Kiley Delaney/kileyfd2/675040797 CS 598 Advanced Bayesian Modeling Assignment 2

```
#!apt-get install -y jags
```

```
install.packages("rjags")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

# Problem 1

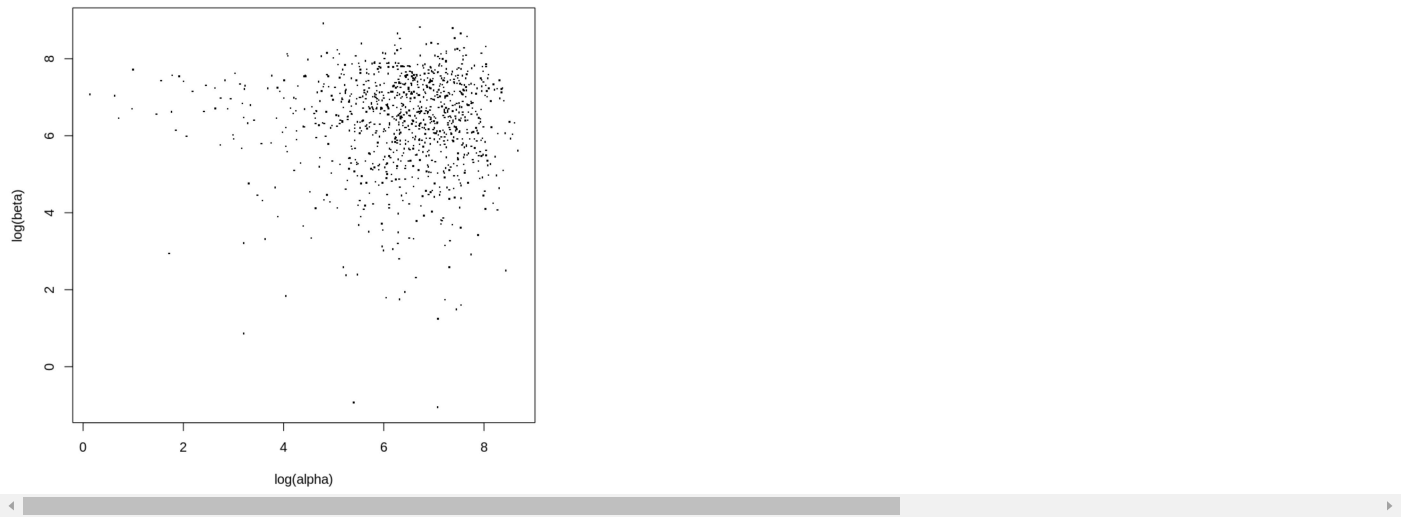
(a) The first prior formulation was

$\theta_j \mid \alpha, \beta \sim$ Beta($\alpha, \beta$)

$\alpha, \beta \sim$ Expon(0.001)

(i) Independently simulate 1000 pairs (α, β) from their hyperprior, and produce a scatterplot of log(β) versus log(α).

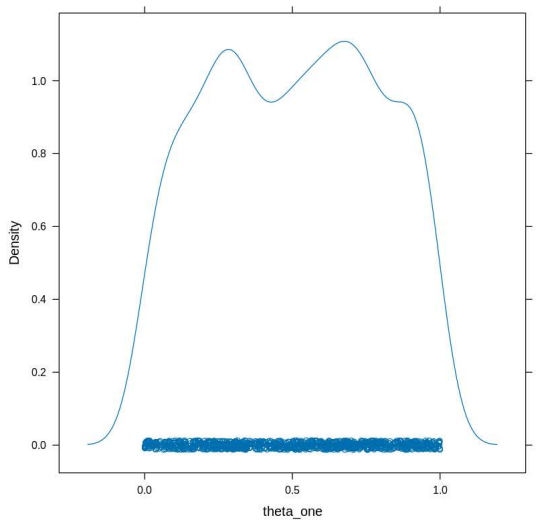
```
alpha <- rexp(n = 1000, rate =0.001)
beta <- rexp(n = 1000, rate =0.001)
plot(log(alpha), log(beta), pch=".", cex=2)
```




(ii) Using the simulated pairs (α, β), forward-simulate $\theta_1$, and produce a histogram of the result (an approximation of its marginal prior).

```
theta_one <- rbeta(1000, alpha, beta)
```

```
#hist(theta_one)
require(lattice)
densityplot(theta_one)
```




(b) The second prior formulation was

$\theta_j \mid \alpha, \beta \sim$ Beta($\alpha, \beta$)

$\alpha = \varphi_1 / \varphi_2^2$

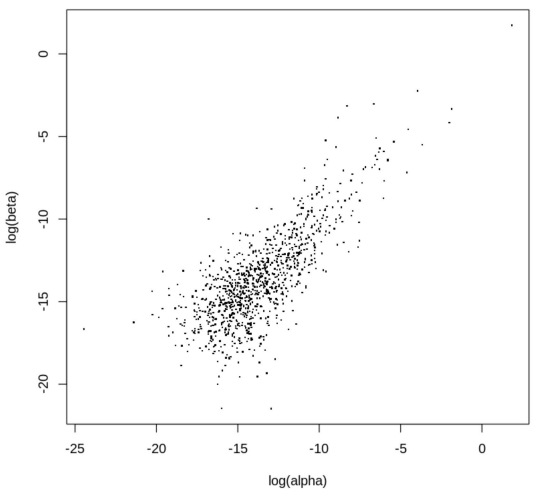
$\beta = (1 - \varphi_1) / \varphi_2^2$

$\varphi_1 \sim$ U(0,1)

$\varphi_2 \sim$ Expon(0.001)

(i) Independently simulate 1000 pairs (α, β) from their hyperprior, and produce a scatterplot of log(β) versus log(α).

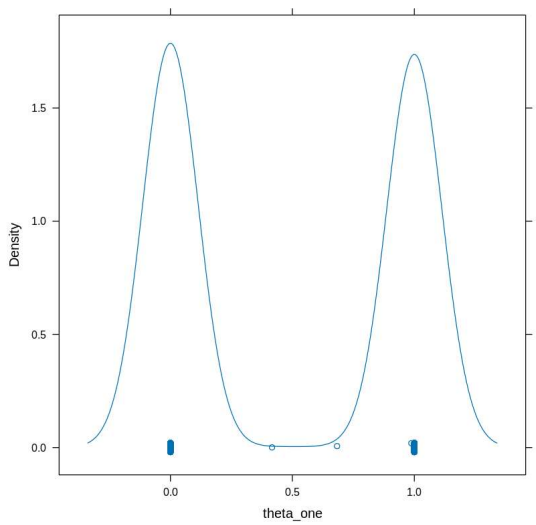
```
sig_one <- runif(n=1000, 0, 1)
sig_two <- rexp(n = 1000, rate =0.001)
alpha <- sig_one / (sig_two ^ 2)
beta <- (1 - sig_one) / (sig_two ^ 2)

plot(log(alpha), log(beta), pch=".", cex=2)
```




(ii) Using the simulated pairs (α, β), forward-simulate $\theta_1$, and produce a histogram of the result (an approximation of its marginal prior).

```
theta_one <- rbeta(1000, alpha, beta)
```

```
#hist(theta_one)
require(lattice)
densityplot(theta_one)
```




## Problem 2

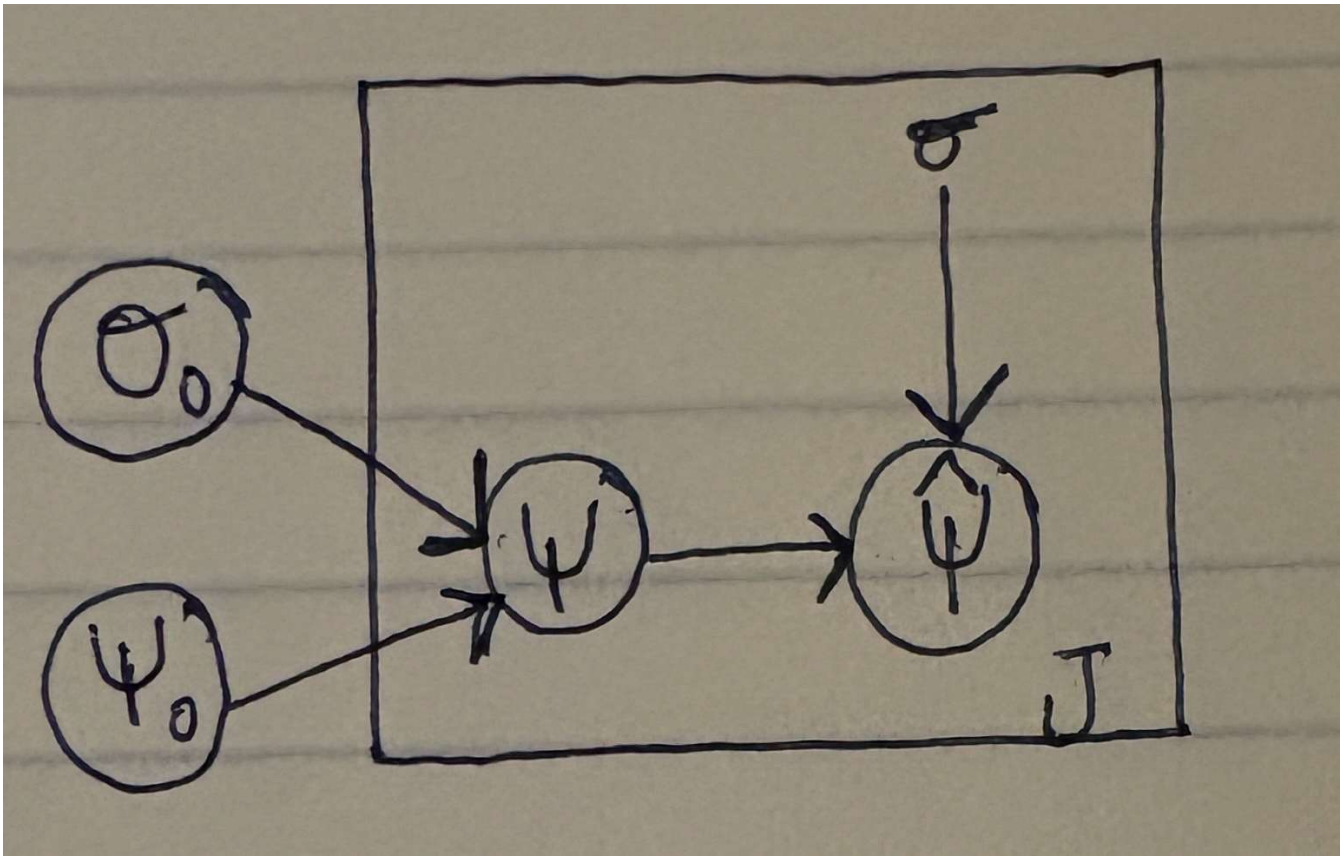
(a) Specify improper densities that the proper hyperpriors given above are apparently intended to approximate. (Which parameters are the hyperparameters?)

The hyperparameters are as follows:

$\psi_0 \sim \mathrm{N}(0, 1000^2)$

$\sigma_0 \sim \mathrm{U}(0, 1000)$

(b) Draw a directed acyclic graph (DAG) appropriate for this model. (Use the notation introduced in lecture, including "plates.") You may draw it neatly by hand or use software.




(c) Using the template asgn2template.bug provided on the course website, form a JAGS model statement (consistent with your DAG). Show your JAGS code. [Remember: JAGS "dnorm" uses precisions, not variance]

```
library(rjags)
```

```
'model {
  for (j in 1:12) {
    psihat[j] ~ dnorm(psi[j], 1/sigma[j]^2)
    psi[j] ~ dnorm(psi0, 1/sigma0^2)
  }

  psi0 ~ dnorm(0,1/1000^2)
  sigma0 ~ dunif(0,1000)

  sigmasq0 <- sigma0^2
}'
```

'model {\n  for (j in 1:12) {\n    psihat[j] ~ dnorm(psi[j], 1/sigma[j]^2)\n    psi[j] ~ dnorm(psi0, 1/sigma0^2)\n  }\n\n  psi0 ~ dnorm(0,1/1000^2)\n  sigma0 ~ dunif(0,1000)\n\n

(d) Set up any R (rjags) statements appropriate for creating a JAGS model. Show your R code, and also show (print) the R list or data frame that you are passing to JAGS. Double check that the variable names in the list or data frame exactly match the corresponding names in your JAGS model, and double check your numbers.

```
data <- read.table("numbersdata.txt", header=FALSE)
colnames(data) <- c("psi", "sigma")
```

```
data
```

A data.frame: 12 × 2

| psi <dbl> | sigma <dbl> |
|---|---|
| 1.055 | 0.373 |
| -0.097 | 0.116 |
| 0.626 | 0.229 |
| 0.017 | 0.117 |
| 1.068 | 0.471 |
| -0.025 | 0.120 |
| -0.117 | 0.220 |
| -0.381 | 0.239 |
| 0.507 | 0.186 |
| 0.000 | 0.328 |
| 0.385 | 0.206 |
| 0.405 | 0.254 |

```
model1 <- jags.model("filledtemplate.bug", data)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 12
   Unobserved stochastic nodes: 14
   Total graph size: 70

Initializing model
```

(e) Run at least 10,000 iterations of burn-in, then 100,000 iterations to use for inference. For both $\psi_0$ and $\sigma_0^2$ (not $\sigma_0$), produce a posterior numerical summary and also graphical estimates of the posterior densities. Explicitly give the approximations of their posterior expected values, posterior standard deviations, and 95% central posterior intervals. (Just showing R output is not enough!)

```
update(model1, 10000)
```

```
iterations = coda.samples(model1, c("psi0","sigmasq0","psi"), n.iter=100000)
```

```
psi0<- as.matrix(iterations)[,"psi0"]
sigmasq0<- as.matrix(iterations)[,"sigmasq0"]
```

```
summary(psi0)
summary(sigmasq0)
psi0_mean=mean(psi0)
sigmasq0_mean=mean(sigmasq0)
psi0_sd=sd(psi0)
sigmasq0_sd=sd(sigmasq0)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.0103  0.1889  0.2878  0.2876  0.3863  1.4777
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.05292 0.19082 0.25584 0.29933 0.35553 3.30118
```

The posterior expected values are approximately 0.2876 for $\Psi$ and approximately 0.2993 for $\sigma$.

```
psi0_mean
sigmasq0_mean
```

```
0.28762532144302
0.2993336411581
```

The posterior standard deviations are approximately 0.1573 for $\Psi$ and approximately 0.1723 for $\sigma$.

```
psi0_sd
sigmasq0_sd
```

```
0.157313746580119
0.172308354206091
```

The 95% cenral posterior interval for $\Psi$ is approximately (0.2867, 0.2886). The 95% central posterior interval for $\sigma$ is approximately (0.2983, 0.3004).

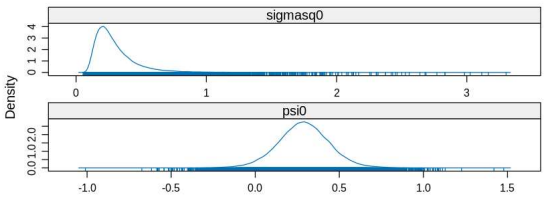
```
confidenceintervalhelper <- function(n, sd , mean){

    error <- qnorm(0.975)* sd/sqrt(n)
    left <- mean - error
    right <- mean + error
    confidenceinterval <- list(left,right)
    return (confidenceinterval)
}

psi0_CI = confidenceintervalhelper(100000, psi0_sd, psi0_mean)
sigmasq0_CI = confidenceintervalhelper(100000, sigmasq0_sd, sigmasq0_mean)
```

```
psi0_CI
sigmasq0_CI
```

```
1. 0.286650298656585
2. 0.288600344229456

1. 0.298265682538485
2. 0.300401599777715
```

The graphical estimates for the posterior densities are as follows:

```
require(lattice)
densityplot(iterations[,c("psi0","sigmasq0")])
```




A posterior numerical summary can be seen in the data frame below.

```
posterior.expected.vals <- c(psi0_mean, sigmasq0_mean)
posterior.sds <- c(psi0_sd, sigmasq0_sd)
confidence.interval <- rbind(psi0_CI, sigmasq0_CI)
```

```
summ = data.frame(posterior.expected.vals, posterior.sds, confidence.interval)
row.names(summ) <- c('psi0','sigmasq0')
colnames(summ) <-c ('posterior.expected.vals','posterior.sds','95%confidence.interval.left','95%confidence.interval.right')
```

```
summ
```

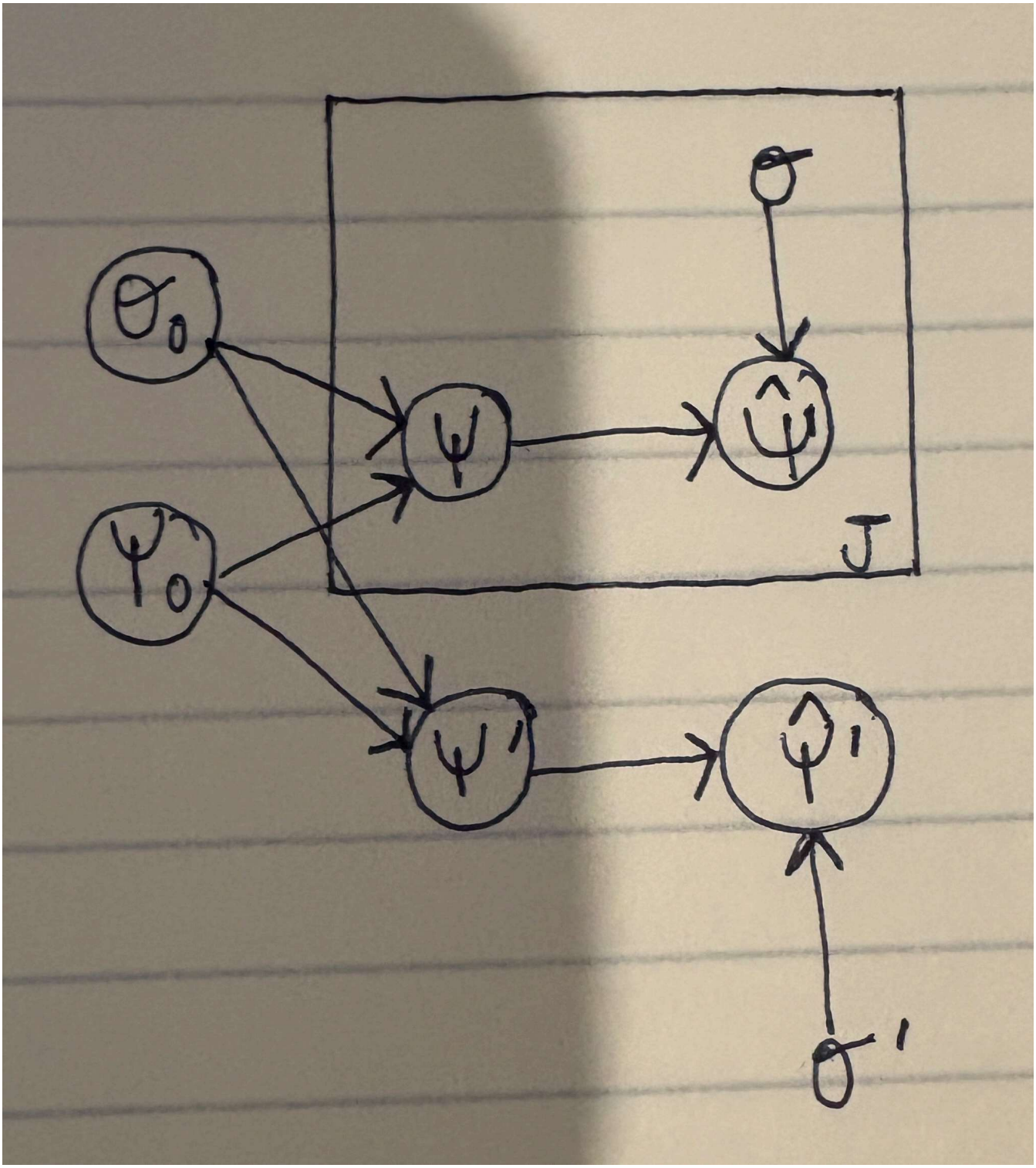
A data.frame: 2 × 4

| | posterior.expected.vals | posterior.sds | 95%confidence.interval.left | 95%confidence.interval.right |
|---|---|---|---|---|
| | <dbl> | <dbl> | <named list> | <named list> |
| **psi0** | 0.2876253 | 0.1573137 | 0.2866503 | 0.2886003 |
| **sigmasq0** | 0.2993336 | 0.1723084 | 0.2982657 | 0.3004016 |

(f) Suppose a new case-control study is to be performed, and assume that its log-odds standard error (new σ) will be 0.125. Assume the ψ for the new study is exchangeable with those for the previous studies (under the Bayesian model).

Use at least 10,000 iterations of burn-in, and 100,000 for inference as before.

[i] Re-draw your DAG, adding new nodes to represent the new $\hat{\psi}$ and new ψ.




[ii] s] Correspondingly modify your JAGS model to answer the following parts. Show the modified JAGS and R code and output that you used.

The updated JAGS model is as follows:

```
'
model {
  for (j in 1:12) {
    psihat[j] ~ dnorm(psi[j], 1/sigma[j]^2
    psi[j] ~ dnorm(psi0, 1/sigma0^2)
  }

  psi_new ~ dnorm(psi0, 1/sigma0^2)
  psihat_new ~ dnorm(psi_new,1/sigma_new^2
  sigma_new <- 0.2
```

```
  sigma_newsq <- sigma_new^2
  psi0 ~ dnorm(0,1/1000^2)
  sigma0 ~ dunif(0,1000)

  sigmasq0 <- sigma0^2
}
'
```

```
'\nmodel {\n  for (j in 1:12) {\n    psihat[j] ~ dnorm(psi[j], 1/sigma[j]^2)\n    psi[j] ~ dnorm(psi0, 1/sigma0^2)\n  }\n\n  psi_new ~ dnorm(psi0, 1/sigma0^2)\n  psihat_new ~ dnorm(psi_new,1/sigma_new^2)\n  sigma_new <- 0.2\n  sigma_newsq <- sigma_new^2\n  psi0 ~ dnorm(0,1/1000^2)\n  sigma0 ~ dunif(0,1000)\n\n  sigmasq0 <- sigma0^2\n}\n'
```

```
model2 <- jags.model("filledtemplate2.bug", data)
```

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
```