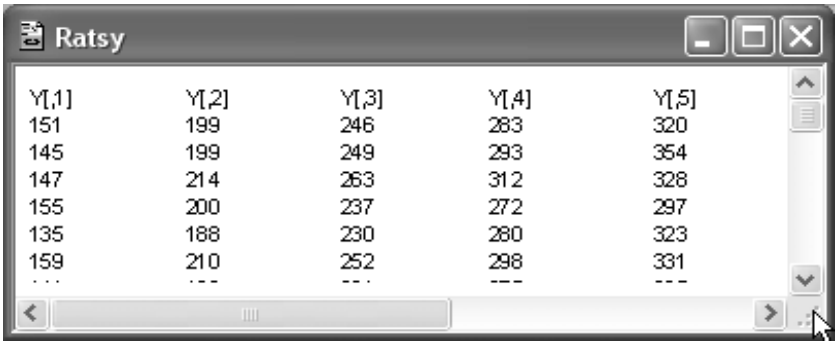


Figura 4.39: *Observaciones*

4.2.11. ¿Cómo correr un modelo en WinBUGS?

Para explicar el procedimiento que debe seguirse para correr un modelo se ilustrará por medio de uno de los ejemplos que se encuentran en el *Volumen I* de la sección de ejemplos de WinBUGS, *Seeds: Random effect logistic regression*. En la Figura 4.40 y 4.41 podrá observar de manera esquemática como llevar a cabo este procedimiento.

Paso 1

Abrir un archivo.

- Vaya a la barra de herramientas al menú *File*.
- Dé *click* en *Open*.
- Localice el directorio `C:\Archivos de Programa\WinBUGS14\Examples` y abra el archivo *Seeds.odc*.

Paso 2

Para correr el modelo primero hay que verificar que el modelo esté completamente especificado:

- En la barra de menú localice *Model*.
- Dé *click* en *Specification*.
- Seleccione el comando `model` que se encuentra al principio del código, para hacerlo, arrastre el *mouse* sobre el comando o simplemente dé doble *click* sobre éste. Si generó gráficamente el modelo dé doble *click* sobre el gráfico. En este caso, el código fue generado de esta última forma; sin embargo, puede proceder de cualquiera de las dos formas.
- Para verificar la sintáxis del modelo, dé *click* en el botón *check model* del cuadro de diálogo *Specification Tool*. Si el modelo es correcto aparece el mensaje, "`model is syntactically correct`", en la barra de estado.

Paso 3

El siguiente paso es cargar los datos, los cuales pueden estar representados como un objeto de S-Plus o R, o una combinación de un objeto de S-Plus o R y de un arreglo rectangular.

- Para cargar los datos con un formato S-Plus o R:
  - Seleccione el comando `list`.

- Dé *click* en el botón *load data* del cuadro de diálogo *Specification Tool*. Si los datos se han cargado se despliega el mensaje "`data loaded`" en la barra de estado.

- Para cargar los datos con un formato S-Plus o R y un arreglo:
  - Abra el o los archivo(s) que contiene(n) los datos.
  - Seleccione el comando `list`.
  - Dé *click* en el botón *load data* del cuadro de diálogo *Specification Tool*, una vez que se han cargado los datos aparece el mensaje "`data loaded`" en la barra de estado.
  - Después seleccione los títulos completos de las columnas del arreglo rectangular de datos. Si únicamente se encuentra el arreglo en la ventana, basta con activarla.
  - Dé nuevamente *click* en el botón *load data*, si WinBUGS ha cargado los datos aparece el mensaje "`data loaded`" en la parte inferior de la ventana principal de WinBUGS. Para llevar a cabo esta forma de especificar los datos, abra el archivo *seeds_mix_data.odc* que se encuentra en el directorio `C:\Archivos de programa\WinBUGS14\Manuals\Tutorial`.

Paso 4

Ahora necesita especificar el número de cadenas, es decir, el conjunto de muestras a simular. WinBUGS simula una por defecto, pero para este ejemplo se usarán dos, ya que al correr múltiples cadenas se puede verificar la convergencia de la simulación MCMC.

- Tecleé el número 2 en el campo *num of chains* de *Specification Tool*. En la práctica, si tiene un modelo muy complicado, puede hacer una corrida preliminar del modelo con una sola cadena para verificar el modelo y obtener una estimación del tiempo por iteración. Después puede correr el modelo para más cadenas obteniendo un conjunto de las estimaciones finales.

Paso 5

En este paso hay que compilar el modelo, presione el botón *compile* de *Specification Tool*, una vez compilado el modelo aparece en la barra de estado

el mensaje "`model compiled`". Al compilar el modelo, WinBUGS construye una estructura de los datos internamente y elige un algoritmo para actualizar las MCMC para un modelo en particular.

Paso 6

Finalmente debe proporcionar los valores iniciales para cada nodo estocástico, a partir de los cuales se generará la muestra MCMC. Estos valores pueden ser arbitrarios; sin embargo en la práctica, la convergencia es pobre si los valores elegidos no son apropiados. Para este ejemplo, usted necesita un conjunto de valores iniciales para cada cadena; los cuales puede encontrar el archivo *seeds_inits.odc* en el directorio `C:\Archivos de programa\WinBUGS14\Manuals\Tutorial`.

- Para cargar los valores iniciales:
  - Seleccione el comando `list`.
  - Dé *click* en el botón *load inits* del cuadro de diálogo *Specification Tool*, si se han cargado los valores iniciales se despliega el mensaje "`model is initialized`". Si aparece el mensaje "`initial values loaded: model contains uninitialized nodes`", dé *click* en el botón *gen inits* o intente proporcionar los valores que faltan.
  - Repita el proceso para los valores iniciales de la segunda cadena.

Note que no necesita proveer una lista inicial de valores para cada parámetro en su modelo. WinBUGS puede generar valores iniciales para cualquier parámetro estocástico que no haya sido iniciado a través de *Specification Tool* →*gen inits*.

Paso 7

Ahora todo está listo para iniciar la simulación. Sin embargo, antes de proceder, usted probablemente desee monitorear algunos parámetros sin someterlos a un *periodo de calentamiento*. Si es así, vaya al menú *Inference*→*Sample Monitor Tool...* y registre el o los nodo(s). Para correr la simulación:

- Dé *click* en *Update...* del menú *Model*.

- Tecleé el número de iteraciones que requiera simular en el espacio blanco (etiquetado como *updates*), el valor predeterminado es 1000.

- Dé *click* en el botón *update*, el programa empezará a simular valores para cada parámetro del modelo. Esto podría tomar algunos segundos, *iteration* indica cuántos datos ya han sido simulados. El número de veces que estos valores son reportados, depende del valor que determinó en *refresh*, por defecto es cada 100 iteraciones. Usted puede modificar este campo dependiendo de qué tan rápido corre el programa.

- Cuando las actualizaciones hayan finalizado, aparece el mensaje "*updates took *** s*" en la parte inferior izquierda de la ventana de WinBUGS (donde *** es el número de segundos que ha tomado completar la simulación).

- Si cuenta con un conjunto de valores previamente monitoreados para cada parámetro, puede verificar la convergencia en la gráfica y el resumen de las muestras (ver sección 4.2.13). Para verificar la convergencia en el ejemplo *Seeds* deberá percatarse que cuenta con al menos 2000 iteraciones.

- Una vez que usted considere que su simulación ha convergido, necesita realizar algunas simulaciones para obtener una muestra de la distribución final. La sección 4.2.14 contiene *tips* para decidir cuántas simulaciones más debe correr. Para este ejemplo, intente correr al menos 10000.

- Ya que haya simulado un número de iteraciones suficientes para obtener una muestra apropiada de la distribución final, puede generar el resumen de las estadísticas numérica y gráficamente[8] para los parámetros monitoreados.

- Para guardar cualquier archivo generado en WinBUGS, vaya al menú *File→Save As...* y proporcione el nombre del archivo.

- Para salir de WinBUGS, vaya al menú *File→Close*.

[8] Ver sección 4.2.15 para obtener detalles de cómo generar los resumenes de los parámetros.

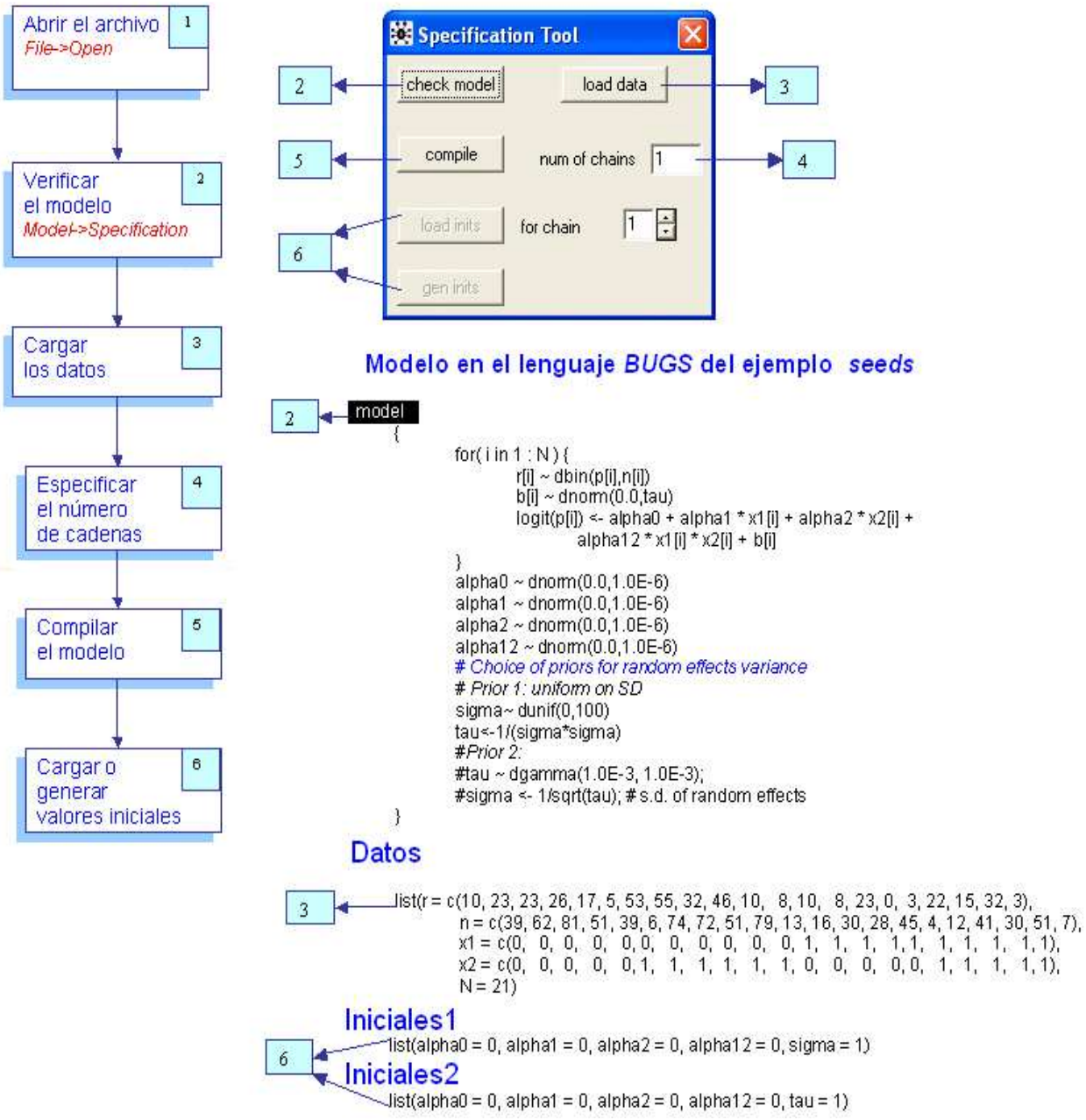


Figura 4.40: *Especificación de un modelo.*

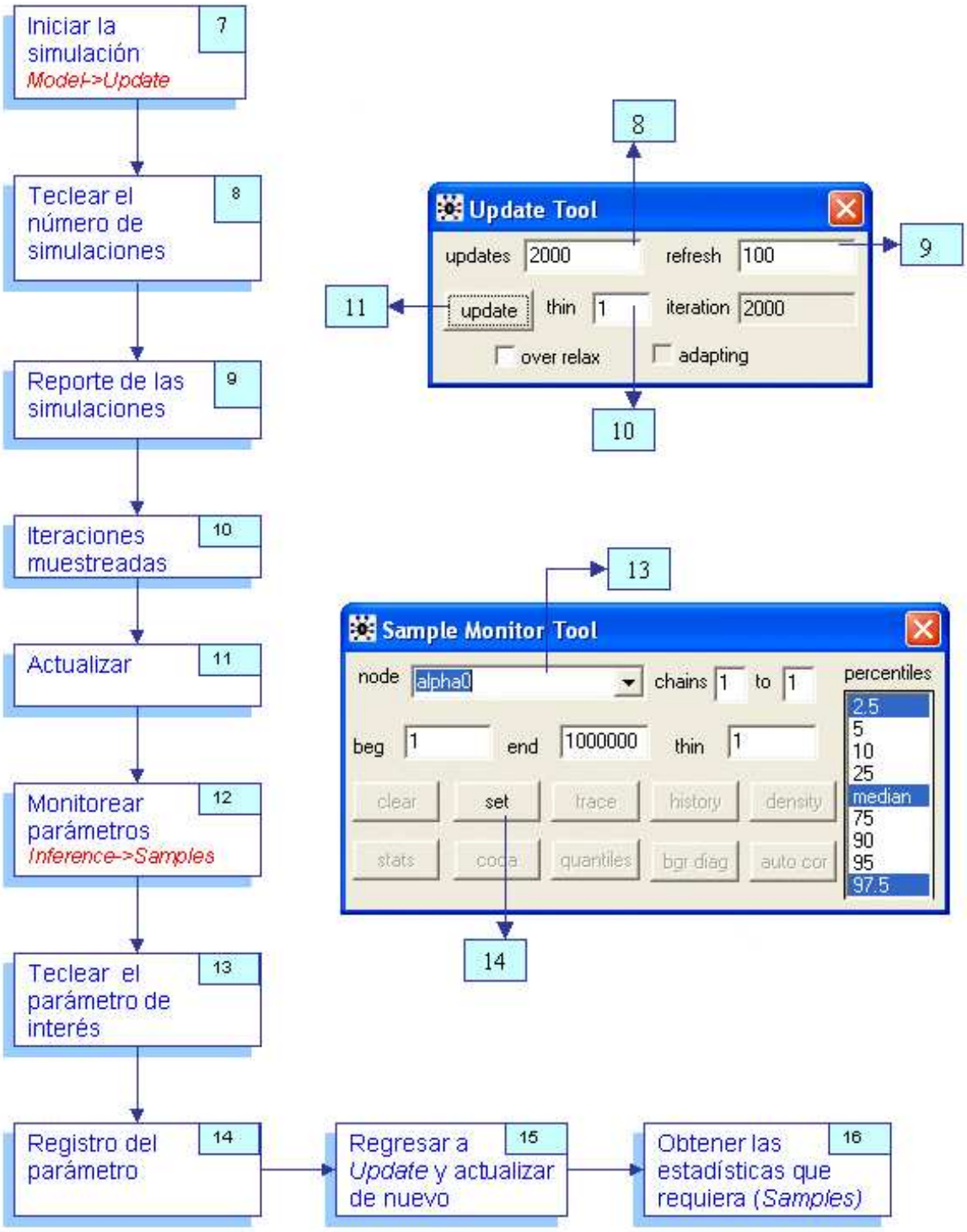


Figura 4.41: *Periodo de calentamiento, monitoreo de los parámetros y obtención de las estadísticas de los parámetros.*

Monitorear los valores de los parámetros

Con el objeto de verificar la convergencia y obtener el resumen de los parámetros del modelo final. Primero debe monitorear los parámetros de interés. Esto le dice a WinBUGS que debe almacenar los valores muestreados para los parámetros. De otra forma, WinBUGS descarta automáticamente los valores simulados.

Existen dos tipos de monitoreo en WinBUGS:

Samples monitors: Al monitorear muestras WinBUGS almacena cada valor que es simulado para el parámetro. Usted necesita monitorear un conjunto de muestras si desea observar la trayectoria de las muestras en la gráfica para verificar la convergencia (ver la sección 4.2.13) o si desea obtener los cuantiles finales exactos, por ejemplo, el intervalo Bayesiano de credibilidad final al 95 % para el parámetro. Note que también puede obtener los cuantiles 2.5 %, 50 % y 97.5 % aproximadamente de la distribución final para cada parámetro a través de *summary monitors*. Pasos para monitorear las muestras:

- Seleccione *Samples* del menú *Inference*.
- Tecleé el nombre del parámetro que será monitoreado en el campo *node*.
- Dé *click* con el botón izquierdo del *mouse* sobre *set*.
- Repita el mismo procedimiento para cada parámetro.

Summary monitors: Este tipo de monitoreo le dice a WinBUGS que almacene las medias y la desviación estándar para el parámetro, además de aproximaciones de los cuantiles (2.5 %, 50 % y 97.5 %). Los valores guardados contienen menos información que los que se guardan para cada muestra en la simulación. Esta consideración es importante cuando las simulaciones son muy largas y los valores que se deben almacenar son para muchas variables.

Para realizar un *summary monitors* se siguen casi los mismos pasos que para hacer un *sample monitors*, la diferencia es que debe seleccionar *Summary...* en lugar de *Samples...*

4.2.12. Funciones

En la siguiente tabla se muestran las funciones que puede encontrar en WinBUGS. Los argumentos de las funciones representados por a son expresiones, mientras que los representados por j son escalares y v es un arreglo.

Una función que no se incluyó en la tabla es la función $I(.,.)$, la cual es útil para restringir el dominio de una distribución; esta notación no aplica para las distribuciones multivariadas a excepción de la distribución normal multivariada. Los argumentos de la función pueden especificarse en blanco o como un vector de valores, también es posible utilizar la notación $I(lower, upper)$, por ejemplo,

```
x ~ ddist(theta)I(lower, upper)
```

lo que significa que una cantidad x cuya distribución es *ddist* con parámetro *theta*, ha sido observada entre el dominio *lower* y *upper*, también puede utilizar $I(lower,)$ o $I(, upper)$ para establecer sólo un límite. Esta construcción es útil sólo si x no ha sido observada, de otra manera la restricción es ignorada. Observe que además de x también *theta*, *lower* y *upper* son no observados, por otro lado *lower* y *upper* no deben estar en función de *theta*.

4.2.13. Verificar la Convergencia

Verificar la convergencia requiere de mucho cuidado, pues es difícil concluir cuándo una cadena (la simulación) ha convergido. En general, sólo es posible diagnosticar cuándo definitivamente no lo ha hecho. A continuación se presenta una serie de puntos para valorar la convergencia:

- Para los modelos con muchos parámetros, resulta poco práctico verificar la convergencia de cada parámetro, por lo que se recomienda elegir un número pequeño de parámetros relevantes para ser monitoreados. Por ejemplo, en vez de verificar la convergencia para cada elemento de un vector, sólo escoja un subconjunto.

- Examine la secuencia de valores de la muestra contra las iteraciones para tener una idea de cuándo la simulación aparentemente es estable.

  Para obtener la gráfica de la secuencia de valores de la muestra contra las iteraciones de un parámetro:

Cuadro 4.2: Funciones

Funciones disponibles en WinBUGS	
Funciones	Descripción
$abs(a)$	$\|a\|$
$cloglog(a)$	$ln(-ln(1-a))$
$cos(a)$	$coseno(a)$
$cut(a)$	corta enlaces en una gráfica
$equals(a_1, a_2)$	1 si $a_1 = a_2$; 0 en otro caso
$exp(a)$	$exponencial(a)$
$inprod(v_1, v_2)$	$\sum_i v_{1_i} v_{2_i}$
$interp.lin(a, v_1, v_2)$	$v_{2_p} + (v_{2_{p+1}} - v_{2_p}) * (a - v_{1_p})/(v_{1_{p+1}} - v_{1_p})$ donde los elementos de v_1 van en orden ascendente y $v_{1_p} < a < v_{1_{p+1}}$
$inverse(v)$	v^{-1} para una matriz v simétrica definida positiva
$log(a)$	logaritmo natural $ln(a)$
$logdet(v)$	$ln(det(v))$ para una matriz v simétrica definida positiva
$logfact(a)$	$ln(a!)$
$logam(a)$	$ln(\Gamma(a))$
$logit(a)$	$ln(\frac{a}{1-a})$
$max(a_1, a_2)$	a_1 si $a_1 > a_2$; a_2 en otro caso
$mean(v)$	$\frac{\sum_i v_i}{n}$; $n = dim(v)$
$min(a_1, a_2)$	a_1 si $a_1 < a_2$; a_2 en otro caso
$phi(a)$	normal estándar acumulada
$pow(a_1, a_2)$	$a_1^{a_2}$
$probit(a)$	inversa de phi
$sin(a)$	$seno(a)$
$sqrt(a)$	$\sqrt{a}$
$rank(v, j)$	número de componentes de v menores o iguales a v_j
$ranked(v, j)$	la j-ésima componente más pequeña de v
$round(a)$	entero más cercano a a
$sd(v)$	la desviación estándar de las componentes de v (con denominador $n-1$)
$step(a)$	1 si $a \geq 0$; 0 en otro caso
$sum(v)$	$\sum_i v_i$
$trunc(a)$	el mayor entero menor o igual que a

- Seleccione *Samples...* del menú *Inference*.
- Tecleé o seleccione el nombre del parámetro en el campo *node*.
- Dé *click* en el botón *trace* para desplegar una gráfica vacía en una ventana.
- Realice el mismo procedimiento para cada parámetro.
- Una vez que corra la simulación (a través del comando *Update...* que se encuentra en el menú *Model*) se dibujarán los valores del parámetro seleccionado sobre la gráfica que se encontraba vacía.

Para obtener la historia completa de los valores que se han generado para el parámetro, debe contar con un conjunto de una muestra monitoreada y generar más valores:

- Seleccione el comando *Samples...* del menú *Inference*.
- Tecleé el nombre del parámetro en el campo *node* (o seleccione el nombre del parámetro en la lista, si ya ha sido introducido).
- Dé *click* en el comando *history*, para desplegar una gráfica de los valores simulados del parámetro de interés.

En la Figura 4.42 se muestran ejemplos de convergencia.

Si desea simular más de una cadena, en la historia se muestra cada cadena de diferente color. En este caso, puede suponer que la convergencia ha sido alcanzada si todas las cadenas aparecen traslapadas unas a otras, observe la Figura 4.43.

Para realizar un diagnóstico más certero de la convergencia, el *software* también provee una implementación de técnicas descritas en Brooks y Gelman (1998), y facilita el monitoreo de las muestras en un formato compatible con *boa* o *coda* (ver sección 4.4.1).

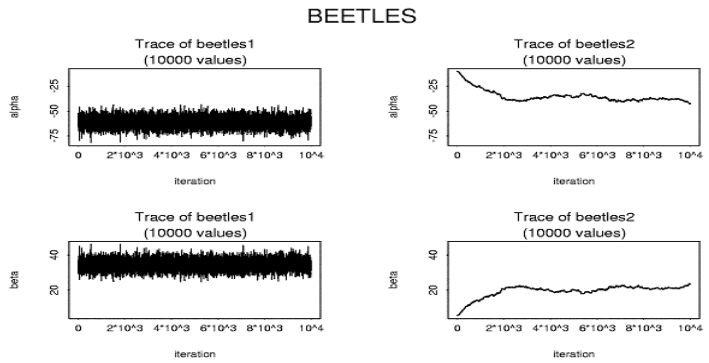


Figura 4.42: *Cadenas para las cuales la convergencia se muestra razonable (izquierdo). Cadenas en las que se observa claramente que no alcanzan la convergencia (derecho).*

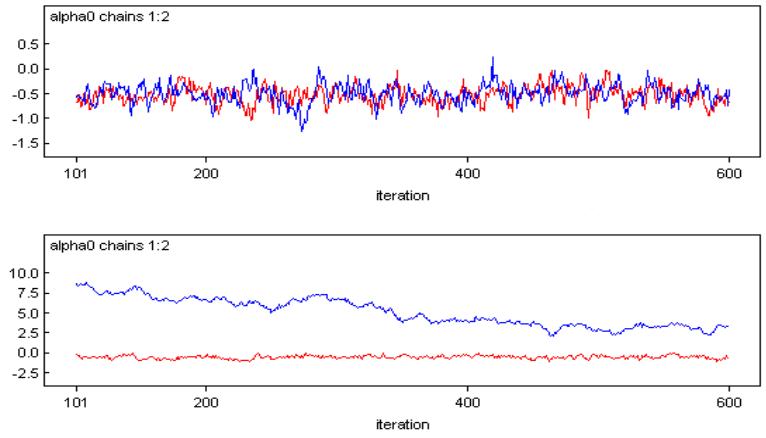


Figura 4.43: *Múltiples cadenas cuya convergencia es razonable (arriba). Múltiples cadenas las cuales no alcanzan la convergencia (abajo).*

4.2.14. ¿Cuántas iteraciones después de la convergencia?

Una vez que se ha alcanzado la convergencia, necesita realizar la simulación para un número mayor de iteraciones que le permita obtener una muestra que pueda ser usada para realizar inferencia *a posteriori*. La muestra que conserve le dará estimaciones *a posteriori* más precisas. Un camino para valorar la precisión de las estimaciones *a posteriori* es calcular el error Monte Carlo para cada parámetro. Esto es, una estimación de las diferencias en la media de los valores muestreados (los cuales son usados como la estimación de la media final para cada parámetro) y la verdadera media *a posteriori*. Como una regla empírica, la simulación debe correrse hasta que el error Monte Carlo para cada parámetro de interés sea menor que el 5 % de la desviación estándar de la muestra. El error Monte Carlo (MC error) y la desviación estándar (SD) son reportados en el cuadro del resumen de las estadísticas (ver la sección 4.2.5→*Sample Monitor Tool*). A continuación se indica cómo obtener este cuadro.

4.2.15. Obtener el resumen de la distribución final

Las muestras finales pueden estar resumidas gráficamente, por ejemplo usando la gráfica de la densidad, o numéricamente a través del resumen de las estadísticas como la media, la desviación estándar y los cuantiles de la muestra.

Para realizar inferencia *a posteriori* siga los siguientes pasos:

- Seleccione *Samples...* del menú *Inference*.
- Tecleé el nombre del parámetro en el campo *node*, selecciónelo si ya fue introducido o tecleé "*" (asterisco) para seleccionar todos los parámetros.
- Proporcione el número de iteraciones que desee resumir en el campo *beg*.
- Dé *click* en el botón *stats*, para desplegar una tabla con el reporte de las estadísticas basados en los valores de la muestra de los parámetros seleccionados.

- Para generar la gráfica del *kernel* de la densidad del parámetro en cuestión, presione el botón *density*.

4.3. Uso avanzado del lenguaje BUGS

Especificar una muestra de una nueva distribución

Suponga que desea usar una muestra de una distribución que no está incluida en la lista de distribuciones, Tabla 4.1, en la cual una observación $x[i]$ contribuye a la verosimilitud de $L[i]$. Puede usar el "truco de los ceros" (*zeros trick*) para generar tal distribución: una observación cero $Poisson(phi)$ tiene una verosimilitud $\exp(-phi)$, así que si las observaciones son un conjunto de ceros y $phi[i]$ está formado por $-\log(L[i])$, se obtiene una contribución correcta a la verosimilitud. Además $phi[i]$ debe ser mayor que cero como lo es la media de una Poisson, por lo que tal vez sea necesario agregar constantes adecuadas para asegurarse de que sea positiva.

```
C <- 10000 # C debe ser suficientemente grande
           # para asegurar que phi[i]'s > 0
for (i in 1:N) {
    zeros[i] <- 0
    phi[i] <- -log(L[i]) + C
    zeros[i] ~ dpois(phi[i])
   }
```

Este truco permite usar una muestra de una distribución arbitraria y es adecuado, particularmente cuando se requiere trabajar con distribuciones truncadas. En el apéndice B.2, se muestra un ejemplo que ilustra este truco, el cual se encuentra en el archivo *new-sampling* de WinBUGS.

Se puede predecir una nueva observación *x.pred* al especificar que falta en el archivo de datos y asignar una distribución uniforme inicial, por ejemplo:

```
x.pred ~ dflat() # distribución inicial impropia
                 # para un nuevo valor de x
```

Sin embargo, el método puede ser ineficiente y arrojar un error MC muy grande. En el apéndice B.3 puede consultar el código para generar una observación futura a partir de la correspondiente distribución predictiva.

Una alternativa al "truco de los ceros" es el "truco de los unos" (*ones trick*). La base de datos está formada por un conjunto de $1's$ que son resultado

de una serie de ensayos Bernoulli con probabilidad $p[i]$. Al hacer cada $p[i]$ proporcional a $L[i]$ (i.e. al especificar una constante lo suficientemente grande para asegurarse que todos los $p[i]'s$ son menores que 1), es posible aproximar el término correspondiente de la verosimilitud. Por ejemplo,

```
C <- 10000    # C tiene que ser suficientemente grande para
              # que todas las p[i]'s sean menores que 1
for (i in 1:N) {
    ones[i] <- 1
    p[i] <- L[i] / C
    ones[i] ~ dbern(p[i])
}
```

Especificar una nueva distribución inicial

Si para un parámetro *theta*, se quiere usar una distribución inicial que no sea parte de la lista de distribuciones, se puede utilizar el "truco de los ceros" (*zeros trick*). Como ya se mencionó, una observación cero de una Poisson (con media $phi = phi(theta)$) contribuye con un término de la $\exp(-phi)$ a la verosimilitud de *theta*; cuando se combina con una distribución inicial "plana" para *theta* resulta la distribución inicial correcta.

```
zero <- 0
theta ~ dflat()
phi <- -log(distribución inicial de theta deseada)
zero ~ dpois(phi)
```

En el apédice B.4 puede consultar el código para general una nueva distribución inicial a partir de una distribución normal. Es importante tomar en cuenta que estos métodos produce una autocorrelación alta, la convergencia puede ser pobre y el error MC puede resultar grande, lo cual constituye una desventaja computacional ya que las corridas deben ser muy largas y son lentas.

Especificar una distribución inicial discreta en un conjunto de valores

Suponga que un parámetro D está formado por un conjunto de valores, $d[1], \ldots, d[k]$, con probabilidad $p[1], \ldots, p[k]$. Se especifican los arreglos $d[1:k]$ y $p[1:k]$ en un archivo de datos (o en el modelo) y se usa:

```
M ~ dcat(p[ ]) # elige el elemento de d[ ]
D <- d[M]
```

Si la distribución inicial está definida sobre una retícula muy burda, entonces puede haber problemas numéricos. Por lo tanto se recomienda el uso de distribuciones iniciales continuas o distribuciones discretas sobre una retícula más fina.

Situaciones en las que el tamaño de un conjunto es aleatorio

Suponga que el tamaño de un conjunto de variables es aleatorio. Esto ocurre con frecuencia en problemas de "puntos de cambio", donde las observaciones hasta un cierto punto k provienen de un modelo, y las observaciones siguientes provienen de otro.

Note que no es posible usar la construcción:

```
for (i in 1:K) {
    y[i] ~ model 1
}
for (i in (K + 1):N) {
    y[i] ~ model 2
}
```

ya que el índice para un ciclo no puede ser un nodo estocástico. Una alternativa es usar la función "*step*" para construir un indicador del conjunto al que pertenece la observación. En el apéndice B.5 se muestra un ejemplo de "punto de cambio" que permite ilustrar esta situación.

Valorar la sensibilidad de los supuestos iniciales

Una opción para valorar la sensibilidad de los supuestos iniciales es repetir el análisis bajo diferentes suposiciones iniciales, pero dentro de la misma simulación para que la comparación de los resultados sea más directa. Para comparar los resultados de las K distribuciones iniciales se puede:

1. Replicar los datos K veces con el código del modelo.
2. Construir un ciclo para repetir el análisis para cada distribución inicial, manteniendo los resultados en arreglos.
3. Comparar los resultados a través del comando *compare*.

En el archivo *Prior Sensitivity* del manual, que puede encontrar en `C:\Archivos de programa\WinBUGS14\Manuals\Tricks`, se exploran seis diferentes sugerencias para las distribuciones iniciales en los efectos aleatorios de la varianza en un meta-análisis.

Modelar denominadores desconocidos

Suponga que desea describir el índice de una distribución Binomial como una distribución inicial. Si se utiliza una distribución inicial Poisson, puede ser difícil expresar una distribución razonablemente uniforme. Una alternativa es especificar una distribución continua y utilizar la función "*round*" para discretizarla. Por ejemplo, suponga que en lanzamientos sucesivos de una moneda, se observan 10 "*águilas*", ¿cuántas veces se ha lanzado la moneda?

```
model {
          r <- 10
          p <- 0.5
          r ~ dbin(p, n)
          n.cont ~ dunif(1, 100)
          n <- round(n.cont)
}
```

```
node    mean   sd     MC error  2.5%   median  97.5% start  sample
n       21.08  4.794  0.07906   13.00  21.0    32.0  1001   5000
n.cont  21.08  4.804  0.07932   13.31  20.6    32.0  1001   5000
```

Se asume una distribución inicial uniforme para el número de lanzamientos y, como se puede observar en los resultados anteriores, con una probabilidad del 95 % la moneda ha sido lanzada entre 13 y 32 veces. Se pudo haber utilizado una distribución inicial discreta para los enteros en este contexto.

Manejo de bases de datos desbalanceadas

Suponga que se observan los siguientes datos en tres individuos:

```
Persona 1: 13.2
Persona 2: 12.3 , 14.1
Persona 3: 11.0, 9.7, 10.3, 9.6
```

Hay tres diferentes formas de introducir este tipo de datos en WinBUGS.

1. **Construcción de una matriz**. Los datos se expresan explícitamente, incluso los datos que faltan.

```
y[,1]  y[,2]  y[,3]  y[,4]
13.2   NA     NA     NA
12.3   14.1   NA     NA
11.0   9.7    10.3   9.6
END
```

   Otra forma es construir una lista:

```
list(y = structure(.Data = c(13.2, NA, NA, NA, 12.3, 14.1, NA,
NA, 11.0, 9.7, 10.3, 9.6), .Dim = c(3, 4))
```

   En este caso, por ejemplo, en principio es posible ajustar un modelo como $y[i, j] \sim dnorm(mu[i], 1)$; pero se vuelve ineficiente, a menos de que se estimen los datos que faltan.

2. **Anidar índices**. En este caso se almacenan los datos e índices en un arreglo muy simple y a cada persona se le asocia un índice.

```
y[ ]  persona[ ]
13.2    1
12.3    2
14.1    2
11.0    3
9.70    3
10.3    3
9.60    3
  END
```

   Alternativamente, puede usarse una lista:

```
list(y = c(13.2, 12.3, 14.1, 11.0, 9.7, 10.3, 9.6),
persona = c(1, 2, 2, 3, 3, 3, 3))
```

   En este caso es posible ajustar en modelo como $y[k] \sim dnorm(persona[k], 1)$, de manera más eficiente y clara.

Parámetros acerca de la distribución Dirichlet

Suponga que como parte de un modelo hay J arreglos de probabilidades $p[j, 1 : k]$, $j = 1 \ldots, J$; donde K es la dimensión de cada arreglo; además, $sum(p[j, 1 : K]) = 1$ para toda j. A continuación se muestra cada distribución inicial Dirichlet:

```
p[j, 1:K] ~ ddirch(alpha[ ])
```

El parámetro $alpha[\,]$ de una distribución Dirichlet **no** puede ser un nodo estocástico.

El truco consiste en percatarse de que si $delta[k] \sim dgamma(alpha[k], 1)$, entonces el vector con elementos $delta[k]/sum(delta[1 : k])$, $k = 1, \ldots, K$ es Dirichlet con parámetros $alpha[k]$, $k = 1, \ldots, K$. El siguiente código nos permite conocer la construcción del parámetro $alpha[\,]$:

```
for (k in 1:K) {
    p[j, k] <- delta[j, k] / sum(delta[j,])
    delta[j, k] ~ dgamma(alpha[k], 1)
}
```

De esta forma es posible asignar una distribución inicial a las $alpha[k]'s$.

Uso de la función "cut"

Suponga que se observan ciertos datos que no desea que contribuyan a la estimación del parámetro pero sí desea considerarlos como una parte del modelo. Esto puede ocurrir, por ejemplo:

1. Cuando desea hacer una predicción sobre algunos individuos con base en datos parciales que no se quieren utilizar para estimar los parámetros del modelo.

2. Si necesita usar datos para hacer inferencias sobre algunos parámetros en específico.

3. Cuando requiere utilizar la evidencia de una parte del modelo con el fin de construir una distribución inicial para otra parte del mismo.

En el siguiente ejemplo no se modifica la distribución de *theta* al observar y.

```
model {
    y <- 2
    y ~ dnorm(theta.cut, 1)
    theta.cut <- cut(theta)
    theta ~ dnorm(0, 1)
}
```

Usted puede encontrar otros ejemplos, en el directorio de WinBUGS, `C:\Archivos de programa\WinBUGS14\Manuals\Tricks`.

4.3.1. Modo Batch: Scripts

Como una alternativa al uso interactivo del menú y de los cuadros de diálogo, WinBUGS maneja el lenguaje *script*. Este lenguaje es útil cuando se requiere realizar rutinas automáticamente; además es posible combinar el lenguaje *script* con el uso de menús y cuadros de diálogo. Para utilizar el lenguaje *script*, se requieren un mínimo de cuatro archivos:

1. El *script* (.odc o .txt).

2. Un archivo que contenga la representación del modelo en lenguaje BUGS (.txt).

3. Un archivo para los datos (puede ser más de uno)(.txt).

4. Para cada cadena se requiere un archivo con sus respectivos valores iniciales (.txt).

De preferencia guarde todos los archivos en una sola carpeta. Si guarda la carpeta en el directorio de WinBUGS (WinBUGS14), sólo debe especificar la carpeta y el nombre del archivo según el comando que lo requiera dentro del *script*. A continuación se muestra el *script* para el ejemplo de regresión introducido en la sección 4.2.9, cuando toda la información se encuentra en el directorio de WinBUGS.

```
display('log')
check(Ejemplo/Regre_model.txt)
data(Ejemplo/Regre_data.txt)
compile(1)
inits(1,Ejemplo/Regre_init.txt)
update(1000)
set(alpha)
set(beta)
set(tau)
set(sigma)
update(10000)
stats(*)
history(*)
trace(*)
density(*)
autoC(*)
coda(*,Ejemplo/output)
save('Ejemplo/RegresionLog')
```

El contenido de los archivos *Regre_model.txt*, *Regre_data.txt* y *Regre_init.txt* los puede consultar en el apéndice C.

La primera línea *display('log')* indica que la rutina se desplegará en la ventana de *Log*. Note que los demás comandos son los mismos que aparecen dentro de los cuadros de diálogo para revisar el modelo, cargar los datos, registrar las variables, etcétera. Para los comandos que manejan información como: *check*, *data* e *inits*; es necesario especificar dónde se encuentran los archivos, puesto que están contenidos en el directorio WinBUGS14; sólo se indica el nombre de la carpeta y el nombre del archivo en cada comando.

Los comandos cuyo parámetro es "*" funcionan como en los cuadros de diálogo, los resultados se obtienen para todos las parámetros registrados. Si sólo desea hacerlo para uno en específico escriba únicamente el nombre del parámetro de interés.

El comando *coda* genera dos archivos; uno que contiene el nombre de los parámetros y el número de iteraciones, y otro con la cadena de valores de los parámetros. Observe en el código que la instrucción con respecto a *coda* indica que se generan cadenas para todos los parámetros y los archivos generados se almacenan en la carpeta *Ejemplo*. Parte del nombre de estos archivos es *output*, pues WinBUGS agrega otra parte del nombre para distinguir las cadenas generadas y los parámetros; en este caso se generan los archivos *output1.txt* que contiene la cadena y *outpuIndex.txt* que contiene los parámetros y el número de iteraciones. Si el comando se especifica de la forma *coda(*,output)*, los archivos de salida se

guardan en el directorio de WinBUGS14.

El último comando (*save*) indica la dirección en la que se guardan los resultados de la rutina.

Cuando la información no se encuentra dentro del directorio WinBUGS14, debe modificarse el *script* para hacer explícita la dirección en la que se encuentran los archivos. El siguiente código muestra cómo debe construirse el *script* para nuestro ejemplo cuando los archivos se encuentran en la carpeta *Ejemplo* en la unidad de disco *C*.

```
display('log')
check(C:/Ejemplo/Regre_model.txt)
data(C:/Ejemplo/Regre_data.txt)
compile(1)
inits(1,C:/Ejemplo/Regre_init.txt)
update(1000)
set(alpha)
set(beta)
set(tau)
set(sigma)
update(10000)
stats(*)
history(*)
trace(*)
density(*)
autoC(*)
coda(*,C:/Ejemplo/output)
save('C:/Ejemplo/RegresionLog')
```

El archivo que contiene el *script* no necesariamente debe estar en el directorio donde se encuentran los demás archivos.

Para correr un *script* abra el archivo que lo contiene, vaya al menú *Model* y dé *click* en *Script*.

En la Tabla 4.3 se muestra la lista de los comandos para realizar una implementación con el lenguaje *script*. Por cada instrucción se muestra su equivalencia con un menú o un cuadro de diálogo. Primero, aparece el nombre del menú y después, una comando del menú, la palabra en negritas corresponde a un botón en un cuadro de diálogo, mientras que las otras corresponden a un conjunto de campos dentro del cuadro de diálogo.

Si la rutina implementada en el *script* no corresponde a un proceso coordinado, el *script* no se ejecuta y se produce un mensaje de error. Puede ocurrir que no se ejecute una línea; pero si las restantes son correctas, éstas sí se ejecutan.

Cuadro 4.3: Script

Comandos del lenguaje script	
Comando	**Menú o cuadro de diálogo equivalente**
display(*option*)	*Option* → Output options... →*option*="window" o "log"
check(*model file*)	*Model* → Specification... → **check model**
data(*data file*)	Model→Specification... → **load data**
blockfe(*option*)	Options→Blocking options... → fixed effects=> option
compile(*chains*)	Model→Specification... → num of chains=> chain + **compile**
inits(*chain, inits file*)	Model→Specification... → for chain=> chain + **load inits**
gen.inits( )	Model→Specification... → **gen inits**
update(*iteraions*)	Model→Update... → updates=> iterations + **update**
refresh(*every*)	Model→Update... → refresh=> every
beg(*iter*)	Inference→Samples... → beg=> *iter*
end(*iter*)	Inference→Samples... → end=> *iter*
first(*iter*)	Inference→Samples... → chains=> *iter*
last(*iter*)	Inference→Samples... → to=> *iter*
thin.samples(*thin*)	Inference→Samples... → thin=> *thin*
set(*node*)	Inference→Samples... → node=> node + **set**
clear(*node*)	Inference→Samples... → node=> node + **clear**
stas(*node*)	Inference→Samples... → node=> node + **stats**
density(*node*)	Inference→Samples... → node=> node + **density**
autoC(*node*)	Inference→Samples... → node=> node + **auto cor**
trace(*node*)	Inference→Samples... → node=> node + **trace**
history(*node*)	Inference→Samples... → node=> node + **history**
quantiles(*node*)	Inference→Samples... → node=> node + **quantiles**
gr(*node*)	Inference→Samples... → node=> node + **bgr diag**
coda(*node, file stem*)[9]	Inference→Samples... → node=> node + **coda**
set.summary(*node*)	Inference→Samples... → node=> node + **set**
stats.summary(*node*)	Inference→Samples... → node=> node + **stats**
mean.summary(*node*)	Inference→Samples... → node=> node + **mean**
clear.summary(*node*)	Inference→Samples... → node=> node + **clear**
set.rank(*node*)	Inference→Rank... → node=> node + **set**
stats.rank(*node*)	Inference→Rank... → node=> node + **stats**
hist.rank(*node*)	Inference→Rank... → node=> node + **histogram**
clear.rank(*node*)	Inference→Rank... → node=> node + **clear**
dic.set( )	Inference→DIC... → **set**
dic.stats( )	Inference→DIC... → **DIC**
quit( )	File→Exit
save(*file*)[10]	File→Save as... → File name=> file + **Save**
script(*file*)[11]	Model→Script

4.4. Análisis de convergencia

Suponga que se desea generar una muestra de tamaño n de la distribución $p(\boldsymbol{\theta} \mid \boldsymbol{x})$. Si para cada uno de n valores iniciales $\boldsymbol{\theta}_1^{(0)}, \boldsymbol{\theta}_2^{(0)}, \ldots, \boldsymbol{\theta}_n^{(0)}$ se corre el algoritmo de Gibbs, después de cierto número de iteraciones suficientemente grande los valores $\boldsymbol{\theta}_1^{(T)}, \boldsymbol{\theta}_2^{(T)}, \ldots, \boldsymbol{\theta}_n^{(T)}$ pueden considerarse como una muestra de tamaño n de la distribución final de $\boldsymbol{\theta}$. Alternativamente, se puede generar una sola cadena y tomar los valores $\boldsymbol{\theta}_1^{(T+K)}, \boldsymbol{\theta}_2^{(T+2K)}, \ldots, \boldsymbol{\theta}_n^{(T+nK)}$ como una muestra de $p(\boldsymbol{\theta} \mid \boldsymbol{x})$, donde K se elige de manera que la correlación entre las observaciones sea pequeña.

En general no es fácil determinar en qué momento la(s) cadena(s) ha(n) convergido. Un método empírico, comúnmente utilizado, consiste en graficar los promedios ergódicos de algunas funciones de $\boldsymbol{\theta}$ contra el número de iteraciones y elegir el valor T a partir del cual las gráficas se estabilizan. En este caso es frecuente omitir los primeros valores de la(s) cadena(s) al calcular los promedios ergódicos. La idea de este *periodo de calentamiento* es permitir que la(s) cadena(s) salga(n) de una primera fase de inestabilidad. En este caso en particular la velocidad de convergencia depende fuertemente de la correlación entre los componentes del vector $\boldsymbol{\theta}$ bajo la distribución final $p(\boldsymbol{\theta} \mid \boldsymbol{x})$; entre más alta sea la correlación más lenta será la convergencia.

¿Cómo se utilizan las muestras para hacer inferencias?

Una vez que se tiene una muestra de la distribución final de $\boldsymbol{\theta}$ es posible aproximar esencialmente cualquier característica de dicha distribución. Por ejemplo, ciertas medidas de tendencia central (generalmente utilizadas como estimadores puntuales de $\boldsymbol{\theta}$) pueden aproximarse utilizando las cantidades muestrales correspondientes. De manera similar los cuantiles de la distribución final de $\boldsymbol{\theta}$ pueden utilizarse para construir intervalos de credibilidad para $\boldsymbol{\theta}$.

4.4.1. BOA

Boa es un paquete que se puede instalar en R[12] o S-Plus proporciona resultados para analizar la convergencia, así como para producir estadísticas y análisis gráficos

[8] Si *file stem* se encuentra en blanco, se despliegan los resultados en WinBUGS, una ventana para cada cadena y otro para el índice (*Index*) que contiene el nombre de los parámetros y el número de iteraciones, de otra forma se generan archivos de texto.

[9] Si el nombre del archivo lleva extesión *.txt* los resultados se guardan en un archivo de texto, si no lleva extensión la rutina se guarda como un archivo *.odc*.

[10] Corre el *script* que se encuentra en el archivo.

[12] R es un lenguaje para el análisis de datos y gráficas. Al igual que WinBUGS es de libre acceso en la red.

usando muestras de Cadenas de Markov vía Monte Carlo. Algunos resultados que proporciona *boa* también los puede producir WinBUGS; sin embargo, los gráficos no son tan escuetos como en WinBUGS y se pueden realizar análisis más completos. *Boa* puede procesar los resultados que arroja WinBUGS. El procedimiento para importar los datos que genera WinBUGS a *boa* por medio de R y para obtener algunos resultados usando estos datos desde R, se ilustrará a través del ejercicio de regresión que se ha usado en secciones anteriores. El procedimiento se muestra a continuación:

- Para cargar el paquete en R, vaya al menú *Packages*, dé *click* en el comando *Load packages...* para desplegar una lista, seleccione *boa* y dé *click* en *OK*[13].

- Para accesar a *boa* tecleé `boa.menu()` más *ENTER*. Al teclear "1", se abrirá otro menú para importar los datos.

```
BOA MAIN MENU
*************
1:File     >>
2:Data     >>
3:Analysis >>
4:Plot     >>
5:Options  >>
6:Window   >>
Selection: 1
```

- Elija la opción "3", `Import Data`.

```
FILE MENU
=========
1:Back
2:-----------------------+
3:Import Data         >> |
4:Load Session           |
5:Save Session           |
6:Exit BOA               |
7:-----------------------+
Selection: 3
```

[13]Previamente tuvo que ser instalado el paquete directamente de *internet* o de un archivo. En el menú *Packages* aparecen las opciones que puede utilizar para instalar *boa*.

- En el siguiente menú tecleé "7", para desplegar las opciones respecto al tipo de archivo.

```
IMPORT DATA MENU
----------------
1:Back
2:---------------------------+
3:CODA Output Files          |
4:Flat ASCII File            |
5:Data Matrix Object         |
6:View Format Specifications |
7:Options...                 |
8:---------------------------+
Selection: 7

Data Parameters
===============

Files
-----
1) Working Directory: ""
2) ASCII File Ext:    ".txt"
```

- Tecleé "1" para especificar el directorio en el que se encuentran los archivos generados por WinBUGS[14]. Como se muestra a continuación, estos archivos se encuentran en la carpeta *Ejemplo* en la unidad de disco *C*.

```
DESCRIPTION: Specified directory must not end with a slash

Enter new character string
1: C:\Ejemplo
Read 1 items
```

- Una vez que R haya reconocido el directorio, aparece otro menú. Ahora tecleé "3" y después proporcione el nombre de los archivos que previamente

[14]La opción *coda* que se encuentra en el menú *Inference* en el comando *Sample* genera estos archivos con las extensiones *.out* y *.ind*. En el primero se despliega la cadena simulada y en el otro los parámetros y el número de iteraciones. Ambos archivos deben llevar el mismo nombre base.

se generaron en WinBUGS. Como los archivos llevan el mismo nombre y no se deben especificar las extensiones, sólo tecleé el nombre. Si los datos han sido importados exitosamente, se despliegan los mensajes que se muestran en el siguiente código.

```
IMPORT DATA MENU
----------------
1:Back
2:---------------------------+
3:CODA Output Files          |
4:Flat ASCII File            |
5:Data Matrix Object         |
6:View Format Specifications |
7:Options...                 |
8:---------------------------+
Selection: 3

Enter filename prefix without the .ind or .out extension
[Working Directory: "C:\\Ejemplo"]

1: Regresion
Read 1 items
Read 1 records
Read 3000 records
+++ Data successfully imported +++
```

- Regrese al menú inicial (`BOA MAIN MENU`) a través de `1:Back` para proceder con el análisis.

El menú `DATA`, que se muestra a continuación, le permite modificar los datos del modelo, es decir, puede incluir un nuevo parámetro o eliminar un parámetro. Para nuestro ejemplo, se maneja a *tau* como la precisión. Suponiendo que desea obtener la varianza y que no se incluyó en el modelo, lleve a cabo la instrucción `PARAMETERS` → `NEW`, proporcione el nombre del parámetro (*sigma*) y la función del mismo (*sigma*=1/*tau*); de esta manera podrá obtener los resultados para la varianza.

```
DATA MANAGEMENT MENU
====================
1:Back
2:---------------------------+
3:Chains                  >> |
```

```
4:Parameters              >> |
5:Display Working Dataset    |
6:Display Master Dataset     |
7:*****                      |
8:---------------------------+
```

El menú `Analysis`, contiene las siguientes opciones.

```
DESCRIPTIVE STATISTICS MENU
---------------------------
1:Back
2:-------------------------------------+
3:Autocorrelations                     |
4:Correlation Matrix                   |
5:Highest Probability Density Intervals  |
6:Summary Statistics                   |
7:-------------------------------------+
```

`Autocorrelations` permite desplegar los valores de autocorrelación para 5, 10 y 50 rezagos (*lags*); mientras que WinBUGS sólo proporciona las gráficas para 50 rezagos. A través de este menú también se puede obtener la matriz de correlaciones y los intervalos al 95 % de probabilidad. Estos dos últimos resultados también se pueden obtener en WinBUGS, así como el resumen de las estadísticas `Summary Statistics`, pero *boa* incluye en el resumen: `Naive, SE, Batch SE` y `Batch ACF`.

La opción `Plot` del menú principal (`BOA MAIN MENU`) le permite obtener distintas gráficas.

```
PLOT MENU
=========
1:Back
2:---------------------------+
3:Descriptive             >> |
4:Convergence Diagnostics >> |
5:Options...                 |
6:---------------------------+
```

A través del comando `Descriptive` puede generar las gráficas de autocorrelación, densidad, promedios ergódicos y la traza. `Converge Diagnostics` le permite obtener resultados para verificar la convergencia mediante distintos métodos: Brooks & Gelman, Gelman & Rubin y Geweke[15].

[15]Para realizar el diagnóstico de covergencia necesita generar al menos dos cadenas y pegar las cadenas en un sólo archivo que deberá llevar extensión *.out*.

El comando `Options` del menú principal de *boa*, le permite manipular diferentes parámetros respecto a los menús `Data, Analysis, Plot`, tales como el nivel de los intervalos, los cuartiles, los rezagos para calcular la autocorrelación, y también se pueden modificar las gráficas. `Options` aparece como otra opción dentro de los menús `Analysis` y `Plot`.

```
 GLOBAL OPTIONS MENU
====================
1:Back
2:------------+
3:Analysis... |
4:Data...     |
5:Plot...     |
6:All...      |
7:------------+
```

4.5. R2WinBUGS

R2WinBUGS es un paquete que proporciona funciones para interactuar con WinBUGS desde R. Produce automáticamente un *script* en un formato que WinBUGS puede leer y ejecutar. Después de que WinBUGS termina el proceso, es posible leer los resultados en R, con el fin de generar gráficas y algunas estadísticas; además facilita el uso de los paquetes *boa* y *coda*.

En esta sección se usará el ejemplo de regresión introducido en la sección 4.2.9 para ilustrar de manera muy general cómo funciona R2WinBUGS.

Al igual que *boa*, R2WinBUGS lo puede instalar desde un archivo o por *internet* a través del menú *Package*; y del mismo modo que *boa*, para cargar el paquete vaya al menú *Package→Load package...* seleccione R2WinBUGS y dé *click* en *OK*.

Una vez que se ha cargado el paquete en R, existen varias formas de manejar la información del modelo. Por ejemplo, para los datos del modelo puede crear un archivo de texto desde R, puede leer un archivo ya creado o puede manejar la información en la consola de R.

Si lo hace en la consola de R los valores iniciales se especifican de la siguiente forma.

```
inits = list(ini=list(alpha=0,beta=0,tau=1))
```

También indique para qué parámetros se llevará a cabo el proceso. Este comando es similar al botón *set* del cuadro de diálogo *Samples*. Ver la sección 4.2.5.

```
parameters = c("alpha", "beta", "tau")
```

Los datos del modelo se especifican de la siguiente manera.

```
data = list(N=5,x = c(8.0, 15.0, 22.0, 29.0, 36.0),
y = c(151, 199, 246, 283, 320))
```

A continuación se indica el número de iteraciones (*n.iter*), así como las iteraciones correspondientes al *periodo de calentamiento*(*n.burnin*). Si no indica el número de iteraciones, se ejecutan por defecto 1000 para el número de iteraciones y 1000 para el *periodo de calentamiento*.

```
n.iter = 11000
n.burnin = 1000
n.thin = 1
```

Además, es necesario crear un archivo de texto que incluya el modelo en lenguaje BUGS, por ejemplo "Regre_model.txt"; que es el mismo archivo que se usó en la sección 4.3.1 para la construcción del *script* (consulte el apéndice C).

Para realizar la simulación MCMC escriba el siguiente código en R.

```
>prueba<-bugs(data, inits,
+model.file = "c:/Ejemplo/Regre_model.txt",
+parameters ,n.chains = 1,
+bugs.directory = "c:/Archivos de programa/WinBUGS14/")
```

Esta instrucción crea automáticamente un archivo de texto con un *script* en el directorio de WinBUGS.

El argumento *bugs.directory* corresponde al directorio en el cual fue instalado WinBUGS. Los resultados obtenidos por la simulación se pueden obtener tecleando *print(prueba)*.

```
>print(prueba)
Inference for Bugs model at "c:/Ejemplo/Regre_model.txt"
 1 chains, each with 2000 iterations (first 1000 discarded)
 n.sims = 1000 iterations saved
          mean   sd  2.5%    25%    50%    75%  97.5%
alpha    106.7  9.0  87.3  102.8  107.2  110.8  122.6
beta       6.0  0.4   5.4    5.9    6.0    6.2    6.7
tau        0.0  0.0   0.0    0.0    0.0    0.1    0.1
deviance  32.2  3.6  28.3   29.6   31.3   33.7   42.0

pD = 6.6 and DIC = 38.8 (using the rule, pD = var(deviance)/2)
DIC is an estimate of expected predictive error (lower deviance
is better).
```

También puede obtener una gráfica que resume brevemente inferencias y diagnósticos de convergencia para todos los parámetros, al teclear *plot(prueba)*; así como un resumen de las estadísticas al usar la instrucción *summary(prueba)*.

Si desea que los paquetes *boa* o *coda* lean los datos generados por WinBUGS. Debe incluir *codaPkg=TRUE*, en el código anterior. Sin embargo, ya no podrá generar resultados a través de *print*, *plot* y *summary*, a menos de que especifique *codaPkg=FALSE*.

```
>prueba.sim<-bugs(data, inits,
+model.file = "c:/Ejemplo/Regre_model.txt",
+parameters ,n.chains = 1, codaPkg=TRUE,
+bugs.directory = "c:/Archivos de
+programa/WinBUGS14/")
```

La siguientes instrucciones le permitirán utilizar el paquete *boa*. Si desea usar *coda* tecleé (*library(coda)*).

```
> library(boa)
> boaobject<-read.bugs(prueba)
```

Para ingresar al menú de *boa* tecleé `boa.menu()`. La forma de importar los datos cambia respecto a como se hizo en la sección 4.4.1. Seleccione *1:File*→*3:Import Data*→*5:Data Matrix Object*→Tecleé el nombre de objeto, *boaobject*, que se creó con la instrucción *read.bugs()*. Una vez que haya realizado este procedimiento podrá obtener las gráficas y resultados que proporciona este paquete.

4.6. Aplicaciones

El objetivo de esta sección es presentar dos problemas reales que permiten ilustrar cómo se pueden analizar de manera relativamente sencilla por medio de WinBUGS.

4.6.1. Participación electoral: el caso de Estados Unidos

La construcción de esta aplicación parte del artículo *Making the Most of Statistical Analyses: Improving Interpretation and Presentation*, King et al. (2000), donde se estudia la probabilidad de que una persona vote según sus características sociodemográficas.

Para llevar a cabo el análisis se utilizaron los datos de una encuesta postelectoral elaborada por un estudio de elecciones nacionales en Estados Unidos (*US National Election Study*), la cual fue realizada en un año de elecciones presidenciales. Posteriormente, se efectuó el mismo análisis para las elecciones presidenciales de México del año 2000; sin embargo se tomaron en cuenta otras consideraciones que se mencionan en la sección 4.6.2.

El análisis se hizo con base en un modelo *logit*, donde Y_i es la variable dependiente y mide el número de personas que votaron con las mismas características sociodemográficas.

$$Y_i \sim Bin(N, p) \quad i = 1, 2, \ldots, n.$$

El problema se discute con detalle en Rosenstone y Hansen (1993), sin embargo al igual que en King et al. (2000) en este trabajo sólo se consideran las siguientes variables sociodemográficas: edad, años de educación, ingreso en 10,000 de dólares y la raza (1 identifica a los blancos, 0 a otra raza). También se incluye un término cuadrático (edad al cuadrado entre cien) bajo la hipótesis de que cuando el encuestado está en edad de retiro, la tendencia se revierte.

$$L_i = ln\left(\frac{p_i}{1-p_i}\right) \quad i = 1, 2, \ldots, n$$

$$L_i = \beta_{2i}\text{raza}_i + \beta_{1i}\text{edad}_i + \beta_{3i}\text{educación}_i + \beta_{4i}\frac{\text{edad}^2}{100} + \beta_{5i}\text{ingreso}_i + \beta_{6i} \quad (4.1)$$

La probabilidad se estima en dos diferentes niveles de educación de acuerdo a la edad. El primer nivel considera a las personas que tienen menos o 12 años de educación y el segundo nivel a las personas que cuentan con más de 12 años de educación, mientras se mantiene a las otras variables en sus medias por nivel de educación. Por otro lado, como la raza es una variable dicotómica, se decidió hacer la

evaluación primero para la raza blanca y después para otra raza. En la Figura 4.44 se puede observar la codificación del modelo en WinBUGS y los resultados se ven en la Figura 4.45. Como se puede apreciar la probabilidad incrementa ligeramente para las personas que son blancas.

WinBUGS

La especificación del modelo en lenguaje BUGS se muestra a continuación:

```
model{
        for( i in 1:13464) {
                turnout[i] ~ dbin(p[i],N[i])
                logit(p[i])<- y[i]
                y[i] <- beta[1]*raza[i] +beta[2]*edad[i] + beta[3]*educacion[i]+
                        beta[4]*pow(edad[i],2)/100+beta[5]*ingreso[i] +beta[6]
        }
  # Cálculo de la probabilidad para las personas con menos de secundaria completa
        for( i in 1:79) {
                logit(p.pm[i])<- p.ym[i]
                p.ym[i] <- beta[1]*raza.m +beta[2]*edad.m[i] +beta[3]*educacion.m +
                        beta[4]*pow(edad.m[i],2)/100+ beta[5]*ingreso.m +beta[6]
        }
  # Cálculo de la probabilidad para las personas con  más de secundaria completa
        for( i in 1:79) {
                logit(p.ph[i])<- p.yh[i]
                p.yh[i] <- beta[1]*raza.h +beta[2]*edad.h[i] +  beta[3]*educacion.h +
                        beta[4]*pow(edad.h[i],2)/100+ beta[5]*ingreso.h +beta[6]
        }
              beta[1:6] ~dmnorm(mu[ ],B[ , ])
}
```

La figura 4.45 que muestra las gráficas fueron elaboradas en R debido a que los resultados se analizaron con bandas del 99% de probabilidad y WinBUGS no cuenta con las herramientas necesarias para producirlas, una vez analizado el problema en WinBUGS se generaron las cadenas de los parámetros a través del comando *coda* (Menú *Inference->Samples...->coda*) y se analizaron a través de *boa* para realizar las gráficas.
También se obtuvieron resultados para la convergencia y la autocorrelación de las variables (Menú *Inference-> Samples...->trace* y Menú *Inference->Samples...->auto cor*)

Figura 4.44: *Codificación.*

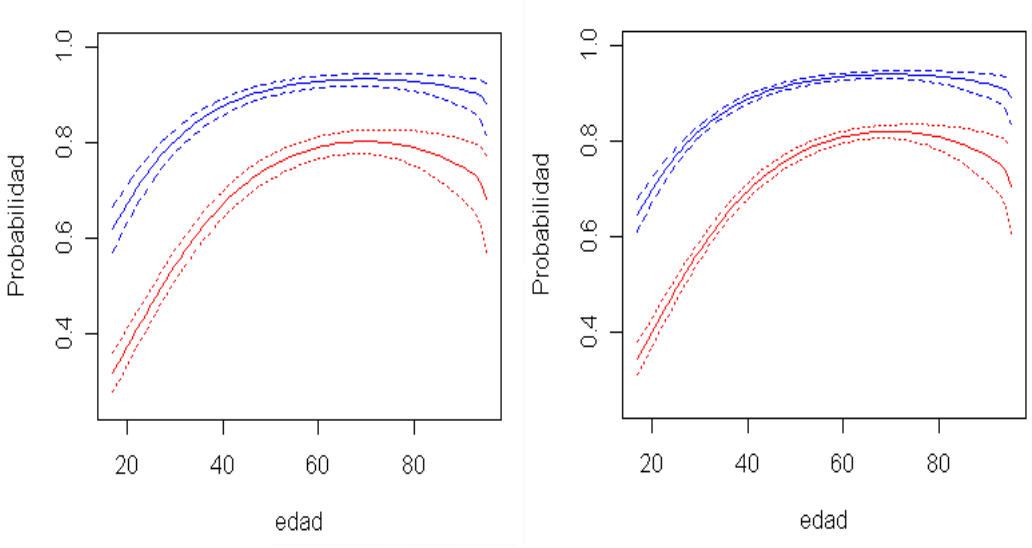


Figura 4.45: *Probabilidad de votar respecto a la edad por nivel de educación. Bandas 99 % de probabilidad, otra raza (izquierdo), raza blanca (derecho).*

En el apéndice D.1 podrá consultar el análisis respecto a la convergencia y a la autocorrelación de las variables en donde se observa que el término independiente, la edad y la edad al cuadrado presentan problemas de autocorrelación y convergencia; por lo que se decidió omitir el término cuadrático del modelo. Los resultados mejoraron de manera considerable. Por esta razón, las conclusiones se darán con base en el siguiente modelo.

$$L_i = ln\left(\frac{p_i}{1-p_i}\right) \qquad i = 1, 2, \ldots, n$$

$$L_i = \beta_{1i}\text{raza}_i + \beta_{2i}\text{edad}_i + \beta_{3i}\text{educación}_i + \beta_{4i}\text{ingreso}_i + \beta_{5i} \tag{4.2}$$

Verba y Nie (1972) plantean una relación entre los niveles de participación y los niveles de educación, ingreso, edad entre otras variables. Con respecto a la edad, a medida que los individuos son mayores, asumen responsabilidades sociales (pagar impuestos, obtener beneficios de programas oficiales, etc.) que influyen en su motivación para participar en la política. Por otro lado, la educación también afecta la participación, ya que "la absorción de los mensajes y símbolos políticos denota un compromiso intelectual o cognitivo con los asuntos políticos", Zaller (1992).

Como se puede observar en la Figura 4.46, la probabilidad de votar es mayor conforme aumenta la edad, mientras que en los extremos se incrementa la incertidumbre. Además es evidente la diferencia respecto a la probabilidad de voto,

según el nivel de educación, por lo que los resultados obtenidos apoyan las hipótesis antes mencionadas. Respecto a la evaluación del modelo según las razas se observa un comportamiento similar al del primer modelo. Por otro lado, la codificación en WinBUGS de este modelo es análoga al modelo que se presenta en la Figura 4.44, salvo que no se considera el término cuadrático.

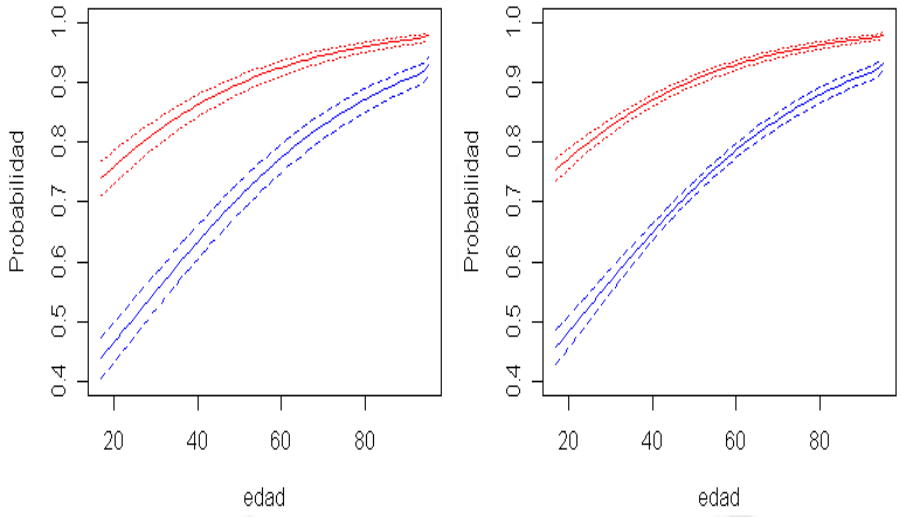


Figura 4.46: *Probabilidad de votar respecto a la edad por nivel de educación. Bandas 99 % de probabilidad, otra raza (izquierdo), raza blanca (derecho).*

4.6.2. Participación electoral: el caso de México

La elección presidencial del año 2000 en México marcó el fin de 71 años de gobierno priísta. Representó una de las elecciones más importantes en la historia electoral de México, aún cuando la participación electoral disminuyó con respecto a la elección anterior de 77 a 64 por ciento. Tomando como base el ejemplo de King et al. (2000), se pretende ilustrar la probabilidad de voto del mexicano utilizando características sociodemográficas muy similares:

Edad: de 18 a 81, 84 y 88 años.

Educación: sin instrucción (1), primaria incompleta (2), primaria completa (3), secundaria incompleta (4), secundaria completa (5), estudios posteriores a secundaria (6), universidad incompleta (7), universidad completa (8).

Ingreso familiar: medido en quintiles.

Raza: aún cuando en México existen distintas razas en esta encuesta no se hace distinción entre ellas.

Los datos analizados provienen de la versión mexicana del *Comparative Study of Electoral System for 2000* (Estudio compartivo de Sistemas Electorales del 2000). Es una encuesta nacional aleatoria de 1766 observaciones realizada días posteriores a la elección presidencial del 2 de julio de 2000. El trabajo de campo lo llevó a cabo Consulta S.A. de C.V. Para reducir la sobre-reportación respecto a la partipación electoral, se le preguntó a la gente *1)* si había votado en la elección presidencial; *2)* si tenían una credencial para votar y si podían mostrarla a los entrevistadores. La gente que votó debía tener una marca en su credencial que registrara que había votado en la elección. Si la gente dijo que había votado pero mostraba su credencial y no tenía ninguna marca, se le consideraba no votante. En un buen número de casos, los ciudadanos dijeron haber votado, pero no tenían con ellos su credencial para votar, por lo que se decidió considerarlos como votantes. Esto condujo a una sobrerepresentación de 12 puntos porcentuales de la cifra agregada real de participación, que está dentro de los límites de sobre-reporte estándar de las encuestas postelectorales tales como los estudios de elecciones nacionales en Estados Unidos *(US National Election Studies)*.

Del mismo modo que para el caso de Estados Unidos, se ajustó un modelo *logit*. Los niveles de educación se dividen en: menos de secundaria completa y más de secundaria.

$$L_i = ln\left(\frac{p_i}{1-p_i}\right) \qquad i = 1, 2, \ldots, n$$

$$L_i = \beta_{1i}\text{edad}_i + \beta_{2i}\text{educación}_i + \beta_{3i}\frac{\text{edad}^2}{100} + \beta_{4i}\text{ingreso}_i + \beta_{5i} \tag{4.3}$$

En la Figura 4.47 se muestra la codificación en WinBUGS del modelo y en la gráfica 4.48 se pueden observar las probabilidades obtenidas a partir de este modelo.

Al igual que el caso de Estados Unidos, el modelo presentó problemas de convergencia y autocorrelación (ver apéndice D.2) por lo que el análisis se efectuó con base en el siguiente modelo.

$$L_i = ln\left(\frac{p_i}{1-p_i}\right) \qquad i = 1, 2, \ldots, n$$

$$L_i = \beta_{1i}\text{edad}_i + \beta_{2i}\text{educación}_i + \beta_{3i}\text{ingreso}_i + \beta_{4i} \tag{4.4}$$

WinBUGS

La especificación del modelo en lenguaje BUGS se muestra a continuación:

```
model{
        for( i in 1:1013) {
                turnout[i] ~ dbin(p[i],N[i])
                logit(p[i])<- y[i]
                y[i] <- beta[1]*edad[i] + beta[2]*educacion[i] + beta[3]*pow(edad[i],2)/100+
                        beta[4]*ingreso[i]+beta[5]
        }
  # Cálculo de la probabilidad para las personas con menos de 12 años de educación
        for( i in 1:71) {
                logit(p.ym[i])<- p.ym[i]
                p.ym[i] <- beta[1]*edad.m[i] + beta[2]*educacion.m[i] + beta[3]* pow(edad.m[i],2)/100+
                        beta[4]*ingreso.m + beta[5]
        }
  # Cálculo de la probabilidad para las personas con más de 12 años de educación

        for( i in 1:71) {
                logit(p.ph[i])<- p.yh[i]
                p.yh[i] <- beta[1]*edad.h[i] + beta[2]*educacion.h + beta[3]*pow(edad.h[i],2)/100+
                        beta[4]*ingreso.h + beta[5]
        }
                beta[1:5] ~dmnorm(mu[],B[,])
}
```

La figura 4.48 que muestra las gráficas fueron elaboradas en R debido a que los resultados se analizaron con bandas del 99% de probabilidad y WinBUGS no cuenta con las herramientas necesarias para producirlas, una vez analizado el problema en WinBUGS se generaron las cadenas de los parámetros a través del comando *coda* (Menú *Inference->Samples...->coda*) y se analizaron a través de *boa* para elaborar las gráficas.
También se obtuvieron resultados para la convergencia y la autocorrelación de las variables (Menú *Inference-> Samples...->trace* y Menú *Inference->Samples...->auto cor*)

Figura 4.47: *Codificación.*

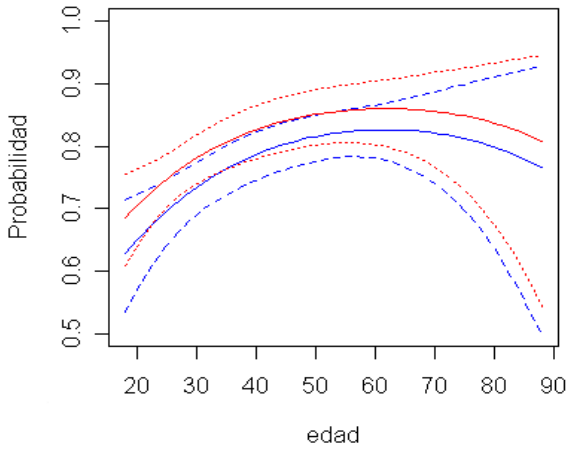


Figura 4.48: *Probabilidad de votar respecto a la edad por nivel de educación. Bandas 99 % de probabilidad.*

La codificación de este modelo es similar al de la Figura 4.47, salvo que no se incluye el término cuadrático.

Los resultados de este modelo muestran que la edad es un factor importante respecto a la probabilidad de voto, ver Figura 4.49. Sin embargo el efecto del nivel de educación no es significativo, por lo que la educación no es un factor determinantemente en la participación electoral. Este comportamiento podría estar permeado por otros factores que influyen con mayor fuerza en la participación electoral, como la movilización[16] que precisamente en las elecciones del año 2000 tuvo gran influencia en la participación electoral de la sociedad mexicana.

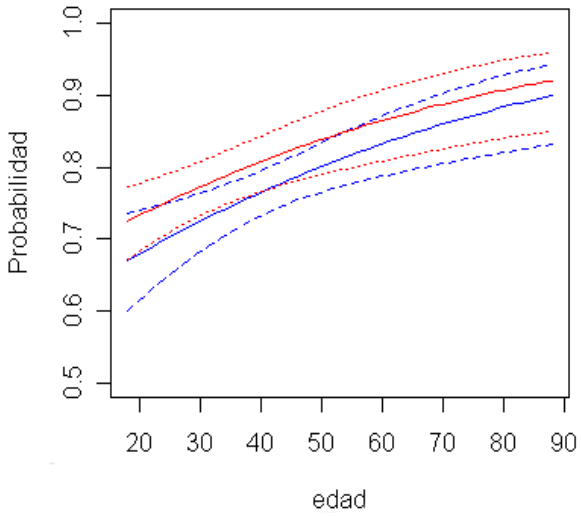


Figura 4.49: *Probabilidad de votar respecto a la edad por nivel de educación. Bandas 99 % de probabilidad.*

En conclusión, si bien es cierto que las variables que se analizan en torno a los dos casos no son idénticas, son lo suficientemente parecidas como para comparar el comportamiento de la participación electoral. En los datos analizados es claro que en Estados Unidos la educación es un factor determinante en la participación electoral, mientras que para México en particular en las elecciones del 2000 no fue así.

[16]La movilización se refiere a "el proceso mediante el cual candidatos, partidos, activistas y grupos inducen a las personas a participar". Por lo que la movilización involucra solicitudes de voto casa por casa, mediante correo, obsequios, mediante anuncios de radio y televisión, etc.

Capítulo 5

Conclusiones

A lo largo de este trabajo hemos demostrado que un *software* como WinBUGS puede ser de gran utilidad para quienes suelen hacer aplicaciones a través de los métodos Bayesianos. Esto se debe a que aprender a utilizar el paquete no lleva mucho tiempo y es relativamente sencillo. Por otro lado, la construcción de los modelos, en particular a través de los gráficos de WinBUGS, permite su formulación de una manera más clara e ilustrativa. Lo que resulta una gran ventaja, ya que el usuario invierte más tiempo en la construcción del modelo y en el análisis de los resultados del mismo, que si tuviera que realizar la codificación MCMC.

Durante el desarrollo del presente trabajo se han encontrado algunas deficiencias del *software* con respecto al manejo de datos. Específicamente, para volúmenes muy grandes de información la simulación puede ser muy lenta y una de las razones es que en ocasiones WinBUGS no es capaz de resumir toda esta información. Sin embargo, es posible utilizar un *software* alternativo (tal como R) para procesar los datos y posteriormente analizarlos en WinBUGS.

Como advierten encarecidamente sus autores en el manual, la simulación MCMC puede ser peligrosa. Si hay algún fallo en el modelo, puede aparecer una ventana de *trap* que usualmente para un usuario común es difícil decifrarla. Más grave aún es el caso en el que un modelo mal especificado corre sin producir ninguna advertencia y se produzcan resultados espurios. Por esta razón, es importante tener claramente establecido el modelo y por otro lado se debe analizar cuidadosamente los resultados obtenidos después de un número conveniente de simulaciones.

Si bien es cierto que WinBUGS es un *software* en desarrollo, la implementación del algoritmo de Gibbs es de gran ayuda para la construcción de modelos. WinBUGS utiliza el método de muestreo más adecuado para diferentes tipos de modelos. En algunos casos no se puede muestrear directamente; sin embargo, WinBUGS emplea alguna forma del muestreo de Gibbs o de Metropolis-Hastings

para muestrear la distribución final. Por otra parte, continuamente se crean interfaces con otros programas, tales como *boa* y R2WinBUGS, que permiten obtener resultados más completos. Una ventaja es que no sólo WinBUGS es *software* libre, también R, *boa* y R2WinBGS lo son.

Finalmente es importane mencionar que recientemente se ha buscado que el *software* trabaje en otras plataformas, lo que motivó la creación de OpenBUGS (http://www.mrc-bsu.cam.ac.uk/), que es una versión de código abierto de WinBUGS para Linux y otros sistemas operativos.

Apéndice A

Resultados Preliminares

A.1. Cadenas de Markov

Una parte importante de la teoría de probabilidad la constituyen los procesos estocásticos en los cuales se modelan los fenómenos aleatorios que evolucionan con el tiempo. Tal es el caso de las Cadenas de Markov, para las que la evolución en el tiempo tiene la característica de que el futuro solamente depende del presente y no de toda la historia del fenómeno en cuestión. El estudio de estos procesos lo inició el matemático ruso A. A. Markov a principios del siglo XX.

Sea (Ω, Φ, P) un espacio de probabilidad y E un conjunto no vacío, finito o numerable. Para facilitar la exposición, sólo se discutirá el caso finito. Una sucesión de variables aleatorias

$$\{X_n : \Omega \to E, \quad n = 0, 1, \ldots, k\} \quad k < \infty$$

se llama cadena de Markov con espacio de estados E si satisface la condición de Markov, esto es, si para todo $n \geq 1$ y toda sucesión $i_0, i_1, \ldots, i_{n-2}, i, j \in E$

$$Pr(X_n = j | X_{n-1} = i, \ldots, X_0 = i_0) = Pr(X_n = j | X_{n-1} = i) \equiv P_{ij} \qquad \text{(A.1)}$$

La distribución de X_0 se llama *distribución inicial* y se denota como π. Las probabilidades P_{ij} se suelen llamar *probabilidades de transición* y se representan por medio de una matriz.

$$T = \begin{bmatrix} P_{11} & \ldots & P_{1k} \\ \vdots & \ddots & \vdots \\ P_{k1} & \ldots & P_{kk} \end{bmatrix}$$

A continuación se enuncian una serie de definiciones y teoremas a manera de resumen, para describir conceptos relacionados con las cadenas de Markov.

- E es un subconjunto de los números enteros. En estos casos E es un conjunto ordenado.
- La distribución de una variable aleatoria discreta $X : \Omega \to E$ es $(Pr(X = i))_{i \in E}$. En el caso $E \subseteq Z$ la distribución se considera como vector de dimensión igual a la cardinalidad de E.
- La igualdad A.1 se refiere a que la probabilidad de encontrarse en el estado j al instante n, dado que en los instantes anteriores de la cadena siguió la trayectoria $\{i_0, \ldots, i_{n-2}, i\}$, sólo depende del estado inmediato anterior, es decir, del estado en el instante $n - 1$.
- A la familia $\{Pr(X_n = j | X_{n-1} = i); n \in N, i, j \in E\}$ se le llama *familia de probabilidades de transición* de la cadena, la cual describe la evolución de la misma en el tiempo.
- Si $Pr(X_n = j | X_{n-1} = i)$ no depende de n se dice que la cadena es *homogénea* con respecto al tiempo. Para una cadena de Markov homogénea se denota a $Pr(X_n = j | X_{n-1} = i)$ como $P_{i,j}$.
- $Pr(X_{n+m} = j | X_n = i)$ se denota como $P_{i,j}^{(m)}$ y es la probabilidad de ir en m pasos o unidades de tiempo de i a j, *probabilidad de transición* en m pasos.
- A la matriz $T = (P_{i,j})_{i,j \in E}$ y se le llama *matriz de transición*.
- Para $i, j \in E$ se define a $P_{i,j}^{(0)}$ como $\delta_{i,j}$, donde $\delta_{i,j}$ es la delta de Kronecker, es decir, vale 1 si $i = j$ y vale 0 si $i \neq j$.
- Si la distribución inicial π es igual al vector $(\delta_{i,j})_{j \in E}$, es decir,

$$Pr(X_0 = i) = 1 \quad y \quad Pr(X_0 \neq i) = 0$$

se dice que la cadena empieza en i.

- La suma de las entradas de los renglones de la matriz de transición es igual a uno, es decir, para todo $i \in E$ se tiene $\sum_{j \in E} P_{i,j} = 1$.

Ejemplo

Suponga que en el periodo $[n, n + 1]$, una línea telefónica puede estar en dos estados: ocupada$\{0\}$ y desocupada$\{1\}$. Para todo el periodo $[n, n+1]$ la probabilidad de que llegue una llamada es $p \in (0, 1)$. Si la línea está ocupada no se registra la llamada. La probabilidad de que se desocupe es $q \in (0, 1)$.

En cada periodo puede llegar una llamada o se puede desocupar la línea (pero no ambas).

El espacio de estados es $E = \{0, 1\}$

$$\begin{aligned} P_{0,0} &= Pr(X_{n+1} = 0 | X_n = 0) = 1 - q \\ P_{0,1} &= Pr(X_{n+1} = 1 | X_n = 0) = q \\ P_{1,0} &= Pr(X_{n+1} = 0 | X_n = 1) = p \\ P_{1,1} &= Pr(X_{n+1} = 1 | X_n = 1) = 1 - p \end{aligned}$$

La matriz de transición se representa de la siguiente forma:

$$T = \begin{bmatrix} 1-q & q \\ p & 1-p \end{bmatrix}$$

La distribución inicial para esta matriz de transición es:

$$\pi(0) = Pr(X_0 = 0)$$

$$\pi(1) = Pr(X_0 = 1)$$

- Un estado es *absorbente* cuando el sistema nunca lo abandona una vez que llega a él.

- Clasificación de estados.

  - Se dice que el estado j accede al estado i si $P_{ij}^n > 0$ para $n \geq 0$ y se representa como $j \to i$.
  - Dos estados i y j que acceden uno al otro se dice que comunican entre sí y se escribe $i \leftrightarrow j$.

    Por definición, un estado se comunica con él mismo, si

    $$P_{ii}^0 = Pr\left[X_0 = i | X_0 = i\right] = 1.$$

    La relación de comunicación es una relación de equivalencia y por lo tanto satisface las siguiente propiedades:

    (i) El estado i comunica con el estado i, para todo $i \geq 0$.

    (ii) Si el estado i comunica con el estado j, entonces el estado j comunica con el estado i.

    (iii) Si el estado i comunica con el estado j, y el estado j comunica con el estado k, entonces el estado i comunica con el estado k.

Si dos estados que se comunican se dice que se encuentran en la misma clase. El concepto de comunicación entre estados divide al espacio en un número de clases separadas. Una cadena de Markov es irreducible si sólo existe una única clase, es decir, si todos los estados se comunican uno con otro. Considere la cadena de Markov que consiste en cuatro estados 0, 1, 2, 3, y tiene una matriz de probabilidades de transición.

$$T = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Las clases de esta cadena de Markov son: $\{0, 1\}$, $\{2\}$ y $\{3\}$. Note que mientras el estado 0 (o 1) es accesible desde el estado 2, al revés no se cumple esta condición. Y el estado 3 es absorbente, ningún estado puede accederse desde éste.

- Un estado es recurrente si una vez que se sale de él, se puede volver a éste con probabilidad 1 ($P_{ii}^{(n)} = 1$). Si la probabilidad es menor a 1 se dice que el estado es transitorio ($P_{ii}^{(n)} < 1$).

- El estado i tiene periodo d si $P_{ii}^{(n)} = 0$ siempre que n no sea divisible por d, y d sea el entero más grande con esta propiedad. Por ejemplo, al estar en i, es posible entrar al estado i sólo en los tiempos $2, 4, 6, \ldots$; en cuyo caso estado i tiene periodo 2. Un estado con periodo 1 se dice que es *aperiódico*. Se puede demostrar que la periodicidad es propiedad de la clase. Esto es, si el estado i tiene periodo d, y los estados i y j comunican entres sí, entonces el estado j también tiene periodo d.

- Si un estado i es recurrente, entonces se dice que un estado es *recurrente positivo* si, estando en i, el valor esperado del tiempo de regresar a i es finito. Por lo tanto, esta es una propiedad de una clase. A los que son recurrentes y no son positivos se les llama *recurrentes nulos*. También se cumple que en una cadena de Markov con un espacio de estados finitos todos los estados son recurrentes positivos. Finalmente, a los estados recurrentes positivos, aperiódicos se les llama ergódicos.

Dada una cadena de Markov con espacio de estados E y matriz de transición T, las distribuciones que tienen la propiedad de que si la posición inicial de la cadena X_0 tiene esa distribución entonces la distribución de cualquier variable X_n de la cadena coincide con la de X_0, reciben el nombre de distribuciones estacionarias o de equilibrio.

Sea $\{\pi_i, i \in E\}$ la distribución inicial de la cadena, esto es,

$$\pi = Pr\left[X_0 = i\right], \ \sum_{i \in E} \pi_i = 1.$$

Se busca que las condiciones para el resto de variables de la cadena X_n tengan la misma distribución.

$$Pr\left[X_n = i\right] = \pi_i \ \ \forall \ \ n.$$

Así por ejemplo, para $n = 1$ se tiene que verificar que

$$\begin{array}{l} \pi_i = Pr\left[X_1 = i\right] = \sum_{j \in E} Pr\left[X_1 = i, X_0 = j\right] \\ = \sum_{j \in E} Pr\left[X_0 = j\right] Pr\left[X_1 = i | X_0 = j\right] \\ = \sum_{j \in E} \pi_j P_{ji}, \quad i \in E. \end{array}$$

Una distribución $\pi = \{\pi_i, i \in E\}$ se dice que es estacionaria respecto a una cadena de Markov con matriz de transición P, si $\pi P = \pi$, es decir, para cualquier estado $i \in E$

$$\pi_i = \sum_{j \in E} \pi_j P_{ji}.$$

Una cadena de Markov es estacionaria si T es una matriz de transición de una cadena de Markov *irreducible* y *aperiódica*. Entonces existe una distribución estacionaria respecto a P si, y sólo si, la cadena es recurrente positiva, i.e. $\lim_{n \to \infty} P_{ji}^{(n)} = \pi_i > 0$. En tal caso, la distribución estacionaria es única y de la forma:

$$\pi_i = \frac{1}{E[N_{ii}]},$$

donde N_{ii} es el número de etapas en las que se regresa a i saliendo de i. Si los estados son transitorios o nulos, $\lim_{n \to \infty} P_{ji}^{(n)} = 0$, no hay distribución estacionaria. Por otra parte si una cadena irreducible finita, tiene un número finito de estados y todos se comunican, todos los estados son recurrentes positivos. Entonces, por el punto anterior, existe una única distribución estacionaria.

Apéndice B

Miscelánea de ejemplos

B.1. Ejemplo de Rangos

A continuación se describe uno de los ejemplos incluidos en WinBUGS, en el que se analizan los resultados de un grupo de escuelas de Londres a través de rangos[1]. El modelo fue presentado por Goldstein et al. (1993), en el cual usaron modelos jerárquicos para estudiar la variación entre las escuelas y calcularon el *nivel residual* para diferenciar si las escuelas son buenas o malas. En el ejemplo se muestra cómo calcular los rangos con respecto a las escuelas y obtener un intervalo de credibilidad para cada rango. Se obtuvo la media estandarizada de las calificaciones (Y) para 1978 alumnos de 38 diferentes escuelas. La mediana del número de alumnos por escuela fue 48, con un rango de 1 a 198. Para modelar el nivel de los alumnos usaron las variables: sexo, la puntuación de un examen de lectura, *London Reading Test* (LRT), y la calificación obtenida de un examen de habilidad verbal (VR). Cada escuela fue clasificada por género tomando en cuenta si la escuela era de niñas, niños o mixta y el tipo de escuela respecto a sus creencias: Iglesia de Inglaterra, Católica Romana, Escuela Estatal u otra.

El modelo que se presenta a continuación corresponde esencialmente al de Goldstein et al.

$$y_{ij} \sim Normal(\mu_{ij}, \tau_{ij})$$

[1]Este ejemplo no está completamente terminado en el manual de WinBUGS, ya que falta incluir el histograma al ejemplo. Por otra parte note que si intenta usar la sintaxis alpha[,k] para obtener los rangos, **Rank**→*node* no reconoce este vector, por lo que tendrá que crear otra variable para guardar este vector, en este caso se construyó un *loop* para `alphauno` que contiene los rangos de la escuela 1.

$$\mu_{ij} = \alpha_{1j} + \alpha_{2j}\text{LRT}_{ij} + \alpha_{3j}\text{VR1}_{ij} + \beta_1\text{LRT}_i^2 + \beta_2\text{VR2}_{ij} + \beta_3\text{Girl}_{ij}$$
$$+\beta_4\text{Girls'school}_j + \beta_5\text{Boys'school}_j + \beta_6\text{CEschool}_j + \beta_7\text{RCschool}_j$$
$$+\beta_6\text{CEschool}_j + \beta_7\text{RCschool}_j + \beta_8\text{other school}_j$$

$$\log \tau_{ij} = \theta + \phi\text{LRT}_{ij}$$

donde i se refiere al niño y j al índice de la escuela. Se especifica un modelo de regresión para la varianza de los componentes y un modelo logarítmico para la precisión τ_{ij} como una función lineal para cada puntuación de los alumnos. Las diferencias con respecto al modelo de Goldstein et al. es que estos autores permiten que la varianza $\sigma_{ij}^2 = 1/\tau_{ij}$ dependa linealmente de LRT. Sin embargo, la parametrización puede llevar a estimaciones negativas de σ_{ij}^2.

Distribuciones iniciales

Los efectos ajustados β_k $(k = 1, \ldots, 8)$, θ y ϕ tienen una distribución normal independiente con media cero y precisión=0.0001. Los coeficientes aleatorios del nivel de la escuela α_{kj} $(k = 1, 2, 3)$ siguen una distribución normal multivariada con media desconocida γ y matriz de covarianzas Σ. Se especifica una matriz normal multivariada no informativa inicial con media γ y una matriz de covarianzas $T = \Sigma^{-1}$ con distribución Wishart. Para representar la información inicial, se eligen los grados de libertad para esta distribución que sea lo más pequeña posible. La matriz R, fue especificada como:

$$R = \begin{bmatrix} 0{,}1 & 0{,}005 & 0{,}005 \\ 0{,}005 & 0{,}01 & 0{,}005 \\ 0{,}005 & 0{,}005 & 0{,}01 \end{bmatrix}$$

y representa la información inicial de la magnitud de Σ. La especificación del modelo en lenguaje BUGS es la siguiente:

```
model
{
  for(p in 1 : N) {
   Y[p] ~ dnorm(mu[p], tau[p])
   mu[p] <- alpha[school[p], 1]
     + alpha[school[p], 2] * LRT[p]
           + alpha[school[p], 3] * VR[p, 1]
           + beta[1] * LRT2[p]
           + beta[2] * VR[p, 2] + beta[3] * Gender[p]
           + beta[4] * School.gender[p, 1]
           + beta[5] * School.gender[p, 2]
```

```
          + beta[6] * School.denom[p, 1]
          + beta[7] * School.denom[p, 2]
          + beta[8] * School.denom[p, 3]
  log(tau[p]) <- theta + phi * LRT[p]
  sigma2[p] <- 1 /  tau[p]
  LRT2[p] <- LRT[p] * LRT[p]
 }

  for (j in 1 : M) {
alphauno[j] <- alpha[j, 1]
}
# el menor LRT puntaje = -34.6193
   min.var <- exp(-(theta + phi * (-34.6193)))

  # el mayor LRT puntaje = 37.3807
   max.var <- exp(-(theta + phi * (37.3807)))

   # Distribuciones iniciales para los efectos fijos:
   for (k in 1 : 8) {  beta[k] ~ dnorm(0.0, 0.0001)}
   theta ~ dnorm(0.0, 0.0001); phi ~ dnorm(0.0, 0.0001)

   # Distribuciones iniciales para coeficientes aleatorios:
   for (j in 1 : M) {
      alpha[j, 1:3 ] ~ dmnorm(gamma[1:3 ], T[1:3 ,1:3 ]);
 alpha1[j] <- alpha[j,1]

   }

   # Hiper-iniciales:
   gamma[1:3] ~ dmnorm(mn[1:3 ], prec[1:3 ,1:3 ]);
   T[1:3 ,1:3 ] ~ dwish(R[1:3 ,1:3 ], 3)
}
```

Al consultar la matriz de datos para la escuela, la matriz es de 1978×3. Se asigna 1 para los alumnos de la escuela 1, 2 para los alumos de la escuela 2 y así sucesivamente. Los variables Y, μ y τ son vectores con índices $p = 1, \ldots, 1978$ y se especifica el alumno i de la escuela j en las ecuaciones planteadas. El coeficiente del nivel de la escuela para el alumno p se selecciona usando el indicador en el renglón p del arreglo de datos, por ejemplo `alpha[school[p],1]`.

La ordenada al origen para la escuela j, α_{j1} mide el "efecto residual" después de ajustar las covariables referentes al alumno y escuela. Esto representa una cantidad

apropiada para representar los rangos de las escuelas. Como el rango es una función de nodos estocásticos, su valor cambia en cada iteración. Sin embargo, se puede obtener la distribución final para los rangos de **alpha[,k]** que se puede resumir con el siguiente histograma.

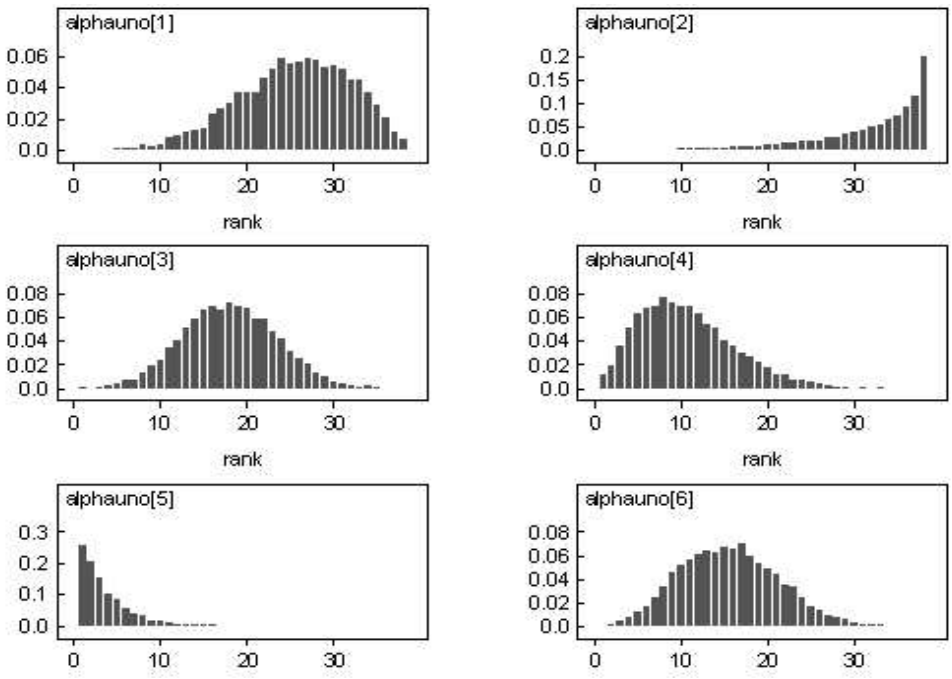


Figura B.1: *En esta figura sólo se muestran los rangos para las primeras seis escuelas*

◇

B.2. Especificar una nueva muestra de una distribución normal

En este código se muestra la construcción de la verosimilitud de una distribución normal a través del "truco de los ceros":

```
model {

  # uso del truco de los ceros
  for (i in 1:7) {
    zeros[i] <- 0

    # la verosimilitud se define como exp(-phi[i])
    zeros[i] ~ dpois(phi[i])

    # -log(verosimilitud)
    phi[i] <- log(sigma[1])+0.5*pow((x[i]-mu[1])/sigma[1],2)
  }

  #revisión a través de la distribución normal
  for (i in 1:7) {
    x[i] ~ dnorm(mu[2], prec)
  }
  prec <- 1 / (sigma[2] * sigma[2])

    for (k in 1:2) {
    mu[k] ~ dunif(-10, 10)
    sigma[k] ~ dunif(0, 10)
  }
}

Datos:
list(x = c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7))

Valores iniciales:
list(sigma = c(1, 1), mu = c(0, 0))
```

Los resultados muestran la similitud entre el método estándar y el truco de los ceros:

```
node     mean   sd     MC error  2.5%    median  97.5%  start sample
mu[1]    0.366  0.489  0.007150  -0.629  0.364   1.336  1001  5000
mu[2]    0.366  0.473  0.007003  -0.559  0.367   1.291  1001  5000
sigma[1] 1.191  0.490  0.014560  0.6293  1.075   2.476  1001  5000
sigma[2] 1.168  0.461  0.009039  0.6265  1.062   2.311  1001  5000
```

B.3. Modelar una observación futura o un dato faltante

Este modelo genera observaciones futuras o datos faltantes.

```
model {

   # uso del truco de los ceros
   for (i in 1:8) {
    zeros[i] <- 0

    # la verosimilitud es exp(-phi[i])
    zeros[i] ~ dpois(phi[i])

    # -log(verosimilitud)
    phi[i] <- log(sigma[1])+0.5*pow((x[i]-mu[1])/sigma[1],2)

    # distribución inicial uniforme impropia sobre cada x[i]
    x[i] ~ dflat()
  }

  # revisión a través de la distribución normal
  for (i in 1:8) {
    x.rep[i] ~ dnorm(mu[2], prec)
  }

  prec <- 1 / (sigma[2] * sigma[2])


  for (k in 1:2) {
    mu[k] ~ dunif(-10, 10)
    sigma[k] ~ dunif(0, 10)
  }
}

Datos:
list(x = c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7, NA),
x.rep=c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7, NA))
```

Note que en este caso se necesita replicar la base de datos para comparar los resultados.

```
Valores iniciales:
list(sigma = c(1, 1), mu = c(0, 0),
x = c(NA, NA, NA, NA, NA, NA, NA, 0))
```

Se necesita dar un valor inicial a los datos que faltan, de otra manera WinBUGS tratará de generar de la distribución inicial impropia y no podrá hacerlo.

El error MC en la predicción es grande y por lo tanto se necesita realizar mayor número de simulaciones.

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
mu[1]	0.368	0.494	0.003145	-0.615	0.3714	1.351	5001	50000
mu[2]	0.368	0.486	0.002319	-0.609	0.3713	1.326	5001	50000
sigma[1]	1.194	0.506	0.005429	0.6222	1.0750	2.460	5001	50000
sigma[2]	1.187	0.498	0.003917	0.6224	1.0690	2.450	5001	50000
x[8]	0.357	1.407	0.015110	-2.494	0.3608	3.133	5001	50000
x.rep[8]	0.358	1.381	0.006126	-2.372	0.3584	3.105	5001	50000

B.4. Especificar una nueva distribución inicial a través de una normal

El siguiente código utiliza el "truco de los ceros" para especificar una distribución inicial. Una observación Poisson cero, con media $phi = phi(mu)$, contribuye el término $\exp(-phi)$, que es la verosimilitud para mu; si ésta se combina con una distribución inicial "plana" para mu, se obtiene como resultado la función de densidad de mu que será proporcional a $\exp(-phi)$.

```
model {
  for (i in 1:7) {
    x.rep[i] <- x[i]  # réplica de los datos
    x[i] ~ dnorm(mu[1], prec[1])
    x.rep[i] ~ dnorm(mu[2], prec[2])
  }
mu[2] ~ dnorm(0, 1)   # distribución inicial conocida

# Uso del "truco de los ceros" para construir una
# normal(0, 1) inicial para mu[1]
zero <- 0

# La verosimilitud es exp(-phi)
```

```
zero ~ dpois(phi)

# -log(N(0, 1))
# phi[i] <- log(1)+0.5*pow((mu[1]-0)/1,2)
phi <- 0.5 * pow(mu[1], 2)

# distribución "plana" inicial
mu[1] ~ dflat()

for (k in 1:2) {
prec[k] <- 1 / (sigma[k] * sigma[k])
sigma[k] ~ dunif(0, 10)
}
}

Datos:
list(x = c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7))

Valores iniciales:
list(sigma = c(1, 1), mu = c(0, 0))
```

Observe que 10000 iteraciones no son suficientes; pues el error MC es grande para mu[1].

```
node     mean   sd      MC error  2.5%    median  97.5% start sample
mu[1]    0.321  0.4179  0.01316   -0.513  0.3275  1.163 5001  5000
mu[2]    0.307  0.4083  0.005944  -0.541  0.3134  1.100 5001  5000
sigma[1] 1.150  0.4463  0.009842  0.6216  1.0510  2.251 5001  5000
sigma[2] 1.170  0.4660  0.010280  0.6254  1.0580  2.389 5001  5000
```

Con 100000 iteraciones el error MC es pequeño.

```
node     mean   sd     MC error  2.5%    median  97.5%  start sample
mu[1]    0.313  0.408  0.003015  -0.523  0.3242  1.102  5001  100000
mu[2]    0.310  0.407  0.001173  -0.539  0.3188  1.103  5001  100000
sigma[1] 1.156  0.453  0.002268  0.6189  1.0530  2.286  5001  100000
sigma[2] 1.158  0.449  0.002164  0.6225  1.0560  2.308  5001  100000
```

Como se puede observar en la tabla, los resultados son similares al utilizar el "truco de los ceros" que al realizar la formulación estándar. Sin embargo, observe (Figura B.2) que la autocorrelación es relativamente alta.

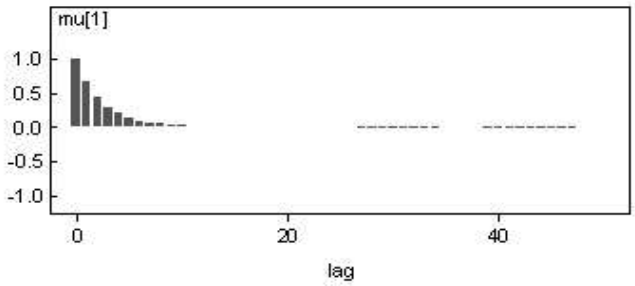


Figura B.2: *Gráfica de autocorrelación de mu[1]*

B.5. Ejemplo de punto de cambio

Carlin, Gelfand y Smith (1992) analizaron los datos de Bacon and Watts (1971) concernientes a un cambio de pendiente en un modelo de regresión lineal. Suponga que dos líneas rectas se intersectan en un punto x_k. Este modelo es ligeramente diferente al de Carlin, Gelfand y Smith, ya que ellos no ponen restricciones sobre el punto de cambio.

$$Y_i \sim Normal(\mu_i, \tau)$$
$$\mu_i = \alpha + \beta_{J[i]}(x_i - x_k) \quad J[i] = 1 \quad \text{si} \quad i \leq k \quad J[i] = 2 \quad \text{si} \quad i > k.$$

Con pendiente β_1 antes, y pendiente β_2 después del punto de cambio.

Dado que $E(Y) = \alpha$ en el de punto de cambio, entonces se encuentra muy correlacionada con k. Esta puede ser por tanto una parametrización muy pobre. El siguiente código muestra una parametrización continua para el punto de cambio.

```
model
{
   for(i in 1 : N) {
    Y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta[J[i]] * (x[i] - x.change)
```

A través de esta instrucción se identifica la pendiente para cada recta

```
        J[i] <- 1 + step(x[i] - x.change)
   }
```

Se asignan distribuciones iniciales no informativas a la precisión de las Y's, a la ordenada al origen y a las pendientes.

```
      tau ~ dgamma(0.001, 0.001)
      alpha ~ dnorm(0.0,1.0E-6)
   for(j in 1 : 2) {
      beta[j] ~ dnorm(0.0,1.0E-6)
   }
      sigma <- 1 / sqrt(tau)

   # distribución inicial del punto de cambio
      x.change ~ dunif(-1.3,1.1)
}
```

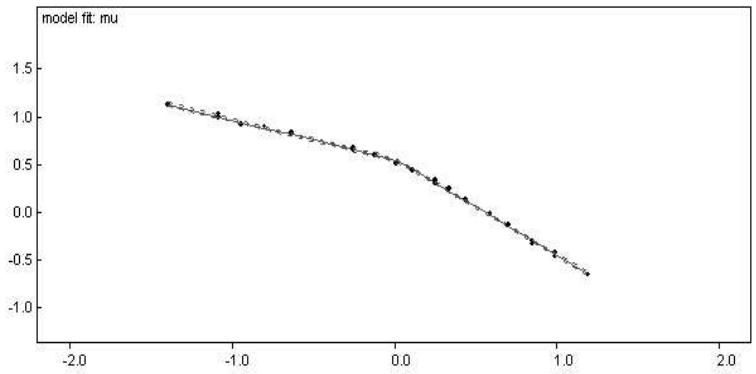


Figura B.3: *Gráfica de las rectas ajustadas a los datos*

Como puede observar en la Figura B.3 el ajuste es muy bueno, pero la correlación entre el punto de cambio (x.change) y *alpha* es grande, cercana a -1.

Apéndice C

Archivos del modelo de regresión

Modelo del archivo *Regre_model.txt*.

```
model;
{
   for( i in 1 : N ) {
      y[i] ~ dnorm(mu[i],tau)
   }
   for( i in 1 : N ) {
      mu[i] <- alpha + beta * x[i]
   }
   alpha ~ dnorm( 0.0,1.0E-6)
   beta ~ dnorm( 0.0,1.0E-6)
   tau ~ dgamma(0.001,0.001)
   sigma <- 1 / tau
}
```

Datos del modelo del archivo *Regre_data.txt*.

```
list(N=5,x = c(8.0, 15.0, 22.0, 29.0, 36.0),
y = c(151, 199, 246, 283, 320))
```

Contenido del archivo *Regre_init.txt* para los valores iniciales del modelo.

```
list(alpha=0,beta=0,tau=1)
```

Apéndice D

Aplicaciones

D.1. Resultados para el caso de Estados Unidos

En esta sección se presenta el modelo final 4.2 (sin término cuadrático), así como valores iniciales, datos y algunos resultados de convergencia.

Modelo

```
model{
   for( i in 1:13464) {
     turnout[i] ~ dbin(p[i],N[i])
     logit(p[i])<- y[i]
         y[i] <- beta[1]*raza[i] + beta[2]*edad[i] +
         beta[3]*educacion[i] + beta[4]*ingreso[i] + beta[5]
}
 for( i in 1:79) {
     logit(p.pm[i])<- p.ym[i]
         p.ym[i] <- beta[1]*raza.m + beta[2]*edad.m[i] +
         beta[3]*educacion.m + beta[4]*ingreso.m + beta[5]
 }
 for( i in 1:79) {
     logit(p.ph[i])<- p.yh[i]
        p.yh[i] <- beta[1]*raza.m + beta[2]*edad.m[i] +
        beta[3]*educacion.h + beta[4]*ingreso.h + beta[5]
 }
beta[1:5] ~dmnorm(mu[ ],B[ , ])
 }
```

Valores iniciales

```
list(b=c(0.12, 0.13, 0.11, 0.15, -5.4))
```

Datos

```
list(B = structure(.Data = c(0.001, 0, 0, 0, 0,
              0, 0.001, 0, 0, 0,
              0, 0, 0.001, 0, 0,
              0, 0, 0, 0.001, 0,
              0, 0, 0, 0, 0.001), .Dim = c(5,5) ) )

list(mu=c(0,0,0,0,0))
list(raza.m=1, educacion.m=10.22, ingreso.m=3.2044)
list(raza.h=1, educacion.h=15.38504, ingreso.h=5.29381)

raza[] educacion[] ingreso[] edad[] turnout[] N[]
   0       11      0.207055    17      0       1
   0       12      2.418       17      0       1
   0       10      2.907       17      0       1
   0       10      0.207055    18      0       1
   0       11      0.236367    18      0       1
   ...     ...     ...         ...     ...     ...
   1       12      0.630443    36      1       2
   1       12      0.691065    36      0       1
   1       14      0.691065    36      1       1
   ...      ...      ...        ...     ...     ...
   1        9      1.356       93      1       1
   1       11      2.278       93      1       1
   1       14      0.75942     95      1       1
   1        1      1.356       95      1       1
   1        0      1.313       99      1       1
END
```

Resultados

Las gráficas D.1 y D.2 corresponden al modelo 4.1, en la gráfica D.1 se observa que existe una autocorrelación muy alta para la edad, el término cuadrático y el término independiente, por otro lado la convergencia (gráfica D.2) para las mismas variables no muestra un buen comportamiento. Las gráficas D.3 y D.4 muestran los resultados de autocorrelación y convergencia para el modelo 4.2. En este caso se

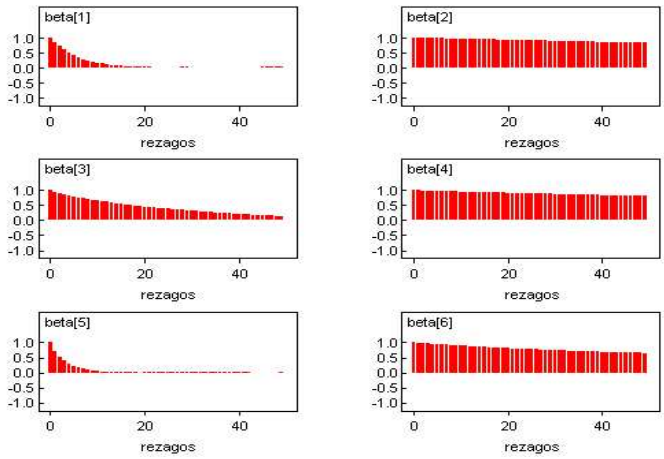


Figura D.1: *Autocorrelación*

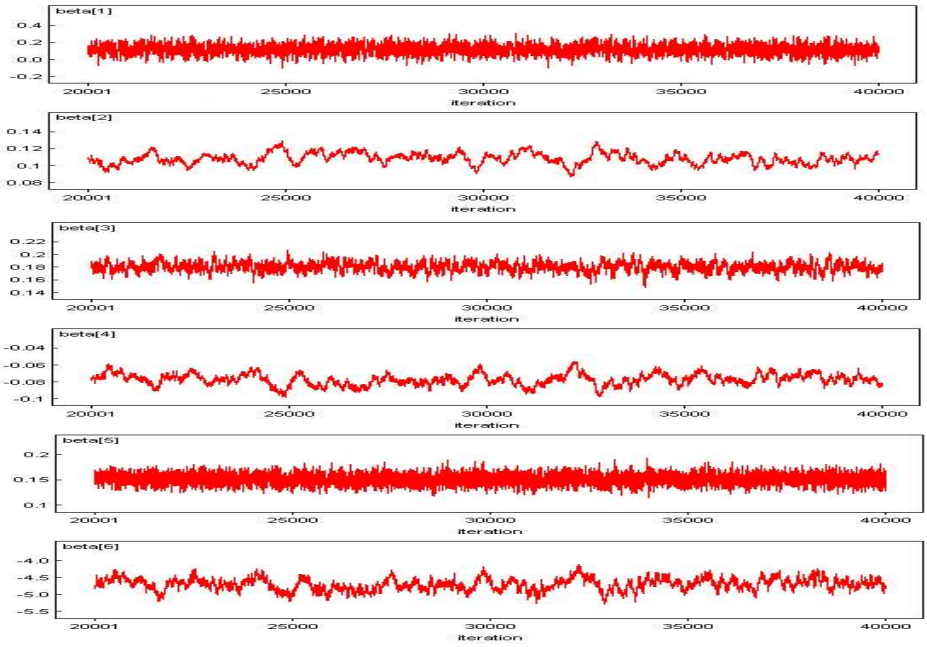


Figura D.2: *Convergencia*

simularon dos cadenas para verificar los resultados, y como se puede observar en las gráficas la autocorrelación y la convergencia muestran un buen comportamiento.

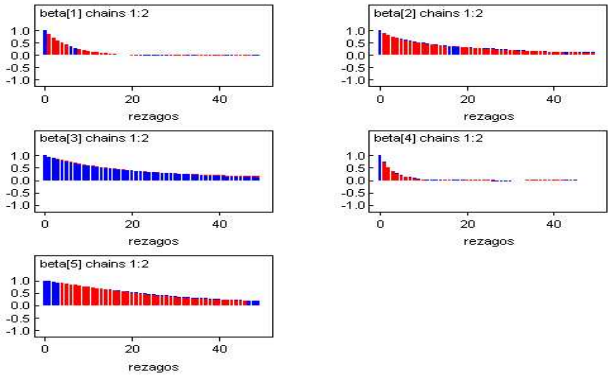


Figura D.3: *Autocorrelación*

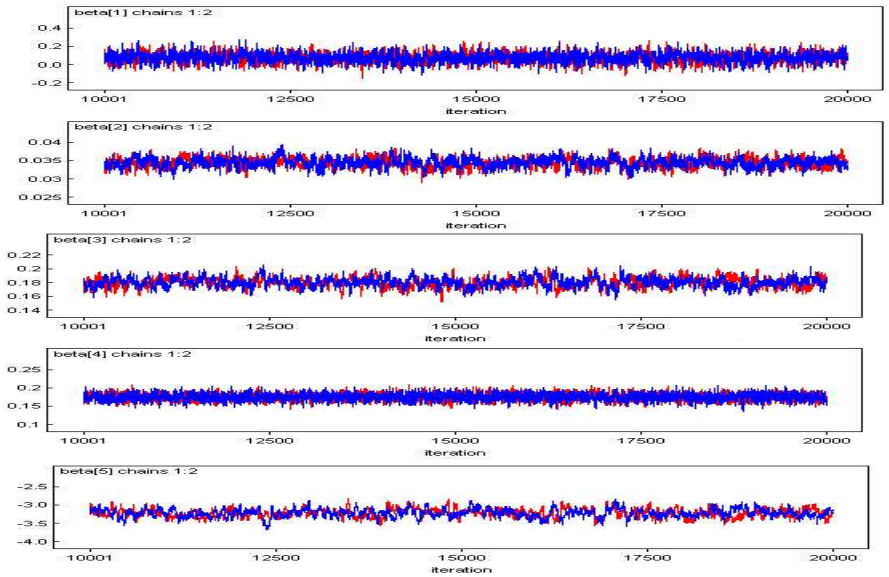


Figura D.4: *Convergencia*

D.2. Resultados para el caso de México

El modelo del caso de México es análogo al de Estados Unidos, salvo que en México no se considera a la raza.

Modelo

```
model
{
   for( i in 1:1013) {
   turnout[i] ~ dbin(p[i],N[i])
   logit(p[i])<- y[i]
       y[i] <- b[1] + b[2] *edad[i] +b[3]*educacion[i] +b[4]*ingreso[i]
}
 for( i in 1:71) {
   logit(p.pm[i])<- p.ym[i]
       p.ym[i] <-b[1]*edad.m[i] + b[2]*edu.m + b[3]*ingreso.m + b[4]
}

 for( i in 1:71) {
   logit(p.ph[i])<- p.yh[i]
       p.yh[i] <-b[1] *edad.m[i] + b[2]*edu.h + b[3]*ingreso.h + b[4]
  }

b[1:4] ~dmnorm(mu[ ],B[ , ])

}
```

Valores iniciales

```
list(b=c(0.02, 0.0864, -0.03, 0.1))
```

Datos

```
list(B = structure(.Data = c(0.001, 0, 0, 0,
              0, 0.001, 0, 0,
              0, 0, 0.001, 0,
              0, 0, 0, 0.001), .Dim = c(4,4) ) )

list(mu=c(0,0,0,0))
list(edu.m=3.2789, ingreso.m=2.611)
list(edu.h=6.66767, ingreso.h=3.8327)
```

```
educacion[] ingreso[] edad[] turnout[] N[]
   2            1        18       0      1
   3            1        18       0      1
  ...          ...       ...      ...    ...
   8            4        24       1      1
   8            5        24       1      1
   2            1        25       2      2
   2            2        25       1      3
   ...         ...       ...      ...    ...
   6            4        80       1      1
   2            3        81       1      1
   1            1        84       1      2
   2            3        84       1      1
   2            2        88       1      1
END
```

Resultados

Las gráficas D.5 y D.6 correspondientes al modelo 4.3 (modelo con término cuadrático), tiene un comportamiento muy similar a los resultados obtenidos para Estados Unidos, las variables que tienen problemas de autocorrelación y convergencia son: la edad, el término cuadrático y el término independiente.

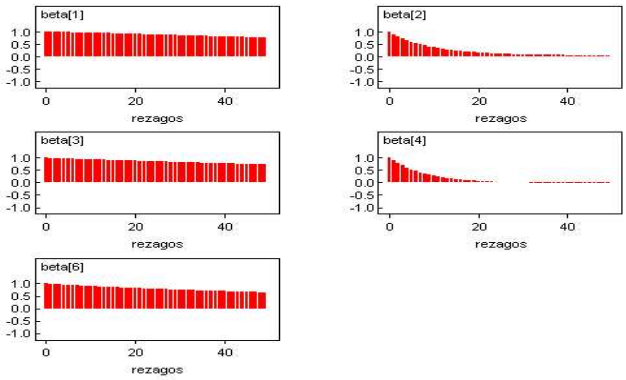


Figura D.5: *Autocorrelación*

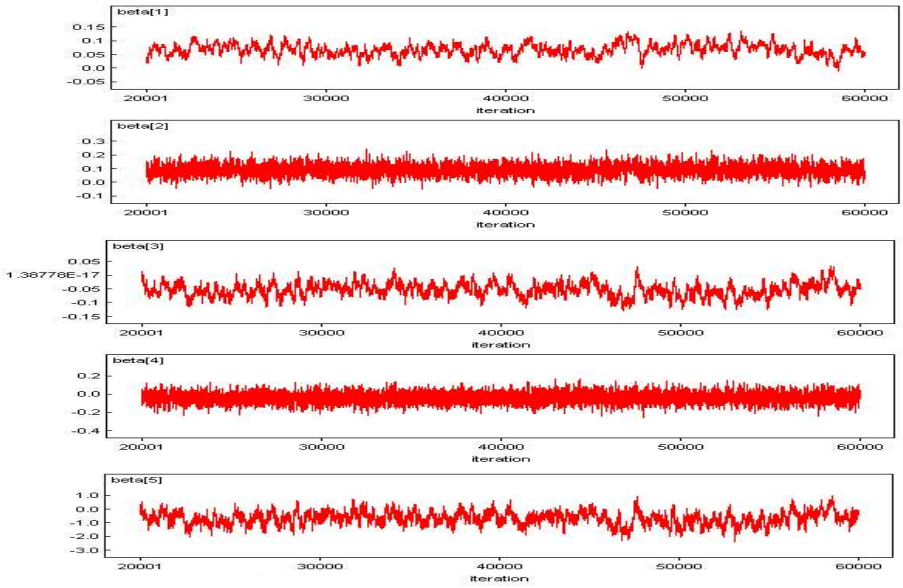


Figura D.6: *Convergencia*

Como se puede observar en las gráficas D.7 y D.8, al quitar el término cuadrático la convergencia y la autocorrelación mejoran de manera considerable.

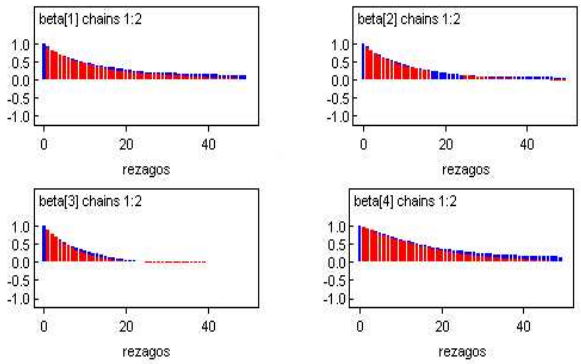


Figura D.7: *Autocorrelación*

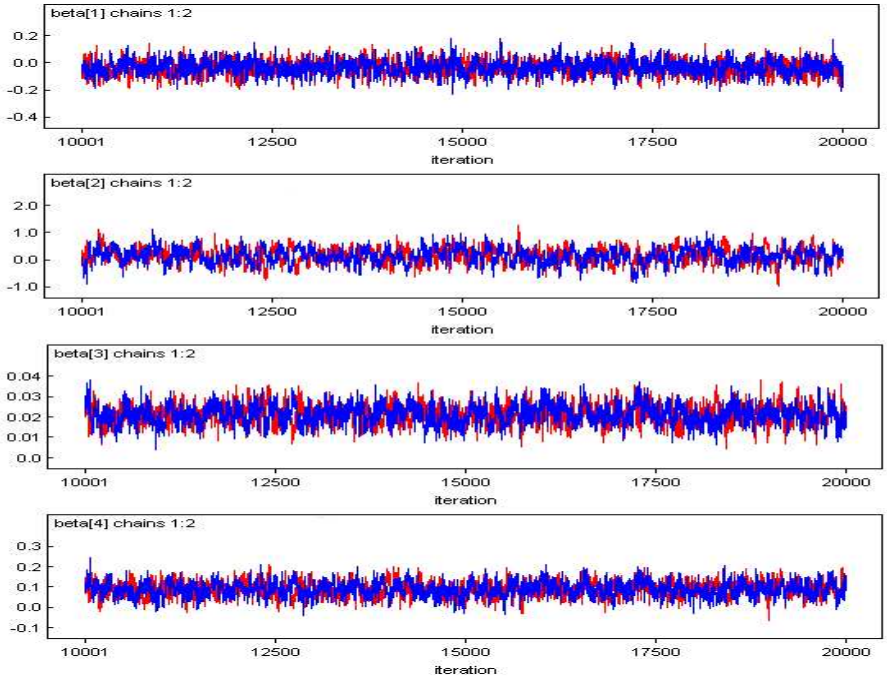


Figura D.8: *Convergencia*

Bibliografía

[1] Brooks, S. P. y Gelman, A. (1998). *Alternative methods for monitoring covergence of iterative simulations*. Journal of Computation and Graphical Statistics. 7, 434-445.

[2] Buendía, J. y Samuano, F. (2003). *Participación electoral en nuevas democracias: elección presidencial de 2000 en México.* **Vol. 10**, No. 2. México: ITAM y Colegio de México.

[3] Caballero, M. E., Rivero V. M., Uribe, G. y Velarde, C. (2004). *Cadenas de Markov. Un enfoque elemental*. México: Instituto de Matemáticas, UNAM.

[4] Congdon, P. (1998). *Applied Bayesian modelling*. London: John Wiley & Sons Ltd.

[5] Congdon, P. (2001). *Bayesian Statistical Modelling*. London: John Wiley & Sons Ltd.

[6] Congdon, P. (2005). *Bayesian Models for Categorical Data*. London: John Wiley & Sons Ltd.

[7] Coss Bu, R. (1998). *Simulación*. Limusa.

[8] Ehlers, R. S. (2003). *Métodos Computacionalmente Intesivos em Estatística.* Departamento de Estadística. Universidades Federal do Paraná.

[9] Erdely-Ruíz, A. (2004). Apuntes de Estadística Bayesiana. Facultad de Ciencias, UNAM.

[10] Gamerman, D. (1997). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. London: Chapman & Hall.

[11] Gelfand, A. E. y Smith, A. F. M. (1990). *Sampling-based approaches to calculating marginal densities. Journal of the American Statistical Association* 85, 398 - 409.

[12] Gelman, A. et. al. (2005). *R2WinBUGS: A Package for Running WinBUGS from R.* Journal of Statistical Software, **Vol.12**, Issue 3.

[13] Geman, S. y Geman, D. (1984). *Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, 721 - 741.

[14] Gilks, W. (1992) *Derivative-free adaptive rejection sampling for Gibbs sampling.* In *Bayesian Statistics 4*, (J M Bernardo, J O Berger, A P Dawid, and A F M Smith, eds), Oxford University Press, UK, Pp. 641-665.

[15] Goldstein, H., Rasbash, J., Yang, M., Woodhouse, G., Pan, H., Nuttall, D., and Thomas, S. (1993). *A Multilevel Analysis of School Examination Results.* Oxford Review of Education.

[16] Gómez-García, J. y Palarea-Albaladero, J. (2003). *Algoritmos Monte Carlo basados en Cadenas de Markov aplicados a la imputación múltiple de datos faltantes.* Departamento de Métodos Cuantitativos. Universidad de Murcia.Departamento de Informática de Sistemas. Universidad Católica San Antonio.

[17] Gutiérrez-Peña, E. (1997). *Métodos Computacionales en la Inferencia Bayesiana.* Series Monografías, **Vol.6**, No. 15. México: IIMAS, UNAM.

[18] Hastings, W. K. (1970). *Monte Carlo sampling methods using Markov chains and their applications.Biometrika* 57, 97 - 109.

[19] Kahn, S. (1956). *Use of Different Monte Carlo Sampling Techniques. Symposium on Monte Carlo Methods.* Nueva York: Wiley.

[20] King, G., Tomz, M. y Wittenberg (2000). *Making the Most of Statistical Analyses: Improving Interpretation and Presentation.* American Journal of Political Science, **Vol.44**, No. 2. Pp. 341-355.

[21] Lunn, D. J., Thomas, A., Best, N. y Spiegelhalter, D. (2000). *WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility.* Imperial College School of Medicine, London. Institute of Public Health, Robinson Way, Cambrige. No. 10, Pp. 325-337.

[22] McHaney, R. (1991). *Computer Simulation: A Practical Perspective.* John Wiley and Sons.

[23] Metropolis, N. et al. (1953). *Equations of state calculations by fast computing machine.* Journal of chemical physics 21, 1087 - 1091.

[24] N. Metropolis y S. Ulam (1949) The Monte Carlo method. Journal of the American Statistical Association, vol. 44, no. 247, Pp. 335-41.

[25] Neal, R. (1997) *Markov chain Monte Carlo methods based on 'slicing' the density function.* Technical Report 9722, Canada: Department of Statistics, University of Toronto.

[26] Neal, R. (1998) *Suppressing random walks in Markov chain Monte Carlo using ordered over-relaxation.* In *Learning in Graphical Models*, (M I Jordan, ed). Kluwer Academic Publishers, Dordrecht, Pp. 205-230.

[27] Robert, C. P. y Casella, G. (1999). *Monte Carlo Statistical Methods.* Springer.

[28] Ross, S. (1996). *Stochastic Processes.* United States: Wiley.

[29] Rosenstone, S. y Hansen J. (1993). *Mobilization, Participation and Democracy in America.* Nueva York: Macmillan.

[30] Sáez-Castillo, A. *Procesos aleatorios en tiempo discreto.* Departamento de Estadística e I. O. Universidad de Jaén.

[31] Spiegelhalter, D. J., Best N. G., Carlin, B. P., van der Linde, A. (2002). *Bayesian measures of model complexity and fit.* Medical Research Council Biostatistic Unit, Cambrige, UK, Imperial College School of Medicine, UK. University of Minnesota, USA, University of Bremen, Germany. No. 64,Part 4, Pp. 583-639.

[32] Verba, S. y Norman, H. N. (1972). *Participation in America.* Nueva York: Harper & Row.

[33] Zaller, J. (1992). *The Nature and Origins of Mass Opinion.* Nueva York: Cambrige University Press.

Electrónica:

[34] Journal of Statistical Software, 23 de mayo de 2006.
http://www.jstatsoft.org/

[35] Gary King, 2006.
http://gking.harvard.edu/

[36] Generalized Linear Latent And Mixed Models, mayo de 2006. http://www.gllamm.org/examples.htm

[37] Comparative Study of Electoral Systems.
http://www.cses.org

Software

- BUGS (The BUGS Proyect)

Disponible en
http://www.mrc-bsu.cam.ac.uk/bugs/

- R (The R Project for Statistical Computing)

Disponible en
http://www.r-project.org/

- BOA (Bayesian Output Analysis Program (BOA) for MCMC)

Disponible en
http://cran.r-project.org/src/contrib/Descriptions/boa.html

- R2WinBUGS (R2WinBUGS: Running WinBUGS and OpenBUGS from R)

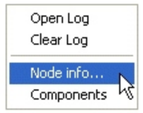
Disponible en
http://cran.r-project.org/src/contrib/Descriptions/R2WinBUGS.html

- OpenBUGS

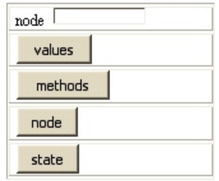
Disponible en
http://mathstat.helsinki.fi/openbugs/

Estructura de los principales menús necesarios para llevar a cabo un análisis bayesiano en WinBUGS

Menú *Info*

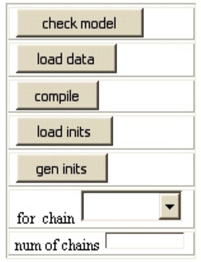


Node info...

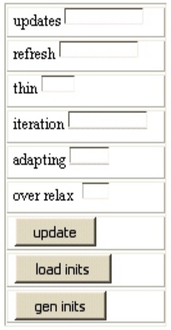


Menú *Model*

Specification...

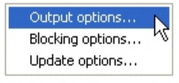


Update...



Seed...

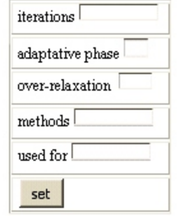
Menú *Options*



Output options...

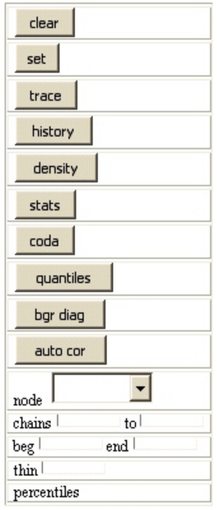
Blocking options...

Update options...

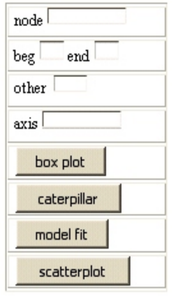


Menú *Inference*

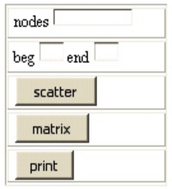
Samples...



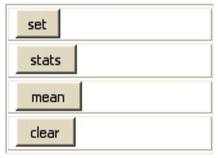
Compare...



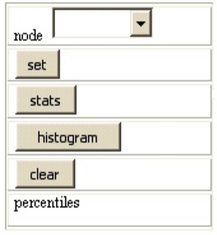
Correlations



Summary...

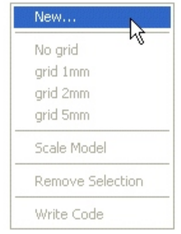


Rank...

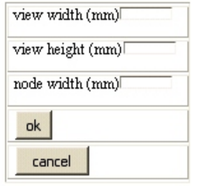


DIC...

Menú *Doodle*

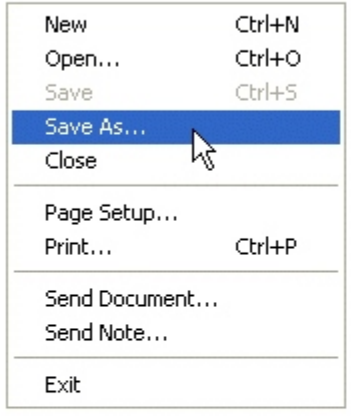


New...

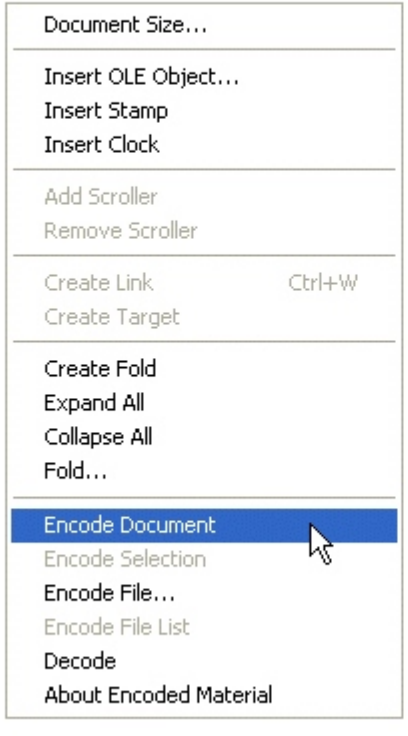


Menús para manipular los documentos de WinBUGS

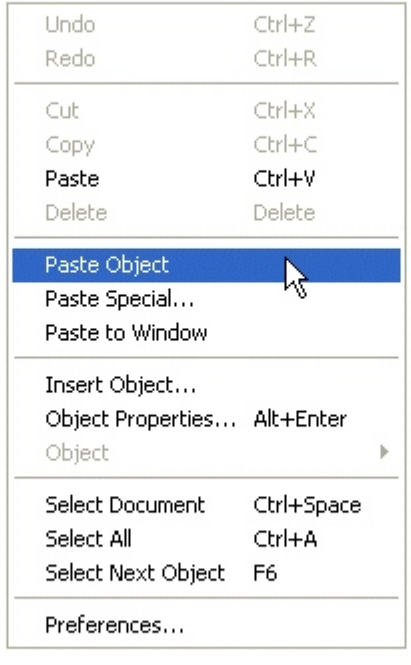
Menú *File*



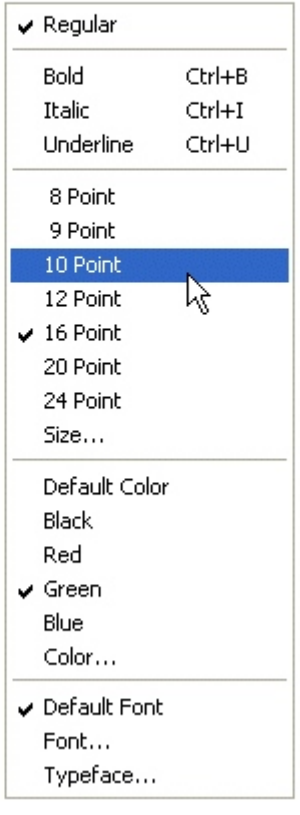
Menú *Tools*



Menú *Edit*



Menú *Attributes*



Menú *Text*