Computational Statistics

Lab 4

Emil K Svensson and Rasmus Holm

2017-03-01

Question 1

In this task we are going to generate samples from the distribution with probability density function

$$f(x) \propto x^5 e^{-x}, x > 0$$

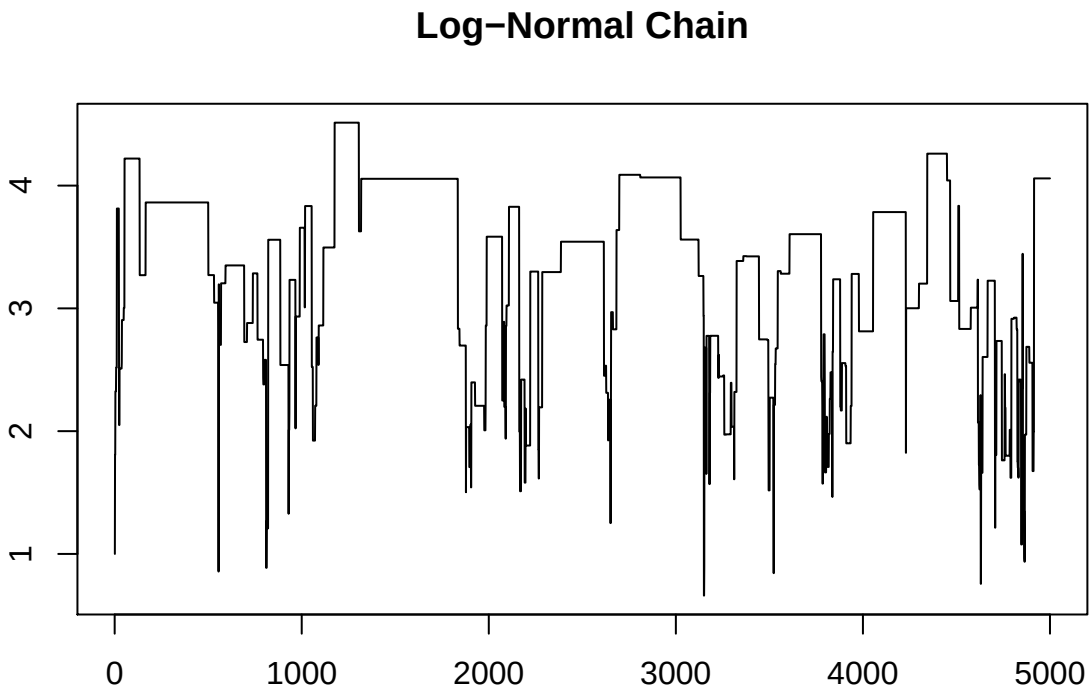
using the Metropolis-Hastings algorithm. We will be trying two different proposal distributions, namely the log-normal and chi-squared. We picked 1 to be the starting point and ran the algorithm for 5000 iterations in each case.

```
targetdensity <- function(x) {
    x^5 * exp(-x)
}

lognormalfuncs <- list(propsample=function(x) { rlnorm(1, meanlog=x, sdlog=1) },
                       propdensity=function(x, y) { dlnorm(x, meanlog=y, sdlog=1) },
                       targdensity=targetdensity)

chisquarefuncs <- list(propsample=function(x) { rchisq(1, df=floor(x + 1)) },
                       propdensity=function(x, y) { dchisq(x, df=floor(y + 1)) },
                       targdensity=targetdensity)

metropolis_hastings <- function(X0, iters, funcs) {
    x <- X0
    values <- rep(0, iters)

    alpha <- function(x, y) {
        numerator <- funcs$targdensity(y) * funcs$propdensity(x, y)
        denominator <- funcs$targdensity(x) * funcs$propdensity(y, x)
        numerator / denominator
    }

    for (i in 1:iters) {
        y <- funcs$propsample(x)
        u <- runif(1)

        if (u < alpha(x, y)) {
            x <- y
        }

        values[i] <- x
    }

    values
```

```
}

iters <- 5000
X0 <- 1

actual <- rgamma(iters, shape=6, rate=1)
```

1.1

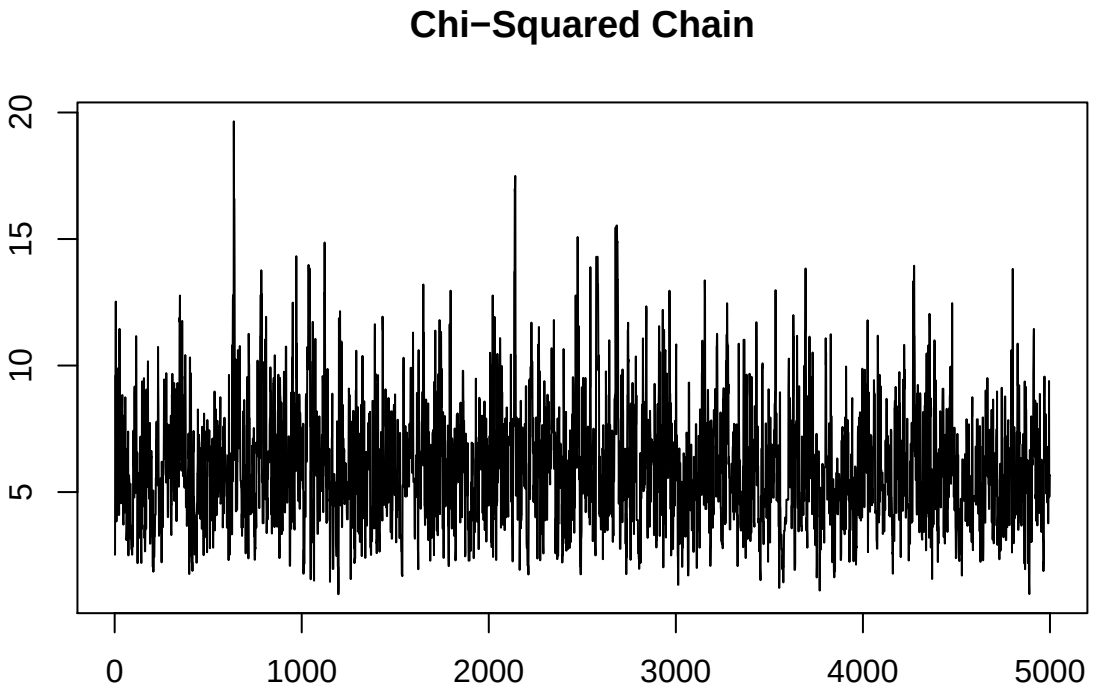
```
set.seed(123456)
lgnsamples <- metropolis_hastings(X0=X0, iters=iters, funcs=lognormalfuncs)
plot(lgnsamples, type="l", main="Log-Normal Chain", xlab="", ylab="")
```



The time-series graph seems to be far from stationary. There is no constant mean and the variation fluctuates over the time-series. For several iterations it keeps the same value before changing to another value that it repeats for some iterations, this pattern repeats for the whole time-series. This time series has not converged and seems to be far from convergence as well. There seem to be some sort of burn in period for the first 10 iterations.

1.2

```
set.seed(123456)
chisamples <- metropolis_hastings(X0=X0, iters=iters, funcs=chisquarefuncs)
plot(chisamples, type="l", main="Chi-Squared Chain", xlab="", ylab="")
```

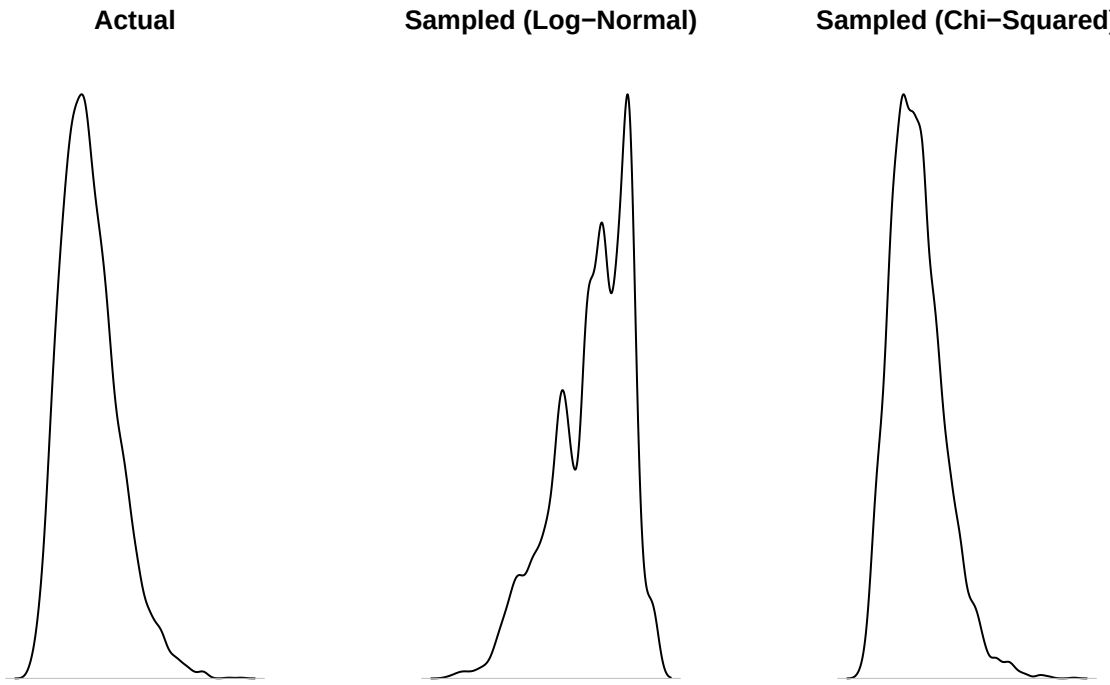


Here the time series looks of trend and seems to have one global mean over the time series. The variation also seems constant although if strict notes could be made that the variation seems marginally higher in the last 200 iterations. We consider that this time series has converged. A very small burn in period can be observed that last for about 2-3 iterations.

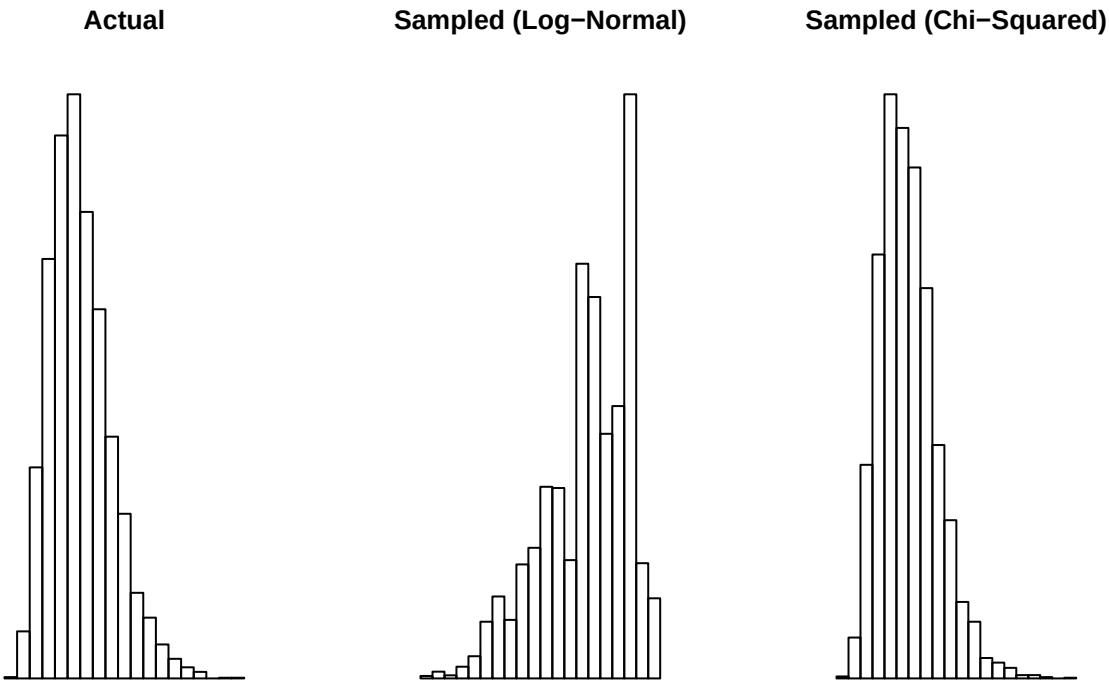
Overall it is clear that the chi-square distribution is a more appropriate proposal than the log-normal distribution.

1.4

```
oldpar <- par(mfrow = c(1, 3))

plot(density(actual), main="Actual", xlab="", ylab="", axes=FALSE)
plot(density(lgnsamples), main="Sampled (Log-Normal)", xlab="", ylab="", axes=FALSE)
plot(density(chisamples), main="Sampled (Chi-Squared)", xlab="", ylab="", axes=FALSE)
```




```
oldpar <- par(mfrow = c(1, 3))

hist(actual, main="Actual", xlab="", ylab="", axes=FALSE)
hist(lgnsamples, main="Sampled (Log-Normal)", xlab="", ylab="", axes=FALSE)
hist(chisamples, main="Sampled (Chi-Squared)", xlab="", ylab="", axes=FALSE)
```



Above are three histogram presented, the left being the actual values from the function, the middle one MC-sampling from the log-normal as the proposal distribution and the right being the MC-sampling from the chi-square as proposal distribution.

The chi-square samples look very similar to the actual values whereas the log-normal samples are far from

resembling the actual distribution. This support our previous interpretation that the chi-square is a better proposal distribution for this specific function.

Gelman-Rubin

```
gelman_rubin <-function(x){
  k <- nrow(x)
  n <- ncol(x)

  B <- (n / (k - 1)) * sum((rowMeans(x) - mean(x))^2)

  W <- sum((x - rowMeans(x))^2) / (k * (n - 1))

  VarV <- ((n - 1) / n) * W + B / n

  sqrtR <- sqrt(VarV / W)
  sqrtR
}

set.seed(123456)
resultMatrix <- do.call(rbind, lapply(1:10, FUN = function(x)
    metropolis_hastings(X0 = x, iters = iters, funcs = chisquarefuncs)))

gelmanrubinres <- gelman_rubin(resultMatrix)
gelmanrubinres
```

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

We get a value of 1 from the Gelman-Rubin algorithm which indicates that the chain has converged.

1.5

```
cat("The estimated expected value by Log-Normal samples:", mean(lgnsamples))
```

```
## The estimated expected value by Log-Normal samples: 3.35
```

```
cat("The estimated expected value by Chi-Squared samples:", mean(chisamples))
```

```
## The estimated expected value by Chi-Squared samples: 5.98
```

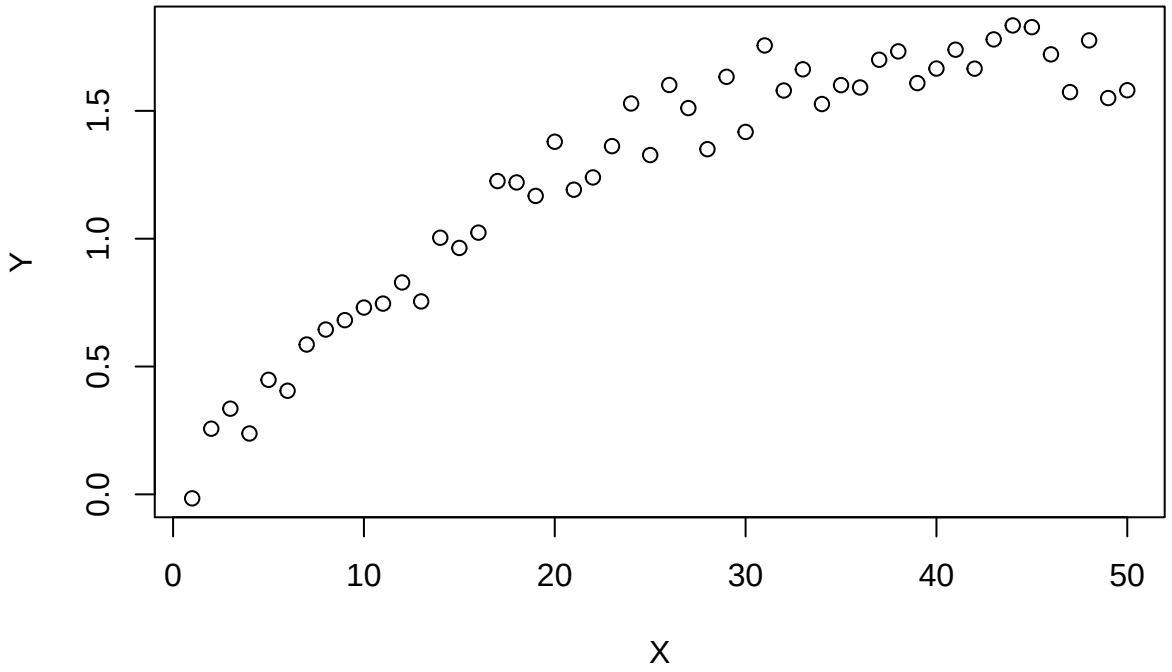
1.6

Since we know that the actual distribution is gamma with $\alpha = 6, \beta = 1$, we also know that the expected value is defined as $\frac{\alpha}{\beta} = 6$. So we can see that the samples generated from metropolis-hasting algorithm using the chi-squared distribution as proposal have a very close estimate while using log-normal distribution does not.

Question 2

We have been given noisy data and the purpose of this task is to restore the expected values with a Gibbs sampler using a Bayesian model introduced in 2.2-2.3.

```
load("../data/chemical.RData")

chem <- data.frame(X = X , Y = Y)

plot(chem)
```



We would think that a quadratic model would be appropriate to use to estimate y given x.

2.2

We have

$$\begin{aligned} p(\mu) &= p(\mu_1)p(\mu_2|\mu_1)\cdots p(\mu_n|\mu_{n-1}) \\ &= \frac{1}{\sqrt{(2\pi\sigma^2)^{n-1}}}\exp\left(-\frac{1}{2\sigma^2}\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2\right), \\ p(y|\mu) &= p(y_1|\mu_1)p(y_2|\mu_2)\cdots p(y_n|\mu_n) \\ &= \frac{1}{\sqrt{(2\pi\sigma^2)^{n}}}\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu_i)^2\right). \end{aligned}$$

2.3

The prosterior is then

$$
\begin{aligned}
p(\mu|y) &\propto p(y|\mu)p(\mu) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left(\sum_{i=1}^{n}(y_i-\mu_i)^2+\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2\right)\right) \\
&\propto \exp\left(-\frac{2}{2\sigma^2}(\mu_1-(y_1+\mu_2)/2)^2-\frac{3}{2\sigma^2}\sum_{j=2}^{i}(\mu_j-(y_j+\mu_{j-1}+\mu_{j+1})/3)^2-\frac{2}{2\sigma^2}(\mu_n-(y_n+\mu_{n-1})/2)^2\right).
\end{aligned}
$$

This gives us

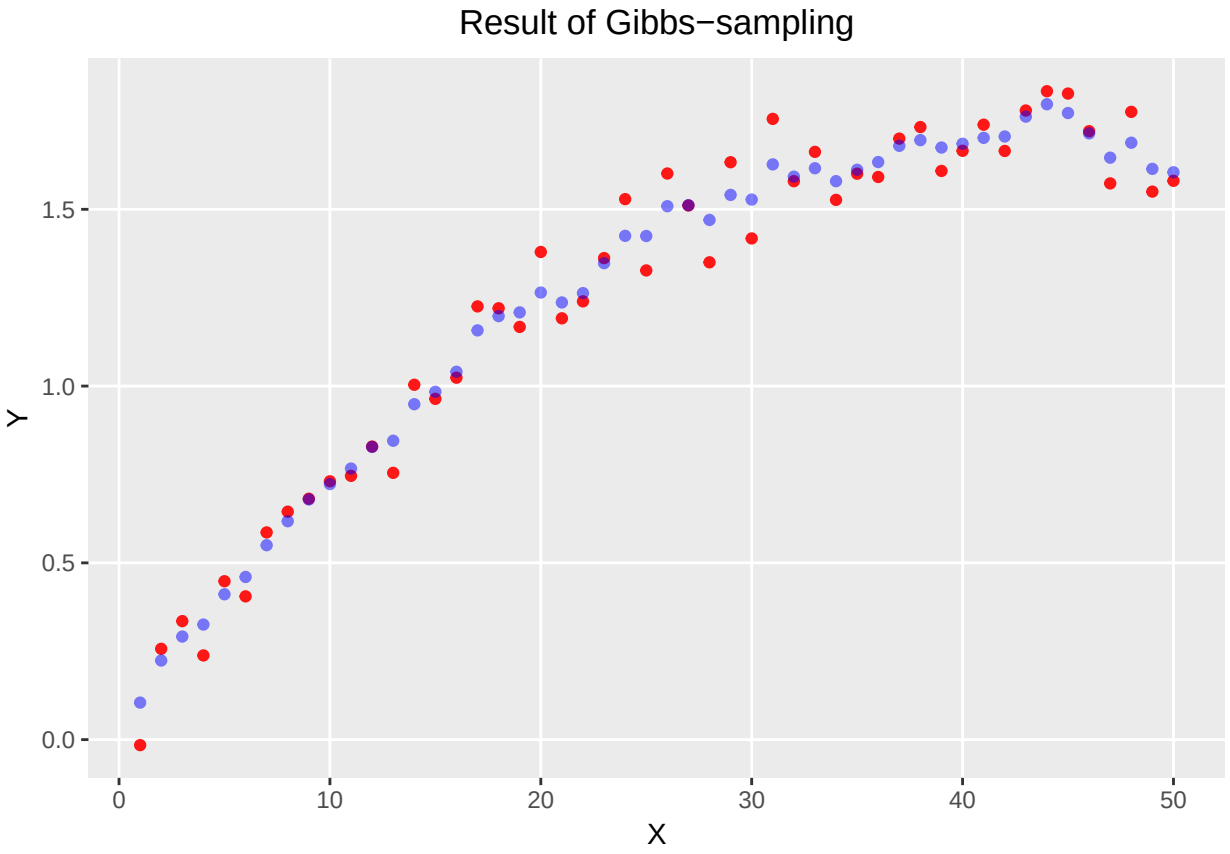
$$
\begin{aligned}
p(\mu_1|\mu_{-1},y) &\propto \exp\left(-\frac{2}{2\sigma^2}(\mu_1-(y_1+\mu_2)/2)^2\right), \\
p(\mu_i|\mu_{-i},y) &\propto \exp\left(-\frac{3}{2\sigma^2}(\mu_i-(y_i+\mu_{i-1}+\mu_{i+1})/3)^2\right) \text{ for } i=2,\ldots,n-1, \\
p(\mu_n|\mu_{-n},y) &\propto \exp\left(-\frac{2}{2\sigma^2}(\mu_n-(y_n+\mu_{n-1})/2)^2\right).
\end{aligned}
$$

2.4

```
posterior <- function(data, mus, index, sigmasq){
    if (index == 1){
        return(rnorm(1,
                     mean = (data[index] + mus[index + 1]) / 2,
                     sd = sqrt(sigmasq / 2)))
    }

    if (index == length(mus)){
        return(rnorm(1,
                     mean = (data[index] + mus[index - 1]) / 2,
                     sd = sqrt(sigmasq / 2)))
    }

    return(rnorm(1,
                 mean = (data[index] + mus[index - 1] + mus[index + 1]) / 3,
                 sd = sqrt(sigmasq / 3)))
}

gibbs <- function(data, tmax){
    d <- nrow(data)
    t <- 0

    mus <- matrix(0, nrow = tmax, ncol = d)
    sigmasq <- 0.2

    for (i in 1:tmax){
        for (j in 1:d){
            mus[i, j] <- posterior(data, mus[i, ], j, sigmasq)
        }
```

```
        if (i != tmax) {
            mus[i+1,] <- mus[i,]
        }
    }

    return(mus)
}
```

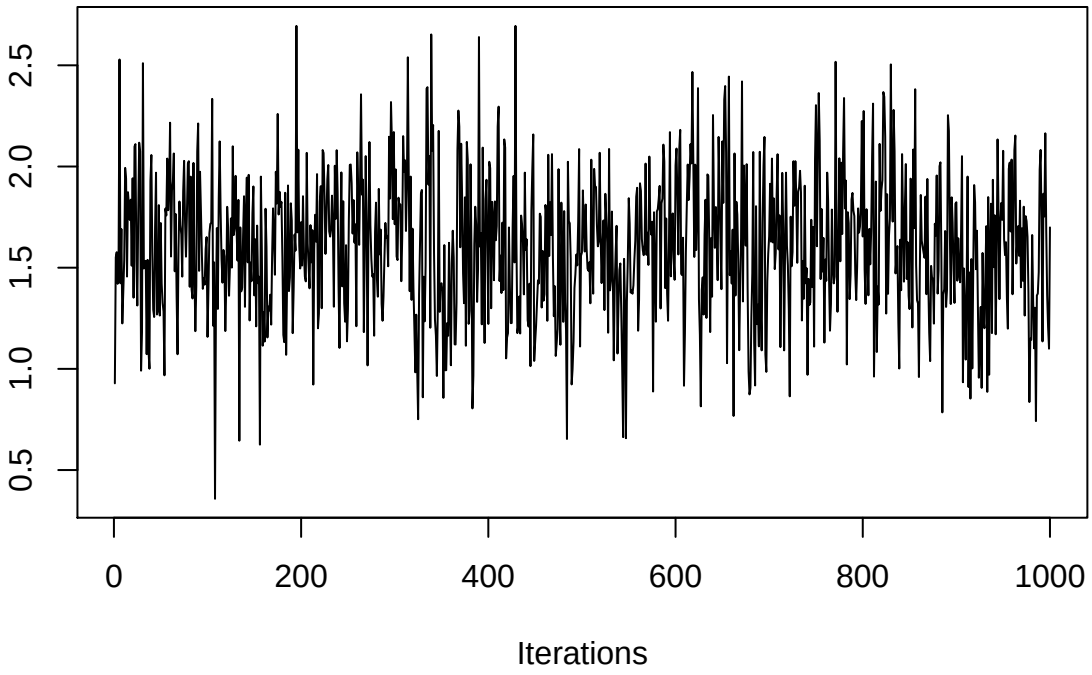
```
set.seed(123456)

d <- as.matrix(chem$Y)
mu <- gibbs(data = d, tmax = 1000)
emu <- colMeans(mu)

library(ggplot2)

ggplot(data = chem, aes(x = X, y = Y)) +
    geom_point(col = "red", alpha = 0.9) +
    geom_point(aes(y = emu), col ="blue", alpha = .5) +
    theme(panel.grid.minor = element_blank(),
          plot.title = element_text(hjust = 0.5)) +
    labs(title = "Result of Gibbs-sampling")
```



We can see that the estimated points (blue) by Gibbs sampling are more stable and do not vary as much between consecutive observations. It catches the relationship between x and y very well and the trend looks to follow a quadratic function just as we thought would be reasonable in the beginning.

2.5

```
library(coda)
traceplot(as.mcmc(mu[, 50]))
```



We can see that after roughly 600 iterations the value is more stable since it doesn't have those large spikes that occur in earlier iterations and the mean value seems to have stabilized.