Biostat 682 Hw 4

David (Daiwei) Zhang

November 20, 2017

1

(a)

We implement and test the cdf method for truncated normal distribution.

```
cdf.inv <- function(p, mu, mu.lo, sigma){
    z.lo = (mu.lo - mu) / sigma
    a = 1 - pnorm(z.lo)
    q = p * a + (1 - a)
    z = qnorm(q)
    x = mu + z * sigma
    return(x)
}

cdf <- function(x, mu, mu.lo, sigma){
    z = (x - mu) / sigma
    z.lo = (mu.lo - mu) / sigma
    s = 1 - pnorm(z.lo)
    p = (pnorm(z) - pnorm(z.lo)) / s
    p = (p + abs(p)) / 2
    return(p)
}

pdf <- function(x, mu, mu.lo, sigma){
    z = (x - mu) / sigma
    z.lo = (mu.lo - mu) / sigma
    a = 1 - pnorm(z.lo)
    p = dnorm(z) / a * (z > z.lo)
}

# Parameters for testing
n <- 1e4
epsi <- 1e-4
mu <- 0
sigma <- 1
z.lo <- -1
mu.lo <- mu + z.lo*sigma

## Testing pdf
x <- mu.lo - 1
p <- pdf(x, mu, mu.lo, sigma)
stopifnot(p == 0)

z <- 3
x <- mu.lo + z * sigma
```

```
p.crct <- dnorm(z.lo + z) / (1 - pnorm(z.lo))
p <- pdf(x, mu, mu.lo, sigma)
err <- p - p.crct
stopifnot(abs(err) < epsi)

z <- c(-1, 3)
x <- mu.lo + z * sigma
p.crct[1] <- 0
p.crct[2] <- dnorm(z.lo + z[2]) / (1 - pnorm(z.lo))
p <- pdf(x, mu, mu.lo, sigma)
err <- p - p.crct
stopifnot(abs(err) < epsi)

# Testing cdf
x <- mu.lo - 1
p.crct <- 0
p <- cdf(x, mu, mu.lo, sigma)
stopifnot(abs(p - p.crct) < epsi)

z <- z.lo + 1
x <- mu + z*sigma
p.crct <- (pnorm(z) - pnorm(z.lo)) / (1 - pnorm(z.lo))
p <- cdf(x, mu, mu.lo, sigma)
err <- p - p.crct
stopifnot(abs(err) < epsi)

# Testing inverse cdf
p <- 0
x.crct <- mu.lo
x <- cdf.inv(p, mu, mu.lo, sigma)
err <- x - x.crct
stopifnot(abs(err) < epsi)

p <- 0.5
x.crct <- qnorm((1 - pnorm(z.lo)) * p + pnorm(z.lo))
x <- cdf.inv(p, mu, mu.lo, sigma)
err <- x - x.crct
stopifnot(abs(err) < epsi)
```

(b)

(i)&(iii)

For $x > \mu^-$, we have

$$
\begin{aligned}
r(x|\alpha) :=& \frac{f(x|\mu, \mu^-, \sigma^2)}{Mg(x|\alpha, \mu^-)} = \exp\{a(x-b)^2 + c \ - \log M\} \quad \text{where} \\
a =& (-2\sigma^2)^{-1} \\
b =& \mu + \alpha\sigma^2 \\
c =& \mu\alpha + \frac{1}{2}\alpha^2\sigma^2 - \frac{1}{2}\log(2\pi\sigma^2) - \log\{1 - \Phi(\frac{\mu^- - \mu}{\sigma})\} - \log\alpha - \alpha\mu^-
\end{aligned}
$$

Let $M \geq \exp(c)$. Then $r \leq 1$ since $a < 0$. In order to maximize $E[r(x)]_{g(x)} = M^{-1}$, we need to minimize M, which is equivalent to letting $M = \exp(c)$ and minimizing c, with the restriction of $\alpha > 0$. Since

$$
\frac{\partial c}{\partial \alpha} = \sigma^2\alpha - \alpha^{-1} + \mu - \mu^-, \quad \frac{\partial^2 c}{\partial^2 \alpha} = \sigma^2 + \alpha^{-2} > 0,
$$

we have $\frac{\partial c}{\partial \alpha} = 0$ when

$$
\alpha = \frac{-(\mu - \mu^-) \pm \sqrt{(\mu - \mu^-)^2 + 4\sigma^2}}{2\sigma^2}.
$$

Since $\sigma^2 > 0$, we have

$$
|\mu - \mu^-| < \sqrt{(\mu - \mu^-)^2 + 4\sigma^2}.
$$

Then

$$
\alpha = \frac{-(\mu - \mu^-) + \sqrt{(\mu - \mu^-)^2 + 4\sigma^2}}{2\sigma^2}
$$

is always positive and is the only positive solution between the two for α. This is the value of α that maximizes the acceptance rate.

(ii) & (iv)

We implement the rejection sampling method.

```
# Rejection sampling
rejsamp <- function(n, mu, mu.lo, sigma, alpha=0){
    stopifnot(alpha >= 0)
    if(alpha == 0){
        ## alpha = 1/2/sigma^2 * (mu + mu.lo + sqrt((mu+mu.lo)^2+4*sigma^2))
        alpha = 1/2/sigma^2 * (-mu + mu.lo + sqrt((mu-mu.lo)^2+4*sigma^2))
    }
    z.lo = (mu.lo - mu) / sigma
    a = 1 - pnorm(z.lo)
    C = mu * alpha + 1/2*alpha^2*sigma^2 - 1/2*log(2*pi*sigma^2) - log(a) - log(alpha) - alpha*mu.lo
    M = exp(C)
    y = mu.lo + rexp(n, alpha)
    p = pdf(y, mu, mu.lo, sigma) / (dexp(y - mu.lo, alpha) * M)
    q = runif(n)
    x = y[q <= p]
    accept = sum(q <= p) / n
    result = list(x=x, accept=accept, M=M, alpha=alpha)
    return(result)
}
```

We now find the α that theoretically maximizes the acceptance rate.

```
## Finding the alpha with the highest acceptance rate
result.rejsamp <- rejsamp(n, mu, mu.lo, sigma, alpha=0)
alpha.opt <- result.rejsamp$alpha
accept.opt <- result.rejsamp$accept
print(alpha.opt)
```

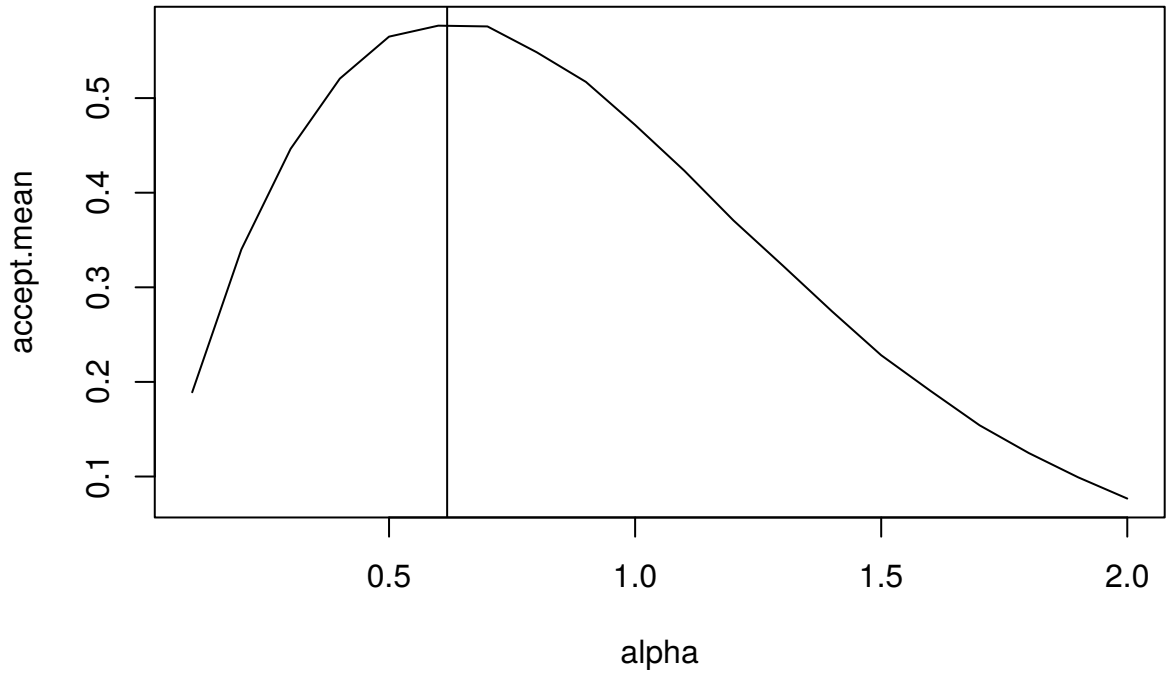
```
## [1] 0.618034
```

```
print(accept.opt)
```

```
## [1] 0.5731
```

Then we compare this value with empirical results.

```
m=20
alpha <- seq(0.1,2,length.out=m)
m <- length(alpha)
h <- 10
accept.mean <- c()
for(i in 1:m){
    accept <- c()
    for(j in 1:h){
        result.rejsamp <- rejsamp(n, mu, mu.lo, sigma, alpha=alpha[i])
        a <- result.rejsamp$accept
        accept <- c(accept, a)
    }
    accept.mn <- mean(accept)
    accept.mean <- c(accept.mean, accept.mn)
}
plot(alpha, accept.mean, type="l")
abline(v=alpha.opt)
```



We see that the theoretical maximizer (the vertical line) for the acceptance rate is consistent with our empirical result. Finally, we compare the accuracy and runtime for the cdf method and rejection sampling

method.

```
## Compare the accuracy and runtime for cdf method and rejection sampling
par(mfrow=c(1,2))
n <- 1e5
x.crct <- rnorm(n, mu, sigma)
x.crct <- x.crct[x.crct >= mu.lo]

## Runtime for the cdf method
print(system.time({
  p <- runif(n, 0, 1)
  x.cdf <- cdf.inv(p, mu, mu.lo, sigma)
}))
```

```
##    user  system elapsed
##   0.008   0.000   0.008
```

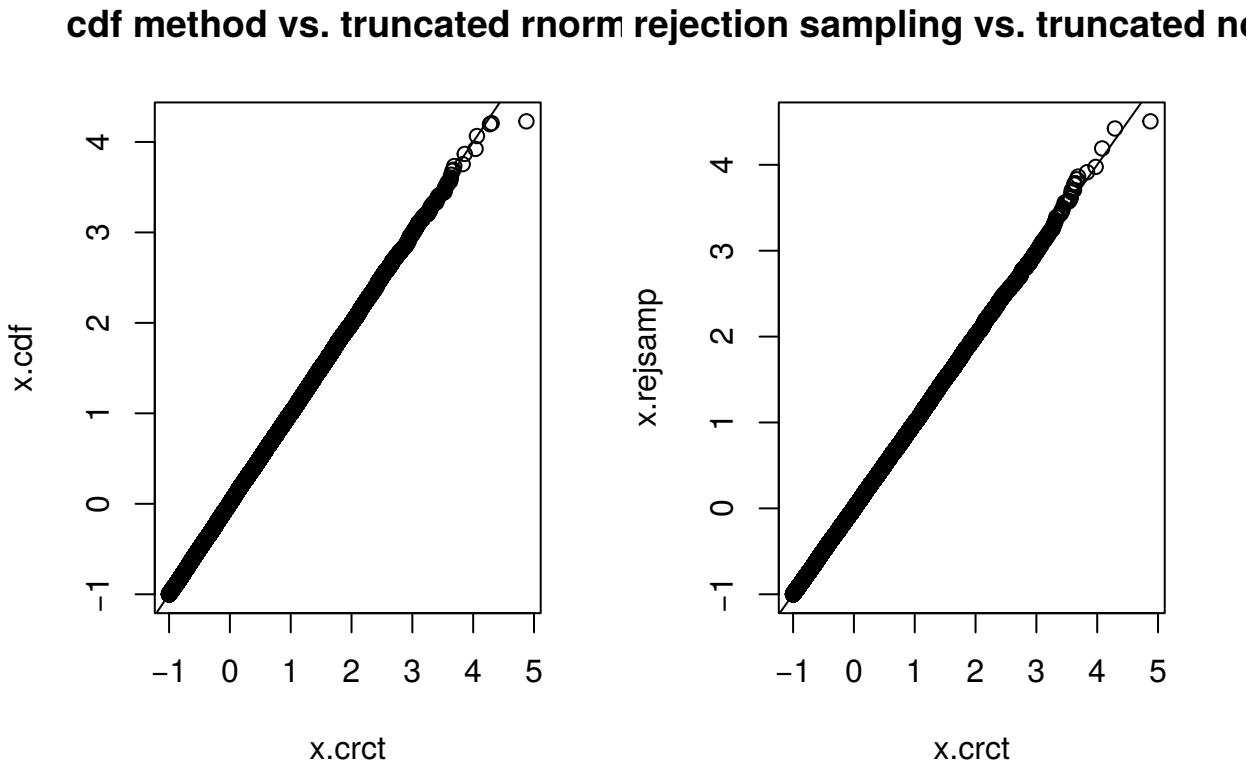
```
qqp.cdf <- qqplot(x.crct, x.cdf, main = "cdf method vs. truncated rnorm")
abline(0,1)
## Accuracy for the cdf method
corr.cdf <- cor(qqp.cdf$x, qqp.cdf$y)
print(corr.cdf)
```

```
## [1] 0.9999782
```

```
## Runtime for the rejection sampling method
print(system.time({
  result.rejsamp <- rejsamp(n, mu, mu.lo, sigma)
}))
```

```
##    user  system elapsed
##   0.020   0.000   0.018
```

```
x.rejsamp <- result.rejsamp$x
qqp.rejsamp <- qqplot(x.crct, x.rejsamp, main = "rejection sampling vs. truncated norm")
abline(0,1)
```




```
## Accuracy for the rejection sampling method
corr.rejsamp <- cor(qqp.rejsamp$x, qqp.rejsamp$y)
print(corr.rejsamp)
```

```
## [1] 0.9999811
```

We see that both methods are very accurate, but rejection sampling is much faster than the cdf method.

3

(a)

We use JAGS to simulate the posterior.

```
linear.model.JAGS = function(){
    for(i in 1:n){
        y[i] ~ dnorm(mu[i], tausq)
        mu[i] <- sum(X[i,] * beta)
    }
    tausq ~ dgamma(1e-3, 1e-3)
    for(i in 1:p){
        beta[i] ~ dnorm(0, tausq*tsq[z+1])
    }
    tsq[1] <- 1/(10*n)
    tsq[2] <- log(n)
    z ~ dbin(0.5, 1)
    sigmasq <- 1/tausq
}
```

```
run.jags <- function(p, n, n.iter = 1e3, n.burnin=1e2){
    mu.X <- matrix(0, p, 1)
    sigma.X <- matrix(0.25, p, p) + diag(rep(0.75, p))
    X <- mvrnorm(n, mu.X, sigma.X)
    beta.true <- c(0.6, 1.2, 1.8, 2.4, 3.0, rep(0, p - 5))
    mu.true <- X %*% beta.true
    sigmasq.true <- 1
    y <- rnorm(n, mu.true, sigmasq.true)

    dat.JAGS = list(y = y, X = X, n = n, p = p)

    para.JAGS = c("beta", "sigmasq")
    inits.JAGS = list(list(beta=rep(0,p), tausq=1))

    fit.JAGS = jags(
        data=dat.JAGS,
        inits=inits.JAGS,
        parameters.to.save = para.JAGS,
        n.chains=1,
        n.iter=n.iter,
        n.burnin=n.burnin,
        model.file=linear.model.JAGS
    )
    return(fit.JAGS)
}

plot.jag <- function(fit.jag){
    ci <- fit.jag$BUGSoutput$summary[c(-(p+1)),c(1,3,7)]
    par(mfrow=c(1,2))
    plotCI(c(1:5), ci[c(1:5),1], li = ci[c(1:5),2], ui = ci[c(1:5),3], main = "beta[1] to beta[5]", xli
    abline(0,0.6)
    plotCI(1, ci[p+1, 1], li = ci[p+1, 2], ui = ci[p+1, 3], main = "sigma^2")
    abline(1, 0)
    par(mfrow=c(1,1))
    plotCI(c(6:p), ci[c(6:p),1], li = ci[c(6:p),2], ui = ci[c(6:p),3], main = "beta[6] to beta[p]")
    abline(0,0)
}
```

(b)

Now we run the simulation with different values for n and p.

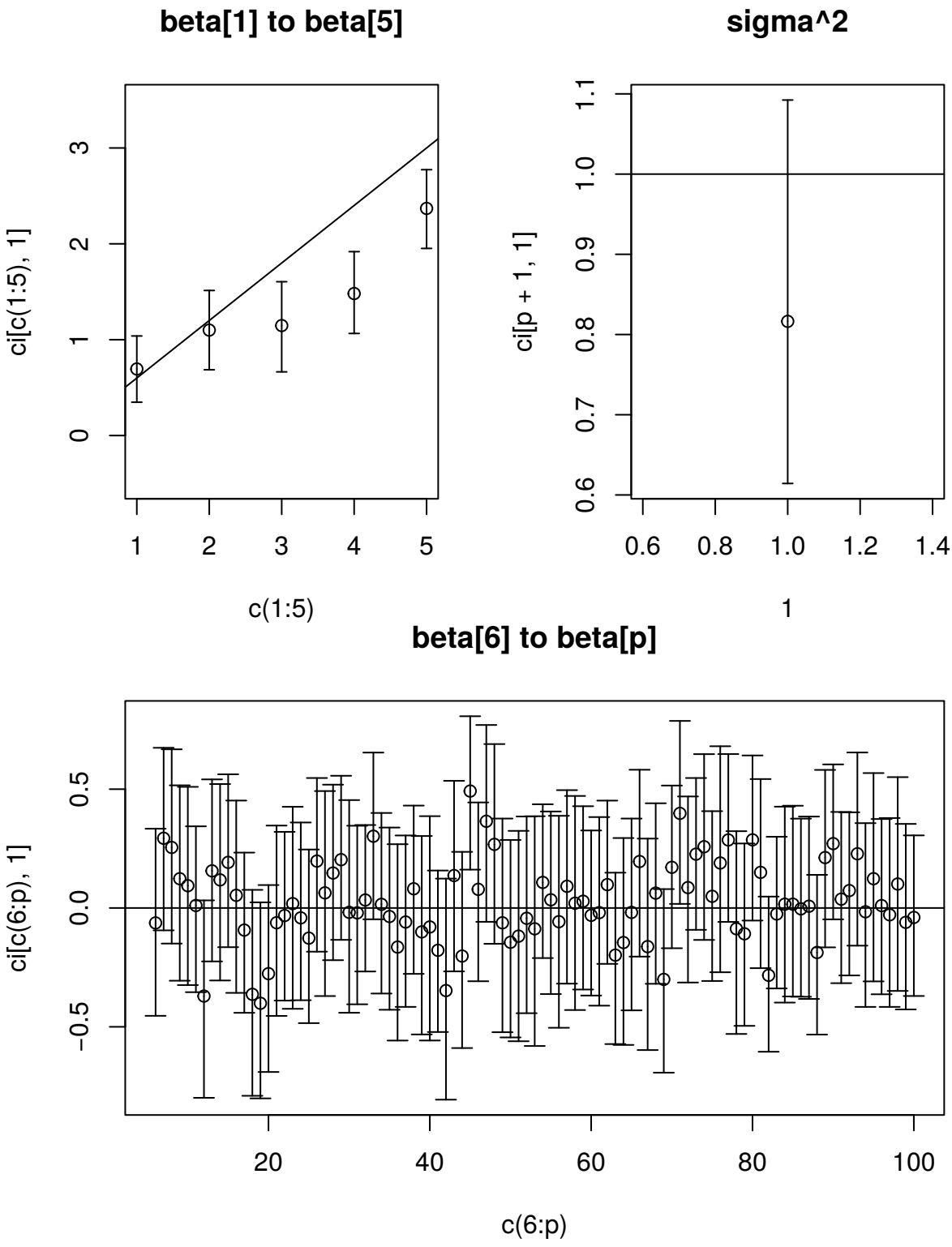
```
i <- 1
n <- 100
p <- 100
fit.jag <- run.jags(p,n)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
```

```
##    Unobserved stochastic nodes: 102
##    Total graph size: 10921
##
## Initializing model
```

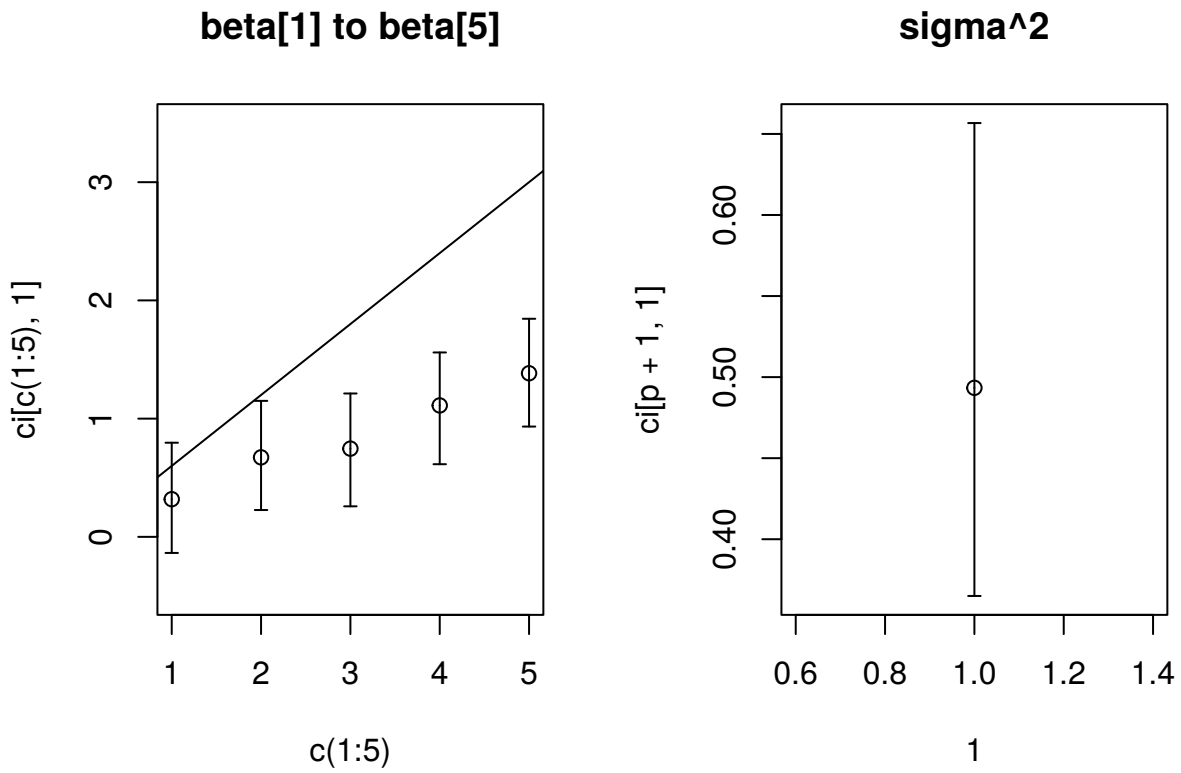
```
plot.jag(fit.jag)
```

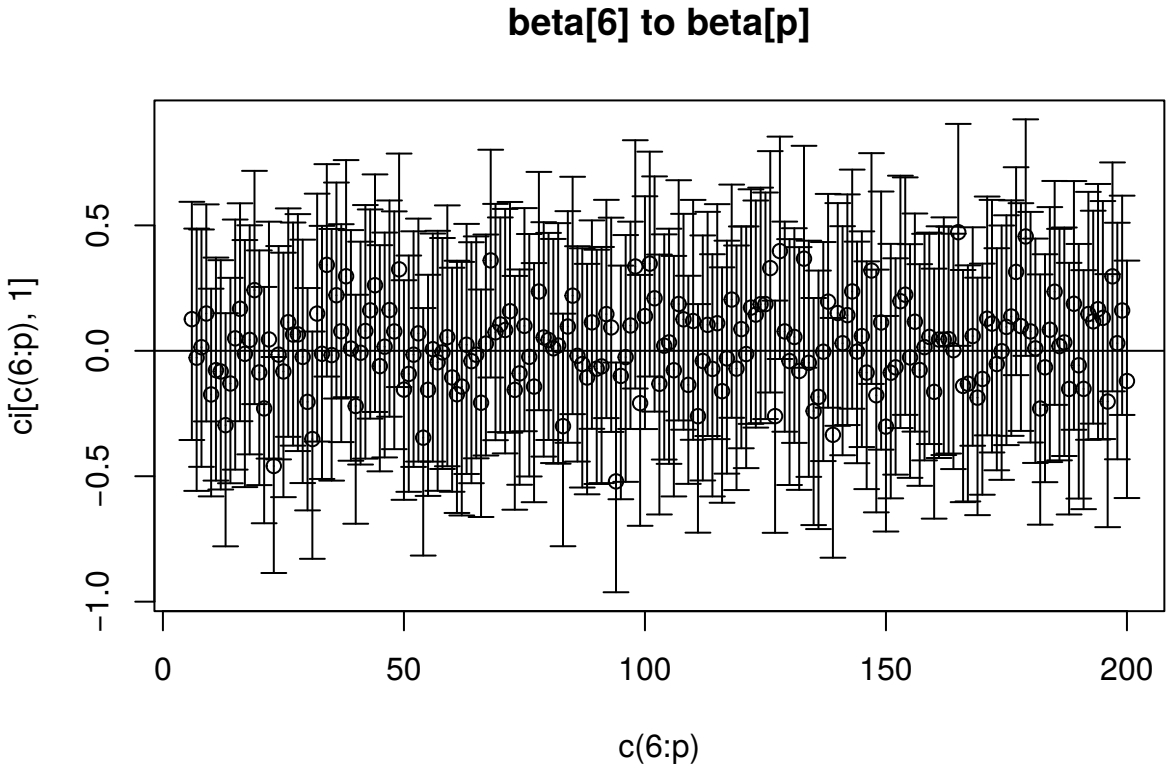


```
i <- 2
n <- 100
p <- 200
fit.jag <- run.jags(p,n)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 202
##    Total graph size: 21421
##
## Initializing model
```

```
plot.jag(fit.jag)
```

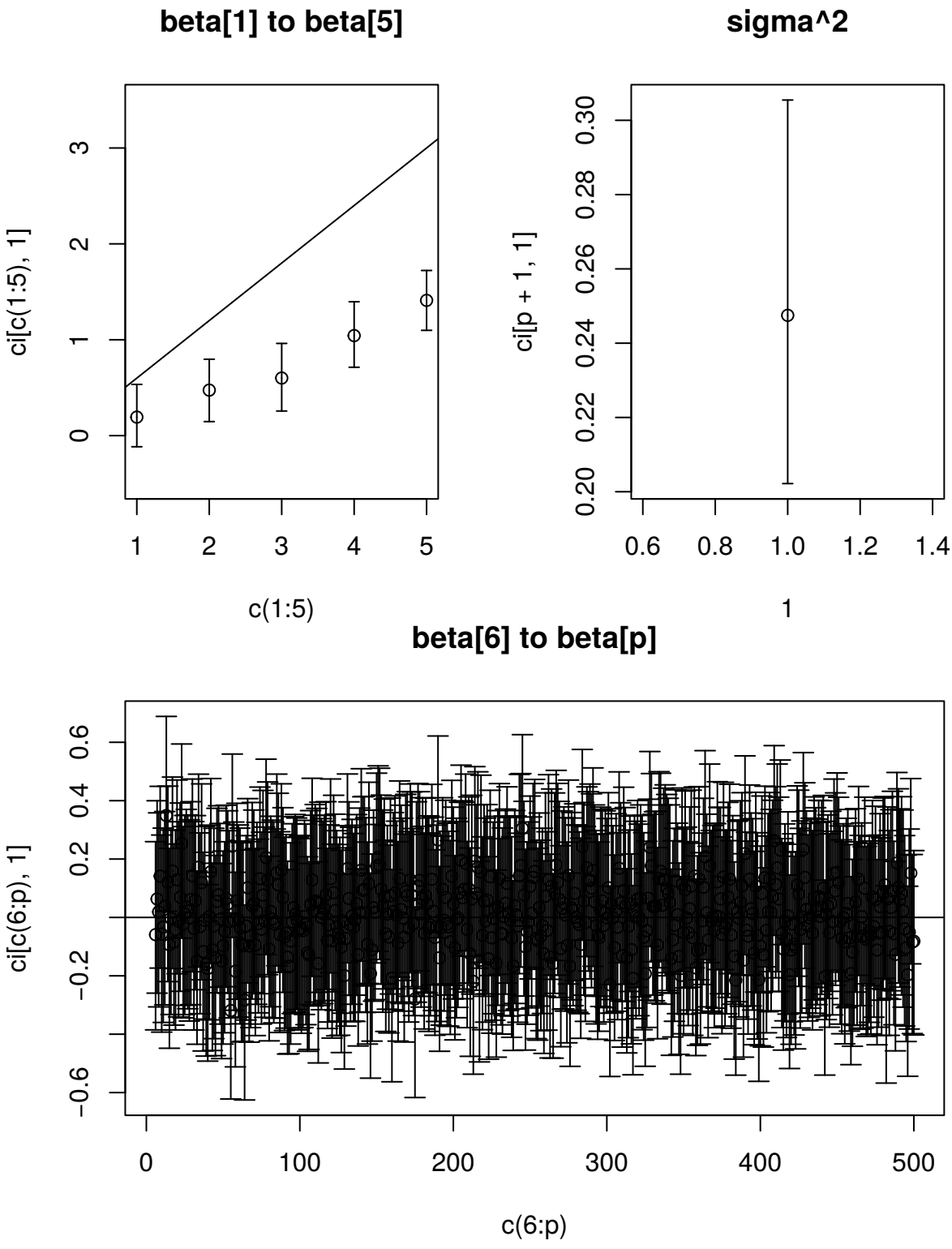





```
i <- 3
n <- 200
p <- 500
fit.jag <- run.jags(p,n)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 200
##    Unobserved stochastic nodes: 502
##    Total graph size: 103321
##
## Initializing model
```

```
plot.jag(fit.jag)
```


beta[1] to beta[5]
ci[c(1:5), 1]
c(1:5)
sigma^2
ci[p + 1, 1]
1
beta[6] to beta[p]
ci[c(6:p), 1]
c(6:p)