**UNIVERSIDADE FEDERAL DO PARANÁ**

# SEMINÁRIO
# WinBUGS

**Leonardo Melo / Ricardo Ehlers**

**CURITIBA**
**JULHO / 2006**

**WINBUGS – BAYESIAN INFERENCE USING GIBS SAMPLING FOR WINDOWS**

## Sumário:

Introdução. ....3
1. Definindo o modelo. ....3
2. Rodando as Iterações. ....5
3. Verificando a convergência. ....7
4. Exemplo: Air – Berkson measurement error ....8
5. Exemplo: Eyes – Normal Mixture Model ....10
6. Exemplo: Leuk – Cox Regression ....12
7. Exemplo: Ciências Atuariais ....15
8. Diagrama de Influência ....16
Comentários Finais ....**Erro! Indicador não definido.**

## Introdução.

O objetivo desta breve oficina é simplesmente divulgar e fornecer um conhecimento básico sobre o software "WinBUGS", que vem sido cada vez mais usado pela comunidade Estatística, dada sua facilidade de manuseio e a grande gama de problemas que ele consegue resolver, utilizando o Algoritmo de Gibbs.
Este material não tem o intuito de substituir o manual do usuário do software, que pode ser obtido no próprio menu do programa ou em http://www.mrc-bsu.cam.ac.uk/bugs/, mesma página de onde o download do programa pode ser efetuado.
Iremos começar mostrando o funcionamento do programa com exemplos mais simples e aumentando a complexidade gradativamente. Esperamos, com isso, mostrar um número diverso de exemplos nos quais o WinBUGS pode ser utilizado na solução de problemas.
É sempre interessante reiterar que este é apenas o passo inicial. O usuário sempre pode e deve procurar um aprendizado mais aprofundado, de acordo com suas necessidades.

## 1. Definindo o modelo.

O WinBUGS trabalha basicamente com três arquivos, sendo cada um deles contendo informações pertinentes ao problema em questão.
O arquivo geralmente nomeado como **"model"** é onde são especificados todos os parâmetros do modelo, bem como suas distribuições a priori, a verossimilhança, a hierarquia e as relações entre os parâmetros.
O segundo arquivo, **"data",** é onde especificamos os dados observados, que irão compor a verossimilhança.
E por, fim, um arquivo denominado **"initial"** especifica valores iniciais para os parâmetros de interesse.
Iremos utilizar um exemplo extraído do Vol. 01 da série de exemplos implementados no WinBUGS para ilustrar o seu funcionamento. O problema *"Pump: conjugate Gamma-Poisson hierarquical model"*) trata de modelos hierárquicos utilizados nas prioris conjugadas Gamma-Poisson.
George *et al* (1993) discute uma análise Bayesiana para modelos hierárquicos onde a priori conjugada é adotada no primeiro nível, mas para qualquer outra distribuição a priori dada para os hiperparâmetros, suas respectivas posterioris não serão fechadas por amostragem. Assim sendo, devemos obter uma posteriori aproximada para estes parâmetros. O WinBUGS sempre calcula uma distribuição aproximada por extrair uma amostra da densidade. O exemplo abaixo considera 10 bombas de usina de força, onde o número de falhas $x_i$ tem distribuição de Poisson.

$$X_i \mid \theta_i \sim \text{Poisson}(\theta_i * t_i) \qquad i = 1,...,10$$

onde $\theta_i$ é a taxa de falha por bomba *i* e $t_i$ é o tempo de operação da bomba (em 1000 segundos de hora). Estas informações são resumidas abaixo:

| Pump | $t_i$ | $x_i$ |
|---|---|---|
| 1 | 94.5 | 5 |
| 2 | 15.7 | 1 |
| 3 | 62.9 | 5 |
| 4 | 126 | 14 |
| 5 | 5.24 | 3 |
| 6 | 31.4 | 19 |
| 7 | 1.05 | 1 |
| 8 | 1.05 | 1 |
| 9 | 2.1 | 4 |
| 10 | 10.5 | 22 |

Deseja-se estimar a taxa média de falhas para cada bomba. Assim, como a conjugada da Poisson é a Distribuição Gamma, assumiu-se uma Distribuição Gamma para $\theta_i$, ou seja:

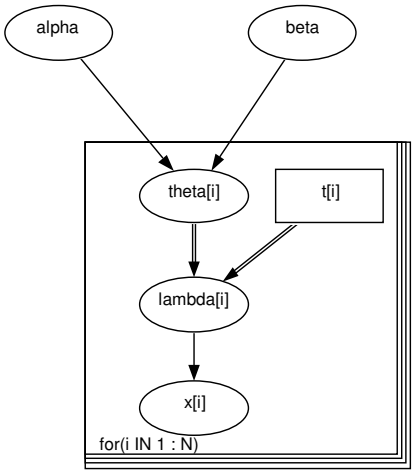
$\theta_i \sim \text{Gamma}(\alpha, \beta), \; i = 1,...,10$

Como trata-se de um modelo hierárquico, devemos estabelecer prioris também para os hiperparâmetros. Então, no segundo nível da hierarquia, assumiu-se que:

$\alpha \sim \text{Exponential}(1.0)$
$\beta \sim \text{Gamma}(0.1, 1.0)$

Desta forma, teríamos uma posteriori Gamma para $\beta$, mas não conhecemos uma posteriori padrão para $\alpha$. Conseqüentemente, usamos o Amostrador de Gibbs para simular uma densidade a posteriori aproximada para os hiperparâmetros, conseguindo, assim, uma posteriori (também aproximada) para $\theta_i \mid X_i$.

**Figura 1 –** Exemplo das Bombas para o Amostrador de Gibbs




**Fonte:** WinBUGS.

A partir desse gráfico de influência, podemos entender todo o funcionamento do modelo. Cada componente acima é chamado de "nó" (node). Há os nós chamados de "constantes", que são envolvidos por um retângulo. Este tipo de nó é especificado no arquivo de "data".
E há os nós "estocásticos", envolvidos por círculos. Note que eles relacionam-se entre si através de dois tipos de setas:

- Simples: indicam uma relação hierárquica. Significa que o nó de origem é um parâmetro do nó de destino. No nosso exemplo, "alpha" e "beta" são parâmetros da distribuição de "theta".
- Duplas: indicam uma relação direta. O nó de origem pode ser uma das variáveis regressoras de um modelo para explicar a variável de destino, por exemplo. Mas no caso em questão, note que lambda = theta*t.

Note também que os nós que variam segundo um mesmo índice estão sobre um mesmo retângulo (que mais tarde darão origem aos "loops"), cuja representação é descrita no rodapé do mesmo. No nosso

exemplo, seria "for (i in 1 : N)". Em exemplos posteriores, com mais de um índice, esta representação ficará mais clara.

Para escrever o modelo no arquivo "model" do WinBUGS, geralmente segue-se uma ordem, indo dos "loops" com índices de maior dimensão para os de menor dimensão. Descrevemos as distribuições ou relações determinísticas dos nós que dependam do mesmo índice em um único "loop". Em seguida, descrevemos os que dependem do segundo índice em outro "loop" e assim por diante, até chegar nos hiperparâmetros finais.
Para o exemplo das bombas, teremos:

**Model:**

```
model
{
        for (i in 1 : N) {
                theta[i] ~ dgamma(alpha, beta)
                lambda[i] <- theta[i] * t[i]
                x[i] ~ dpois(lambda[i])
        }
        alpha ~ dexp(1)
        beta ~ dgamma(0.1, 1.0)
}
```

Especificado o modelo, o segundo passo é descrever o arquivo de dados. Geralmente, faz-se uso de "listas", conforme segue:

**Data:**

```
list(t = c(94.3, 15.7, 62.9, 126, 5.24, 31.4, 1.05, 1.05, 2.1, 10.5),
   x = c(    5,     1,     5,   14,     3,    19,     1,     1,    4,    22), N = 10)
```

Note que, além de fornecer os valores dos dados observados, estamos fornecendo também o tamanho da amostra (N=10).
Por fim, especificamos os valores iniciais dos parâmetros de interesse no terceiro arquivo, conforme segue:

**Initial:** list(alpha = 1, beta = 1)

## 2. Rodando as Iterações.

Siga os seguintes passos:

**1)** Clique em **Model -> Specification**.
**2)** Clique no arquivo de "model" e depois, no botão *"check model"*. O WinBUGS verificará a sintaxe do seu modelo.
**3)** Clique no arquivo de "data" e depois, nos botões *"load data"* e *"compile"*. Quando você faz isso, além de verificar as consistências entre esses dois primeiros arquivos, o WinBUGS automaticamente gera o nó "deviance", que pode ser monitorado junto com os demais nós.
**Obs:** caso seja desejo do usuário utilizar mais de uma cadeia nas iterações, é preciso especificar o número de cadeiras requerido no campo *"num of chains"* antes de clicar em *"compile"*.
**4)** Se não há nenhum erro na especificação do seu modelo e do arquivo de dados, é hora de gerar os valores iniciais. Clique no arquivo de *"initial"* e depois, nos botões *"load inits"*. Com isso, ele carrega os valores especificados. Clique então em *"gen inits"* para que ele possa gerar os valores iniciais para aqueles que não foram especificados.

**Obs:** No caso de mais de uma cadeia, deve haver um arquivo de valores iniciais para cada cadeia. Assim sendo, clique no primeiro arquivo, depois em "load inits", depois no segundo arquivo, em "load inits" novamente e assim por diante, até o último arquivo. Só então clique em "gen inits" para inicializar o modelo.

**5)** Com o modelo inicializado, vá em **Model -> Update**. Escolha o número de iterações em *"updates"* e de quantas em quantas iterações ele deverá salvar os resultados em *"thin"*. A opção *"refresh"* diz simplesmente de quantas em quantas iterações a janela deve ser atualizada para mostrar ao usuário o número da iteração que estará rodando no momento. Clique em "update" e espere rodar as iterações. Essa "rodagem" inicial é o que chamamos de "burn-in", processo pelo qual a cadeia é "aquecida" antes de serem feitas as iterações que realmente interessam.

**6)** Depois de feito o "burn-in", clique em **Inference -> Samples**. Em "node", digite a(s) variável(s) a ser(em) monitorada(s). Você terá de clicar em "set" para selecionar cada uma delas.

**7)** Feito o monitoramento das variáveis de interesse, volte na ferramenta "Update" e rode as novas iterações.

**8)** Depois de rodadas todas as iterações, volte na "Sample Monitor Toll" e digite "*" novamente em "node". Ao fazer isso, todos os botões poderão ser clicados. Clique em "stats" para gerar os resultados e se quiser, salve-os como *.txt.

Outras opções da "Sample Monitor Tool" incluem:

- *"chains"*: seleciona quais cadeias (se tiver sido utilizada mais de uma) serão utilizadas para gerar os resultados.
- *"beg"* e "*end"*: pode ser utilizada uma amostra das iterações para gerar os resultados.
- *"thin"*: de quantas em quantas iterações o WinBUGS deve utilizar para gerar os resultados (note que este *thin* é diferente do *thin* da ferramenta "Update").
- *"percentiles"*: seleciona os percentis a serem utilizados para construção dos intervalos de credibilidade.

Claro que, depois de feito todo esse processo, é necessário verificar a convergência, mas trataremos desse assunto mais à frente.

Para o nosso exemplo das bombas, poderíamos fazer um aquecimento de 1000 iterações, seguido de um processo de 10.000, obtendo então os seguintes resultados:

| | mean | sd | MC_error | val2.5pc | median | val97.5pc | start | sample |
|---|---|---|---|---|---|---|---|---|
| alpha | 0.6867 | 0.265 | 0.004712 | 0.2886 | 0.6519 | 1.297 | 1001 | 10000 |
| beta | 0.9024 | 0.529 | 0.009881 | 0.1863 | 0.8023 | 2.201 | 1001 | 10000 |
| theta[1] | 0.05986 | 0.02488 | 2.26E-4 | 0.02134 | 0.0565 | 0.1176 | 1001 | 10000 |
| theta[2] | 0.1015 | 0.07971 | 8.03E-4 | 0.0085 | 0.0817 | 0.3045 | 1001 | 10000 |
| theta[3] | 0.08899 | 0.0375 | 3.893E-4 | 0.03056 | 0.0836 | 0.1774 | 1001 | 10000 |
| theta[4] | 0.1156 | 0.03018 | 3.267E-4 | 0.06408 | 0.1134 | 0.1808 | 1001 | 10000 |
| theta[5] | 0.6043 | 0.3181 | 0.003656 | 0.1478 | 0.5518 | 1.348 | 1001 | 10000 |
| theta[6] | 0.6121 | 0.14 | 0.001406 | 0.3686 | 0.6009 | 0.9133 | 1001 | 10000 |
| theta[7] | 0.899 | 0.7265 | 0.00671 | 0.08038 | 0.7092 | 2.758 | 1001 | 10000 |
| theta[8] | 0.9095 | 0.7333 | 0.008292 | 0.07935 | 0.7213 | 2.803 | 1001 | 10000 |
| theta[9] | 1.587 | 0.7769 | 0.008188 | 0.4597 | 1.454 | 3.461 | 1001 | 10000 |
| theta[10] | 1.995 | 0.427 | 0.004109 | 1.254 | 1.965 | 2.913 | 1001 | 10000 |

Não é difícil concluir que, apesar de não conhecermos a distribuição a posteriori de $\theta_i \mid x_i$, conseguimos as esperanças para os parâmetros de interesse. Assim, sabemos que $\alpha$ e $\beta$ têm, em média, os valores de 0,6867 e 0,9024, respectivamente. Além disso, obtivemos a média de taxa de falhas para cada uma das 10 bombas. . É interessante notar que, como se trata das médias das distribuições para cada parâmetro, temos estimadores de bayes por perda quadrática. Caso desejássemos um estimador por perda absoluta, bastaria utilizarmos a mediana (median).

O WinBUGS também fornece o desvio-padrão da estimativa (sd) e o erro para a média estimada via MCMC (MC error). Se os valores da média forem não correlacionados, este erro será $\sigma / \sqrt{n}$. Mas, por definição da Cadeia de Markov, temos que estes valores **serão** correlacionados. Para contornar este problema, Roberts (1996, pág. 50) propõe que sejam usados lotes (batchs) de valores simulados e sejam calculadas as médias desses lotes. Estas médias serão aproximadamente não correlacionadas e sendo n o número de lotes, pode-se obter uma estimativa para o desvio-padrão e conseqüentemente, para o erro.
Os valores "val2.5pc" e "val97.5pc" nada mais são do que os limites inferiores e superiores para o Intervalo de Credibilidade de 95% de confiança, respectivamente.
A coluna "start" nos diz simplesmente qual o número da iteração inicial daqueles resultados, enquanto "sample" é o tamanho da amostra (número de iterações produzidas).
Agora, suponha que quiséssemos encontrar a distribuição preditiva de x. Para isso, basta replicar a função de verossimilhança, acrescentando estas linha no modelo:

```
xrep[i] ~ dpois(lambda[i])}
```

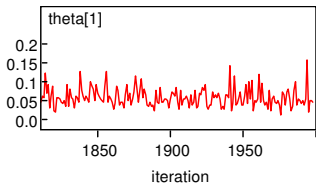
E em seguida, monitorar a variável "xrep". Esse procedimento é feito porque o WinBUGS não permite que se monitore diretamente a variável "x", já que isso obviamente não faz sentido, uma vez que "x" são os dados observados. Fazendo esse monitoramento e rodando novas iterações, obtemos os seguintes resultados:

| | mean | sd | MC_error | val2.5pc | median | val97.5pc | start | sample |
|---|---|---|---|---|---|---|---|---|
| xrep[1] | 5.663 | 3.382 | 0.03245 | 1.0 | 5.0 | 14.0 | 1 | 11000 |
| xrep[2] | 1.602 | 1.765 | 0.01565 | 0.0 | 1.0 | 6.0 | 1 | 11000 |
| xrep[3] | 5.655 | 3.396 | 0.03247 | 1.0 | 5.0 | 14.0 | 1 | 11000 |
| xrep[4] | 14.57 | 5.396 | 0.05116 | 5.0 | 14.0 | 26.0 | 1 | 11000 |
| xrep[5] | 3.162 | 2.431 | 0.02436 | 0.0 | 3.0 | 9.0 | 1 | 11000 |
| xrep[6] | 19.15 | 6.185 | 0.06288 | 8.0 | 19.0 | 33.0 | 1 | 11000 |
| xrep[7] | 0.9308 | 1.226 | 0.01107 | 0.0 | 1.0 | 4.0 | 1 | 11000 |
| xrep[8] | 0.9447 | 1.233 | 0.01207 | 0.0 | 1.0 | 4.0 | 1 | 11000 |
| xrep[9] | 3.311 | 2.471 | 0.02678 | 0.0 | 3.0 | 9.0 | 1 | 11000 |
| xrep[10] | 20.87 | 6.41 | 0.05768 | 10.0 | 20.0 | 35.0 | 1 | 11000 |

A interpretação, para este caso (usando a mediana como estimador), é que para a bomba 1, o número de falhas esperado é 5. Para a bomba 2, o número esperado seria 1, e assim por diante.

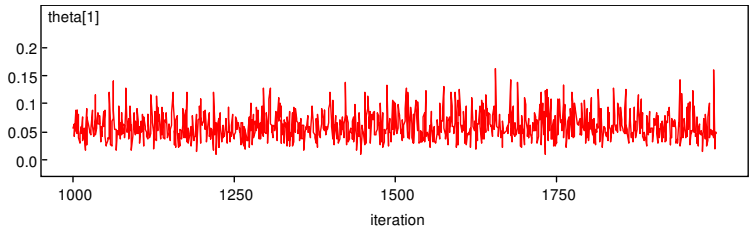
## 3. Verificando a convergência.

Ainda na "Sample Monitoring Tool", existem várias ferramentas que podem ser utilizadas para verificar se a cadeia convergiu ou não para a distribuição de interesse.
A ferramenta *"trace"* constrói um gráfico entre o valor da variável e o número de iterações. Este gráfico deve ser aleatório, isto é, não deve ficar "preso" em nenhuma área ao longo das iterações.

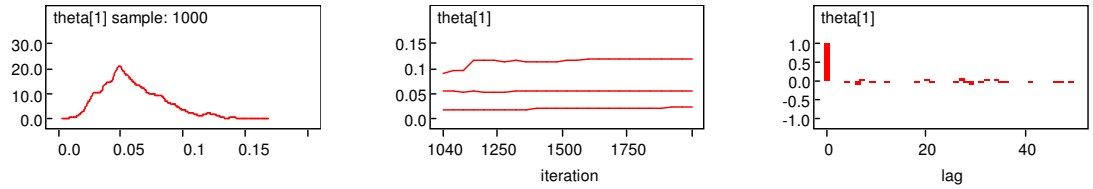



Caso isso venha a ocorrer, uma possível solução seria a de aumentar o número de iterações.

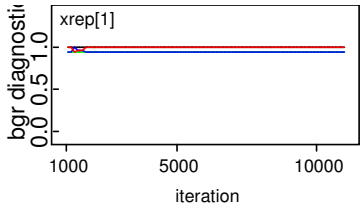
Outro gráfico com a mesma idéia é o *"history"*:




Novamente, não deve haver nenhum "vício" nesse gráfico. Se há um número muito concentrado de iterações em torno de algum valor, pode haver algum problema com a especificação do modelo ou mesmo dos valores iniciais.
E *"density"* retorna a densidade (aproximada, naturalmente) de cada uma das variáveis monitoradas.




A ferramenta *"quantiles"* nos retorna os gráficos das variáveis juntamente com seus respectivos intervalos de confiança. Espera-se, claro, que estes intervalos não sejam muito amplos.
O gráfico *"auto cor"* nos fornece o gráfico de autocorrelação, onde deve haver um "decaimento" o mais imediato possível nas barras desse gráfico, já que, apesar de, por definição da cadeia de markov termos valores correlacionados, espera-se que essa correlação seja a mais baixa possível, a fim de se obter estimativas mais confiáveis para os parâmetros em questão.
Por fim, a ferramenta *"bgr"* retorna o Diagrama de Gelman-Rubin.




Linhas próximas de 1 indicam convergência.

**Obs 1:** "Coda" retorna simplesmente os valores amostrados das variáveis monitoradas, sendo utilizados no pacote de diagnóstico do S-Plus chamado "CODA". Também pode ser implementado no R.
**Obs 2:** os valores desses gráficos podem ser visualizados com um duplo-clique do mouse sobre o gráfico, seguido de um clique com o botão esquerdo mantendo pressionada a tecla *ctrl.*

## 4. Exemplo: Air – Berkson measurement error

No volume 02 de exemplos do WinBUGS, encontramos um caso que utiliza regressão logística no estudo de doenças respiratórias relacionadas à exposição ao Dióxido de Nitrogênio ($NO_2$) em 103 crianças.

| Respiratory illness (y) | Bedroom $NO_2$ level in ppb (z) <20 | 20--40 | 40+ | Total |
|---|---|---|---|---|
| Yes | 21 | 20 | 15 | 56 |
| No | 27 | 14 | 6 | 47 |
| Total | 48 | 34 | 21 | 103 |

Assume-se a variável $z_j$ $(j = 1,2,3)$ para representar a concentração de $(NO_2)$ no quarto das crianças, dividida em 3 categorias. A natureza da relação do erro de medida associado com esta covariável é conhecida por:

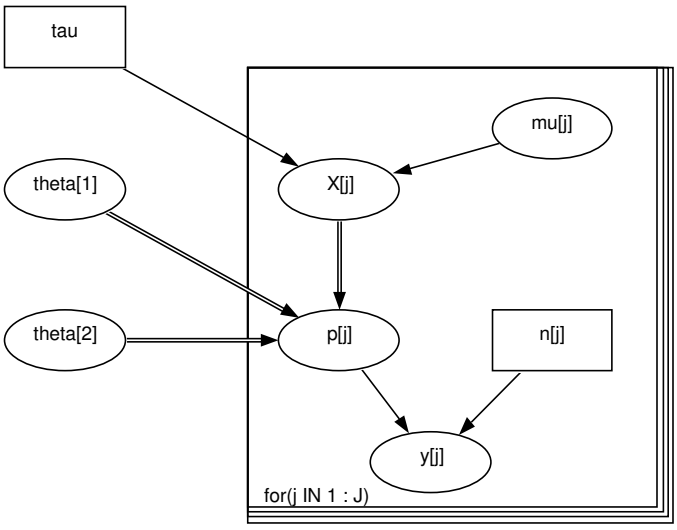
$$x_j = \alpha + \beta z_j + \varepsilon_j$$

Onde $\alpha = 4.48$, $\beta = 0.76$ e $\varepsilon_j$ é o elemento aleatório com Distribuição Normal de média zero e variância $\sigma^2$ $(= 1/\tau) = 81.14$. Neste exemplo, a covariável z assume os valores 10, 30 e 50 (os pontos médios de cada categoria) e x é interpretado como o "verdadeiro valor" da média de $(NO_2)$ no grupo j. A variável resposta é binária, refletindo a presença / ausência de doença respiratória. Assim, um modelo de regressão logística é assumido:

$$y_j \sim \text{Binomial}(p_j, n_j)$$
$$\text{logit}(p_j) = \theta_1 + \theta_2 x_j$$

Onde $p_j$ é a probabilidade de doença respiratória para crianças no j-ésimo grupo. Para os coeficientes $\theta_1$ e $\theta_2$ são assumidas prioris vagas e independentes, com Distribuição Normal.

**Figura 2 –** Exemplo "Measurement Error"




**Fonte:** WinBUGS

**Model:**

```
model
        {
                for(j in 1 : J) {
                        y[j] ~ dbin(p[j], n[j])
                        logit(p[j]) <- theta[1] + theta[2] * X[j]
                        X[j] ~ dnorm(mu[j], tau)
                        mu[j] <- alpha + beta * Z[j]
                }
                theta[1] ~ dnorm(0.0, 0.001)
                theta[2] ~ dnorm(0.0, 0.001)
        }
```

**Data:**

```
list(J = 3, y = c(21, 20, 15), n = c(48, 34, 21), Z = c(10, 30, 50), tau = 0.01234, alpha = 4.48, beta = 0.76)
```

**Initial:**

```
list(theta = c(0.0, 0.0), X = c(0.0, 0.0, 0.0))
```

Novamente foram geradas 1.000 amostras de aquecimento, seguidas de outras 10.000:

| | mean | sd | MC_error | val2.5pc | median | val97.5pc | start | sample |
|---|---|---|---|---|---|---|---|---|
| X[1] | 12.92 | 7.877 | 0.4227 | -3.775 | 13.3 | 26.96 | 1001 | 10000 |
| X[2] | 27.21 | 7.473 | 0.1946 | 13.05 | 27.01 | 42.63 | 1001 | 10000 |
| X[3] | 40.85 | 8.721 | 0.3502 | 24.18 | 40.84 | 58.37 | 1001 | 10000 |
| theta[1] | -0.9628 | 1.0 | 0.08808 | -4.233 | -0.7183 | 0.2104 | 1001 | 10000 |
| theta[2] | 0.04927 | 0.04276 | 0.003759 | 0.004071 | 0.03857 | 0.1951 | 1001 | 10000 |

## 5. Exemplo: Eyes – Normal Mixture Model

Também no volume 02 de exemplos, vemos de que modo o software trabalha com "misturas" de distribuições.

Bowmaker *et al* (1985) analisa dados de pico de ondas de sensitividade para gravações individuais microespectrofotométricas nos olhos de macacos. Os dados para cada macaco são dados abaixo:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 29.0 | 30.0 | 32.0 | 33.1 | 33.4 | 33.6 | 33.7 | 34.1 | 34.8 | 35.3 |
| 35.4 | 35.9 | 36.1 | 36.3 | 36.4 | 36.6 | 37.0 | 37.4 | 37.5 | 38.3 |
| 38.5 | 38.6 | 39.4 | 39.6 | 40.4 | 40.8 | 42.0 | 42.8 | 43.0 | 43.5 |
| 43.8 | 43.9 | 45.3 | 46.2 | 48.8 | 48.7 | 48.9 | 49.0 | 49.4 | 49.9 |
| 50.6 | 51.2 | 51.4 | 51.5 | 51.6 | 52.8 | 52.9 | 53.2 | | |

Parte da análise envolve uma mistura adequada de duas distribuições Normais com variância comum, ou seja, cada observação $y_i$ é assumida como pertencendo a um de dois grupos. $T_i = 1, 2$ denota o grupo da *i-ésima* observação, onde o grupo *j* tem distribuição Normal com média $\lambda_j$ e precisão $\tau$. Assumimos como desconhecida a fração P de observações que estão no grupo 2 e 1-P, no grupo 1. O modelo é dado conforme segue:

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

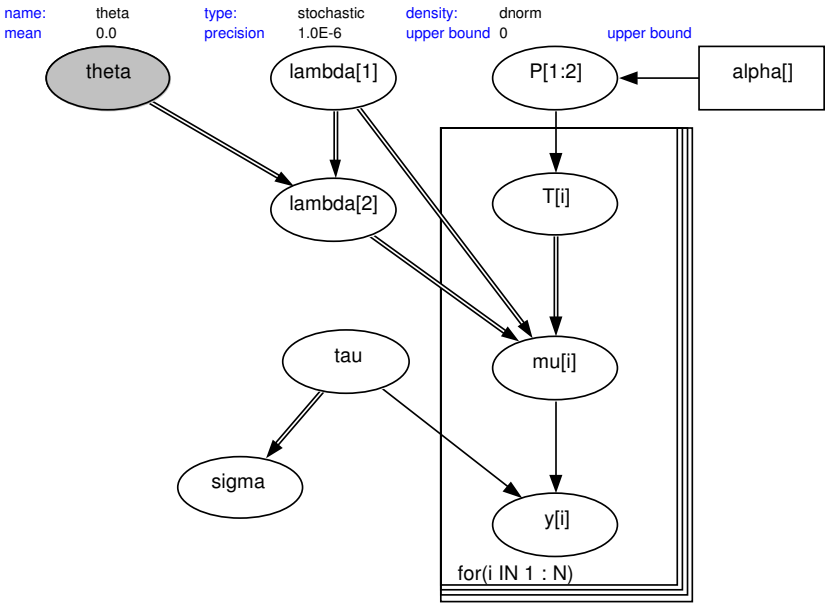
$$T_i \sim \text{Categorical}(P).$$

Note que poderíamos escrever $y_i = p*N(\lambda_1, \tau) + (1-p)* N(\lambda_2, \tau)$. Contudo, o WinBUGS utiliza esta forma por ser de fácil generalização.
Robert (1994) alerta para o fato de que, quando usamos este modelo, é perigoso que, em alguma iteração, os dados fiquem "presos" em um único componente da mistura. Para tentar contornar esse problema, Robert sugere uma re-parametrisação, assumindo que:

$$\lambda_2 = \lambda_1 + \theta, \quad \theta > 0.$$

Para $\lambda_1$, $\theta$, $\tau$, $P$, são dadas prioris independentes e não-informativas, incluindo uma priori uniforme para P em (0,1).

**Figura 3 –** Exemplo "Normal Mixture Model"




**Fonte:** WinBUGS

**Modelo:**

```
model
{
    for( i in 1 : N ) {
            y[i] ~ dnorm(mu[i], tau)
            mu[i] <- lambda[T[i]]
            T[i] ~ dcat(P[])
    }
    P[1:2] ~ ddirch(alpha[])
    theta ~ dnorm(0.0, 1.0E-6)I(0.0, )
    lambda[2] <- lambda[1] + theta
    lambda[1] ~ dnorm(0.0, 1.0E-6)
    tau ~ dgamma(0.001, 0.001)
    sigma <- 1 / sqrt(tau)
}
```

**Data:**

```
list(y = c(529.0, 530.0, 532.0, 533.1, 533.4, 533.6, 533.7, 534.1, 534.8, 535.3,
           535.4, 535.9, 536.1, 536.3, 536.4, 536.6, 537.0, 537.4, 537.5, 538.3,
           538.5, 538.6, 539.4, 539.6, 540.4, 540.8, 542.0, 542.8, 543.0, 543.5,
           543.8, 543.9, 545.3, 546.2, 548.8, 548.7, 548.9, 549.0, 549.4, 549.9,
           550.6, 551.2, 551.4, 551.5, 551.6, 552.8, 552.9,553.2), N = 48, alpha = c(1, 1),
   T = c(1, NA, NA, NA, NA, NA, NA, NA, NA, NA,
         NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
         NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
         NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
         NA, NA, NA, NA, NA, NA, NA, 2))
```

**Initial:**

```
list(lambda = c(535, NA), theta = 5, tau = 0.1)
```

1.000 iterações de aquecimento seguidas de 10.000 adicionais nos dão as seguintes estimativas:

| | mean | sd | MC_error | val2.5pc | median | val97.5pc | start | sample |
|---|---|---|---|---|---|---|---|---|
| P[1] | 0.6014 | 0.08981 | 0.002305 | 0.4267 | 0.602 | 0.7701 | 1001 | 10000 |
| P[2] | 0.3986 | 0.08981 | 0.002305 | 0.2299 | 0.398 | 0.5733 | 1001 | 10000 |
| lambda[1] | 536.8 | 1.023 | 0.03708 | 535.0 | 536.7 | 539.0 | 1001 | 10000 |
| lambda[2] | 548.9 | 1.388 | 0.03856 | 546.0 | 548.9 | 551.3 | 1001 | 10000 |
| sigma | 3.805 | 0.726 | 0.03322 | 2.932 | 3.652 | 6.014 | 1001 | 10000 |

## 6. Exemplo: Leuk – Cox Regression

Outro exemplo extraído do volume 01 trata do modelo de Regressão Cox em um contexto de Análise de Sobrevivência.

Para os indivíduos i = 1, ..., n, temos processos $N_i(t)$ que contam o número de falhas que ocorreram até o tempo t. O correspondente processo de intensidade é dado por:

$$I_i(t)dt = E(dN_i(t) \mid F_{t-})$$

Onde $dN_i(t)$ é o incremento de $N_i$ até o intervalo de tempo [t, t+dt) e $F_{t-}$ representa os dados disponíveis até o tempo imediatamente anterior a t. Se o indivíduo i é observado falhar neste intervalo, $dN_i(t)$ assume o valor 1, caso contrário, assume o valor 0. Portanto, $E(dN_i(t) \mid F_t)$ corresponde à probabilidade do indivíduo i falhar no intervalo [t, t+dt). Sendo dt ->0 (assumindo o tempo como contínuo), então esta probabilidade se torna o risco instantâneo no tempo t para o indivíduo i. Isto é assumido ter a forma de riscos proporcionais:

$$I_i(t) = Y_i(t)\lambda_0(t)\exp(\beta \mathbf{z}_i)$$

Onde Y é um processo assumindo os valores 0 ou 1 dependendo se o indivíduo i é, ou não, observado no tempo t e $\lambda_0(t)\exp(\beta \mathbf{z}_i)$ é o conhecido Modelo de Regressão de Cox.

Temos então os dados observados $D = (N_i(t), Y_i(t), \mathbf{z}_i)$ $i = 1,..n$ e os parâmetros desconhecidos $\beta$ e $\Lambda_0(t) = \text{Integral}(\lambda_0(u), u, t, 0)$, o último a ser estimado não-parametricamente. A distribuição conjunta a posteriori para o modelo acima é dada por:

$$P(\beta, \Lambda_0() \mid D) \sim P(D \mid \beta, \Lambda_0()) P(\beta) P(\Lambda_0())$$

Para o BUGS, nós devemos especificar a forma da verossimilhança $P(D \mid \beta, \Lambda_0())$ e as distribuições à priori para $\beta$ e $\Lambda_0()$. Sob a suposição de censura não-informativa, a verossimilhança para os dados é proporcional a

$$\prod_{i=1}^{n}[\prod_{t>=0} I_i(t)^{dN_i(t)}] \exp(-\int I_i(t)dt)$$

Isto é, essencialmente, como se os incrementos $dN_i(t)$ do processo de contagem no intervalo de tempo [t, t+dt) fossem variáveis aleatórias independentes Poisson com médias $I_i(t)dt$:

$$dN_i(t) \sim \text{Poisson}(I_i(t)dt)$$

E podemos escrever:

$$I_i(t)dt = Y_i(t)\exp(\beta \mathbf{z}_i)d\Lambda_0(t)$$

Onde $d\Lambda_0(t) = \Lambda_0(t)dt$ é o incremento na função de risco acumulada ocorrendo no intervalo de tempo [t, t+dt). Visto que a priori conjugada para a média da Poisson é a Gama, seria conveniente se $\Lambda_0()$ fosse um processo no qual os incrementos $d\Lambda_0(t)$ fossem distribuídos de acordo com a distribuição Gama. Assumiremos, portanto, a priori conjugada para incrementos independentes sugerida por Kalbfleisch (1978):

$$d\Lambda_0(t) \sim \text{Gamma}(cd\Lambda^*_0(t), c)$$

Aqui, $d\Lambda^*_0(t)$ pode ser pensada como uma priori suposta para a função de risco desconhecida, com c representando o grau de confiança nesta suposição. Valores pequenos de c correspondem a prioris não-informativas. No exemplo, utilizaremos $d\Lambda^*_0(t) = r\,dt$ onde r é uma suposição (chute inicial) para a taxa de falha por unidade de tempo e dt é o tamanho do intervalo de tempo.
A formulação acima é apropriada quando prioris informativas genuínas existem, relativas à subjacente função de risco. Alternativamente, se desejássemos reproduzir uma análise de Cox, mas com uma estrutura hierárquica adicional, poderíamos usar, como um artifício, a Multinomial-Poisson descrita no manual do BUGS. Isto é equivalente a assumir incrementos independentes nas prioris não-informativas cumulativas. Esta formulação também é mostrada abaixo.
Para os parâmetros $\beta$, são assumidas as seguintes prioris vagas:

$$\beta \sim \text{Normal}(0.0, 0.000001)$$

**Model:**

```
model
{
# Set up data
    for(i in 1:N) {
            for(j in 1:T) {
# risk set = 1 if obs.t >= t
                Y[i,j] <- step(obs.t[i] - t[j] + eps)
# counting process jump = 1 if obs.t in [ t[j], t[j+1] )
#               i.e. if t[j] <= obs.t < t[j+1]
                dN[i, j] <- Y[i, j] * step(t[j + 1] - obs.t[i] - eps) * fail[i]
            }
    }
# Model
    for(j in 1:T) {
            for(i in 1:N) {
                dN[i, j]   ~ dpois(Idt[i, j])            # Likelihood
                Idt[i, j] <- Y[i, j] * exp(beta * Z[i]) * dL0[j]  # Intensity
            }
            dL0[j] ~ dgamma(mu[j], c)
            mu[j] <- dL0.star[j] * c    # prior mean hazard
# Survivor function = exp(-Integral{l0(u)du})^exp(beta*z)
            S.treat[j] <- pow(exp(-sum(dL0[1 : j])), exp(beta * -0.5));
            S.placebo[j] <- pow(exp(-sum(dL0[1 : j])), exp(beta * 0.5));
    }
    c <- 0.001
    r <- 0.1
    for (j in 1 : T) {
            dL0.star[j] <- r * (t[j + 1] - t[j])
    }
    beta ~ dnorm(0.0,0.000001)
}
```

**Data:**

```
list(N = 42, T = 17, eps = 1.0E-10,
    obs.t = c(1, 1, 2, 2, 3, 4, 4, 5, 5, 8, 8, 8, 8, 11, 11, 12, 12, 15, 17, 22, 23, 6,
    6, 6, 6, 7, 9, 10, 10, 11, 13, 16, 17, 19, 20, 22, 23, 25, 32, 32, 34, 35),
    fail = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0),
    Z = c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
        -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5),
    t = c(1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 15, 16, 17, 22, 23, 35))
```

**Initial:**

```
list( beta = 0.0,
    dL0 = c(1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
        1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0))
```

1000 iterações de aquecimento seguidas de mais 10.000 forneceram os seguintes resultados:

| | mean | sd | MC_error | val2.5pc | median | val97.5pc | start | sample |
|---|---|---|---|---|---|---|---|---|
| S.placebo[1] | 0.9282 | 0.04863 | 5.004E-4 | 0.8094 | 0.9387 | 0.9907 | 1001 | 10000 |
| S.placebo[2] | 0.8538 | 0.06843 | 7.467E-4 | 0.6926 | 0.8639 | 0.9571 | 1001 | 10000 |
| S.placebo[3] | 0.8161 | 0.07561 | 7.661E-4 | 0.6422 | 0.8244 | 0.9362 | 1001 | 10000 |
| S.placebo[4] | 0.7432 | 0.08534 | 8.86E-4 | 0.5586 | 0.7503 | 0.8892 | 1001 | 10000 |
| S.placebo[5] | 0.6703 | 0.09256 | 9.762E-4 | 0.4749 | 0.6755 | 0.835 | 1001 | 10000 |
| S.placebo[6] | 0.5633 | 0.09747 | 9.302E-4 | 0.3666 | 0.5661 | 0.7477 | 1001 | 10000 |
| S.placebo[7] | 0.5304 | 0.09778 | 9.097E-4 | 0.338 | 0.5336 | 0.7148 | 1001 | 10000 |
| S.placebo[8] | 0.4142 | 0.09387 | 8.073E-4 | 0.2374 | 0.4119 | 0.6037 | 1001 | 10000 |
| S.placebo[9] | 0.3812 | 0.09325 | 8.172E-4 | 0.2086 | 0.3779 | 0.5701 | 1001 | 10000 |
| S.placebo[10] | 0.32 | 0.08945 | 8.307E-4 | 0.1583 | 0.315 | 0.509 | 1001 | 10000 |
| S.placebo[11] | 0.2583 | 0.0845 | 7.771E-4 | 0.111 | 0.2511 | 0.4395 | 1001 | 10000 |
| S.placebo[12] | 0.2257 | 0.08105 | 7.359E-4 | 0.08703 | 0.2181 | 0.402 | 1001 | 10000 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S.placebo[13] | 0.1956 | 0.07723 | 7.293E-4 | 0.06867 | 0.1873 | 0.3668 | 1001 | 10000 |
| S.placebo[14] | 0.1656 | 0.07326 | 6.788E-4 | 0.04889 | 0.1567 | 0.3298 | 1001 | 10000 |
| S.placebo[15] | 0.1398 | 0.06788 | 6.183E-4 | 0.03602 | 0.1305 | 0.2953 | 1001 | 10000 |
| S.placebo[16] | 0.0867 | 0.05455 | 5.259E-4 | 0.01301 | 0.07663 | 0.22 | 1001 | 10000 |
| S.placebo[17] | 0.04445 | 0.03913 | 4.092E-4 | 0.002506 | 0.03349 | 0.1484 | 1001 | 10000 |
| S.treat[1] | 0.983 | 0.01372 | 1.541E-4 | 0.9473 | 0.9866 | 0.9982 | 1001 | 10000 |
| S.treat[2] | 0.9643 | 0.02175 | 2.58E-4 | 0.9115 | 0.9692 | 0.9922 | 1001 | 10000 |
| S.treat[3] | 0.9544 | 0.02538 | 3.003E-4 | 0.8918 | 0.9598 | 0.9884 | 1001 | 10000 |
| S.treat[4] | 0.9343 | 0.03217 | 4.071E-4 | 0.8573 | 0.9398 | 0.9797 | 1001 | 10000 |
| S.treat[5] | 0.9125 | 0.03913 | 5.007E-4 | 0.821 | 0.9185 | 0.9701 | 1001 | 10000 |
| S.treat[6] | 0.8772 | 0.04896 | 6.526E-4 | 0.7654 | 0.8838 | 0.9521 | 1001 | 10000 |
| S.treat[7] | 0.8652 | 0.05234 | 6.984E-4 | 0.745 | 0.8717 | 0.947 | 1001 | 10000 |
| S.treat[8] | 0.8178 | 0.06456 | 8.45E-4 | 0.6736 | 0.8246 | 0.9229 | 1001 | 10000 |
| S.treat[9] | 0.8024 | 0.06872 | 9.057E-4 | 0.6528 | 0.8099 | 0.9151 | 1001 | 10000 |
| S.treat[10] | 0.771 | 0.07613 | 9.916E-4 | 0.6064 | 0.7786 | 0.8976 | 1001 | 10000 |
| S.treat[11] | 0.7339 | 0.08462 | 0.001154 | 0.5522 | 0.7409 | 0.8774 | 1001 | 10000 |
| S.treat[12] | 0.7114 | 0.08897 | 0.001204 | 0.5224 | 0.7174 | 0.8659 | 1001 | 10000 |
| S.treat[13] | 0.6882 | 0.0932 | 0.001268 | 0.4913 | 0.6937 | 0.8528 | 1001 | 10000 |
| S.treat[14] | 0.6619 | 0.097 | 0.001318 | 0.4641 | 0.6669 | 0.8355 | 1001 | 10000 |
| S.treat[15] | 0.636 | 0.1007 | 0.00137 | 0.4318 | 0.6406 | 0.8191 | 1001 | 10000 |
| S.treat[16] | 0.5662 | 0.111 | 0.001493 | 0.3453 | 0.5688 | 0.773 | 1001 | 10000 |
| S.treat[17] | 0.4761 | 0.1189 | 0.001548 | 0.2502 | 0.4747 | 0.7085 | 1001 | 10000 |
| beta | 1.538 | 0.4176 | 0.005644 | 0.7718 | 1.521 | 2.384 | 1001 | 10000 |

## 7. Exemplo: Ciências Atuariais

Outra aplicação do uso do pacote pode ser encontrado em um problema que as seguradoras freqüentemente encontram, que é o de constituir provisões matemáticas. No artigo "Método Bayesiano para Provisão de Eventos Ocorridos mas Não Avisados", de Leonardo Melo e Ricardo Ehlers, faz-se uso da modelagem MCMC para este caso.
Abaixo segue um dos modelos utilizado neste estudo:

$$Y_{ij} = \log(X_{ij}), \text{ com } X_{ij} \sim N(\mu_{ij}, \sigma^2)$$
$$\mu_{ij} = \mu + \alpha_i + \beta_j$$
$$\mu \sim P * N(\theta_1, \sigma_\mu^2) + (1-P) * N(\theta_2, \sigma_\mu^2)$$
$$\alpha_i \sim N(o, \sigma_\alpha^2)$$
$$\beta_i \sim N(o, \sigma_\beta^2)$$

Sendo:

$Y_{ij}$ os montantes de sinistros ajustados pelo logaritmo;
$\mu$ o valor esperado dos montantes de sinistros ajustados pelo logaritmo;
$\alpha_i$ o efeito do i-ésimo mês;
$\beta_{ij}$ o efeito do j-ésimo período de atraso referente ao i-ésimo mês;

**Modelo:**

```
model;
{
        for (i in 1 : r) {
                for(j in 1 : r){
                        y[i , j] ~ dnorm(mu[i , j], tau)
                        mu[i , j]<- mu.y + alpha.y[i] + beta.y[j]
```

```
            }
        }
        alpha.y[1]<- 0
        beta.y[1]<- 0
            for (i in 2 : r){
            alpha.y[i] ~ dnorm(0.0, tau.alpha)
            beta.y[i] ~ dnorm(0.0, tau.beta)
            }
        mu.y ~ dnorm(theta[T], tau.y)
        T ~ dcat(P[])
        P[1:2] ~ ddirch(delta[])
        delta[1]<- 1
        delta[2]<- 1
        theta[2] <- theta[1] + k
        theta[1] ~ dnorm(0.0, tau.theta)
        k ~ dnorm(0.0, 1.0E-3)
        tau ~ dgamma(0.001 , 0.001)
        tau.y<- 1 / 1000
        tau.alpha <- 1 / 100
        tau.beta <- 1 / 100
        tau.theta<- 1/100
for( i in 2 : r ) {
    outstand.row[ i ] <- sum( y[ i, ( r +2 - i ) : r ] ) / 1000
  }
        outstand.cal[2]<- (y[2,r] + y[3,r-1] + y[4,r-2] + y[5,r-3] + y[6,r-4] + y[7,r-5] + y[8,r-6]+y[9,r-7] + y[10,r-8] + y [11, r-9] + y[12, r-10]) / 1000
        outstand.cal[3]<- (y[3,r] + y[4,r-1] + y[5,r-2] + y[6,r-3] + y[7,r-4] + y[8,r-5]+y[9,r-6] + y[10,r-7] + y [11, r-8] + y[12, r-9]) / 1000
        outstand.cal[4]<- (y[4,r] + y[5,r-1] + y[6,r-2] + y[7,r-3] + y[8,r-4]+y[9,r-5] + y[10,r-6] + y [11, r-7] + y[12, r-8]) / 1000
        outstand.cal[5]<- (y[5,r] + y[6,r-1] + y[7,r-2] + y[8,r-3]+y[9,r-4] + y[10,r-5] + y [11, r-6] + y[12, r-7]) / 1000
        outstand.cal[6]<- (y[6,r] + y[7,r-1] + y[8,r-2]+ y[9,r-3] + y[10,r-4] + y [11, r-5] + y[12, r-6]) / 1000
        outstand.cal[7]<- (y[7,r] + y[8,r-1]+ y[9,r-2] + y[10,r-3] + y [11, r-4] + y[12, r-5]) / 1000
        outstand.cal[8]<- (y[8,r]+ y[9,r-1] + y[10,r-2] + y [11, r-3] + y[12, r-4]) / 1000
        outstand.cal[9]<- (y[9,r] + y[10,r-1] + y [11, r-2] + y[12, r-3]) / 1000
        outstand.cal[10]<- (y[10,r] + y [11, r-1] + y[12, r-2]) / 1000
        outstand.cal[11]<- (y [11, r] + y[12, r-1]) / 1000
        outstand.cal[12]<- (y[12, r]) / 1000
        outstand.cal[13]<- sum(outstand.cal[2 : 12])
}
```

**Data:**

```
list(r=12,
y=structure(.Data = c(
15.38,  16.67,  15.25,  13.80,  12.00,  10.68,  10.91,  9.76,  7.52,  10.04,  7.32,  7.26,
15.57,  16.61,  15.06,  13.07,  11.73,  11.93,  10.53,  8.89,  9.11,  9.35,  7.02,  NA,
15.66,  16.76,  15.32,  13.16,  12.40,  11.53,  10.41,  10.58,  10.21,  8.07,  NA,  NA,
15.55,  16.74,  15.21,  13.59,  11.69,  10.38,  9.74,  11.01,  8.32,  NA,  NA,  NA,
15.47,  16.74,  15.24,  13.53,  12.02,  12.52,  10.13,  9.65,  NA,  NA,  NA,  NA,
15.61,  16.76,  15.22,  13.80,  12.57,  12.26,  10.58,  NA,  NA,  NA,  NA,  NA,
15.40,  16.64,  15.19,  13.45,  11.76,  11.12,  NA,  NA,  NA,  NA,  NA,  NA,
15.48,  16.45,  15.05,  13.51,  11.79,  NA,  NA,  NA,  NA,  NA,  NA,  NA,
15.25,  16.59,  15.07,  13.89,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,
15.39,  16.46,  15.01,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,
15.66,  16.76,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,
15.53,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA,  NA), .Dim=c(12,12)))
```

**Initial:**

```
list(mu.y = 0.0, lambda=c(10, NA), tau = 0.001, k=5)
```

## 8. Diagrama de Influência

Conforme pudemos perceber, o Diagrama de Influência é útil para verificar as relações entre as variáveis e nos ajudar na construção de nosso modelo. No WinBUGS existe uma ferramenta própria para construção destes diagramas, sendo de fácil utilização.

Para construir um Diagrama, vá no menu "**Doodle**" e clique em "**New**". Especifique o tamanho do gráfico e clique em "*ok*". Uma nova janela se abrirá. É onde você poderá "desenhar" o diagrama, conforme segue:

- **Criar um nó:** basta clicar em qualquer área do gráfico. Para movê-lo, clique sobre ele e arraste-o com o mouse. O formato do nó (circular ou retangular) muda conforme você seleciona seu tipo.
- **Criar uma placa:** as "placas" são os retângulos ("loops") onde os nós com índices devem ficar. Parar criar uma delas, clique sobre a área do gráfico mantendo pressionada a tecla *ctrl.* Você pode arrastá-las ou redimensioná-las.
- **Criar uma seta:** selecione o nó de destino, depois clique no nó de origem mantendo pressionada a tecla *ctrl.* Para deletar a seta, basta clicar novamente. O tipo da seta é automaticamente selecionada, dependendo das relações que você estabeleceu entre as variáveis.
- **Definir as variáveis:** note que, ao selecionar um nó, um pequeno menu se abre na parte superior da janela. Ali você descreve o nome da variável, seu tipo e seu valor (se for um nó do tipo "lógico") ou sua distribuição (se for um nó do tipo "estocástico").
- **Deletar um nó ou uma placa:** basta selecionar o objeto que você deseja deletar e digitar *ctrl+del*.

Depois que o gráfico estiver pronto, basta ir no menu "Doodle" novamente e clicar em "Write Code". Se não houver nenhum erro em seu gráfico, o WinBUGS automaticamente irá construir o seu arquivo de modelo.