Computational statistics - Lab 4

Alexander Karlsson (aleka769) Lennart Schilling(lensc874)

2019-02-19

Contents

Question 1: Computations with Metropolis-Hastings

Within this excercise we try to generate a sample from

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

using the Metropolis-Hastings algorithm.

1.1: Generating samples using log-normal distribution as proposal distribution

Since the log-normal function should be used as a proposal function, we first need the pdf of this distribution:

$$q(x|\mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(lnx-\mu)^2}{2\sigma^2}}$$

Specifically adjusted related to $LN(X_t, 1)$, the pdf of our proposal function is

$$q(X_t|\mu = X_t, \sigma = 1) = \frac{1}{X_t\sqrt{2\pi}} e^{-\frac{(lnX_t - X_t)^2}{2}}.$$

As it can be seen in the formula, the current value of X_t will be used as the mean argument of $q(x|\mu = X_t, \sigma = 1)$. Then, a sampled value of this distribution will be the candidate point Y. For reasons of convenience, the built-in-function $rlnorm()$ will be used to get the sampled value Y.

Together with the previous sampled value X_t, Y will be used as an input for the function $\alpha(X_t, Y)$. This will return a value which will be compared to a value U which will be sampled from $Unif(0, 1)$. If the returned value from $\alpha(X_t, Y)$ is larger than the value returned from $Unif(0, 1)$, Y will be accepted as a sampled value. Otherwise, again the previous sampled value X_t will be chosen as a sampled value.

Considering that the proposal function $q(X_t)$ is not symmetric, the function $\alpha(X_t, Y)$ results as follows:

$$\alpha(X_t, Y) = min\left\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\right\}$$

Since we know $\pi(x)$ which is the original distribution $f(x) \propto x^5 e^{-x}, x > 0$ we want to sample from and we also know the proposal function $q(x)$, we easily can calculate the returned value of $\alpha(X_t, Y)$.

Based on the fact that the Metropolis-Hastings algorithm will be run again within the next steps using different settings, a function will be defined.

```
mh_algorithm = function(pdf_original, pdf_proposal, sample_f_proposal, x0, n) {
  # Setting up starting point.
  x = x0
  # Setting up vector of collected samples.
  samples = numeric()
  # Simulating n times.
  for (i in 1:n) {
    # Sampling candidate point y from proposal function.
    y = sample_f_proposal(x)
    # Generating u from Unif(0,1).
    u = runif(1)
    # Generating return value of alpha function.
    alpha = min(1,
                (pdf_original(y) * pdf_proposal(x, y)) /
                  (pdf_original(x) * pdf_proposal(y, x)))
    # Comparing alpha and u to decide about acceptance / rejection.
```

```
    # Adding simulated sample to vector of collected samples.
    if (alpha > u) {
      samples[i] = y
      x = y
    } else {
      samples[i] = x
    }
  }
  # Returning vector of collected samples.
  return(samples)
}
```

For the simulation, the starting point will be set to 2. The algorithm will be run 10,000 times.

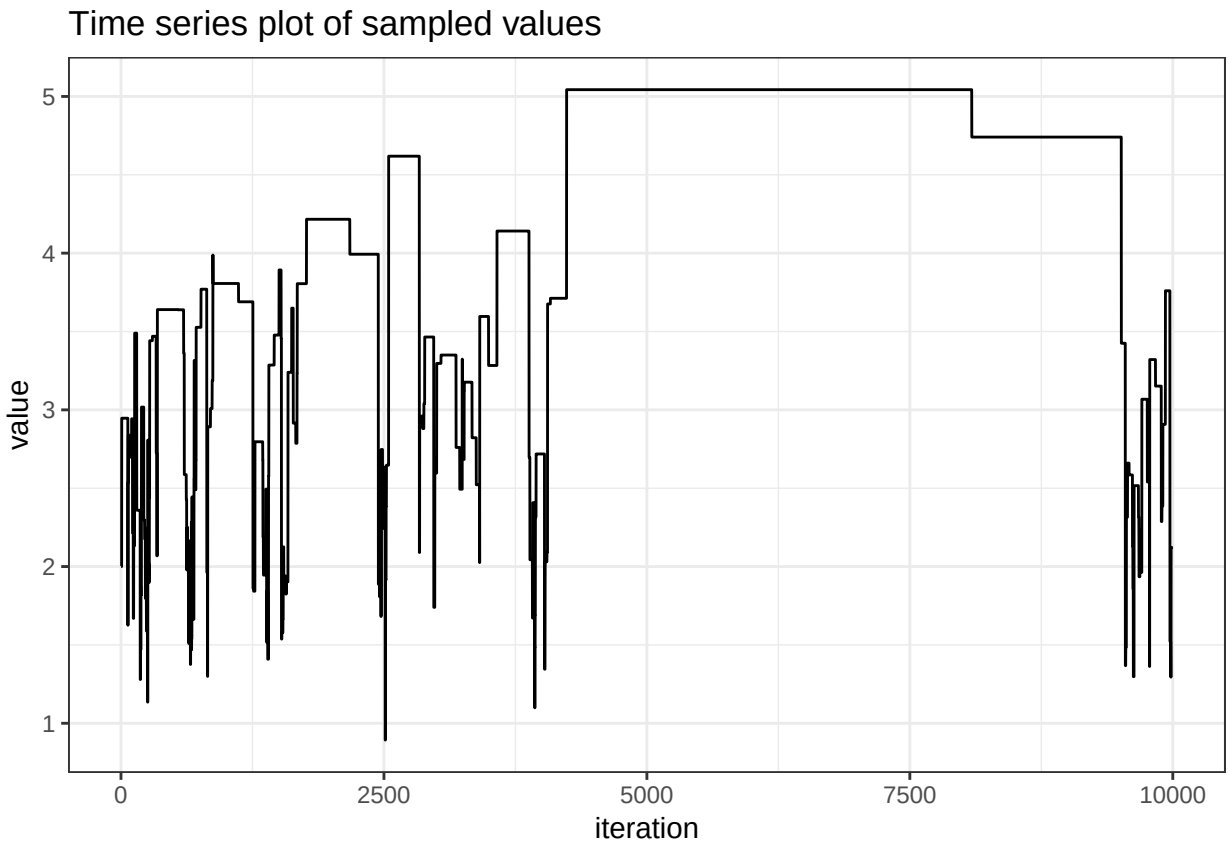
```
starting_point = 2
iterations = 10000
```

Using the orginal distribution $f(x)$, the proposal density function $q(x)$, the sampling function of the proposal distribution, the starting point and the number of iterations, the Metropolis-Hastings algorithm returns the simulated samples for $f(x)$.

```
# Setting seed.
set.seed(12345)
# Performing mh algortihm.
samples_log_norm = mh_algorithm(
  # Given density function f(x) will be defined as pdf_original.
  pdf_original = function(x) x^5*exp(-x),
  # Pdf of defined log-normal distribution q(x) will be used for pdf_proposal.
  pdf_proposal = function(x, mu, sigma = 1) {
    return((1/(x*sigma*sqrt(2*pi)))*exp(1)^(-((log(x)-mu)^2)/2*sigma^2))
  },
  # Candidate point y will be sampled from proposal function with mean = x and sd = 1
  # using rlnorm().
  sample_f_proposal = function(x) rlnorm(n = 1, meanlog = x),
  # Starting point 2 will be chosen.
  x0 = starting_point,
  # Number of iterations will be set to 10000.
  n = iterations)
```

The obtained chain will be plotted.

```
library(ggplot2)
plotting_samples = function(collected_samples) {
  plot_data = data.frame(iteration = 1:length(collected_samples),
                         value = collected_samples)
  ggplot(data = plot_data,
         aes(x = iteration, y = value)) +
    geom_line() +
    theme_bw() +
    ggtitle("Time series plot of sampled values")
}
plotting_samples(samples_log_norm)
```



Since the time series of the simulated chain does not follow a pattern but instead is very irregular, it does not seem to converge. Furthermore, there is no burn-in-period to observe since the first simulated sampled values do not increase slowly but are directly similar to the other samples.

1.2: Generating samples using chi-square distribution as proposal distribution

Since the chi-square distribution $x \sim \chi^2(\lfloor X_t + 1 \rfloor)$ should be used as a proposal function, we first need the pdf of this distribution:

$$q(x|k) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} e^{-x/2}$$

Specifically adjusted related to $\chi^2(\lfloor X_t + 1 \rfloor)$, the pdf of our proposal function is

$$q(X_t|\lfloor X_t + 1 \rfloor) = \frac{1}{2^{\lfloor X_t+1 \rfloor/2}\Gamma(\lfloor X_t + 1 \rfloor/2)} x^{\lfloor X_t+1 \rfloor/2-1} e^{-X_t/2}.$$

For reasons of convenience, the function will not be implemented on our own within the following code. Instead, the built-in-functions *dchisq*() and *floor*() will be used to obtain the pdf value of X_t so that the alpha value can be calculated.

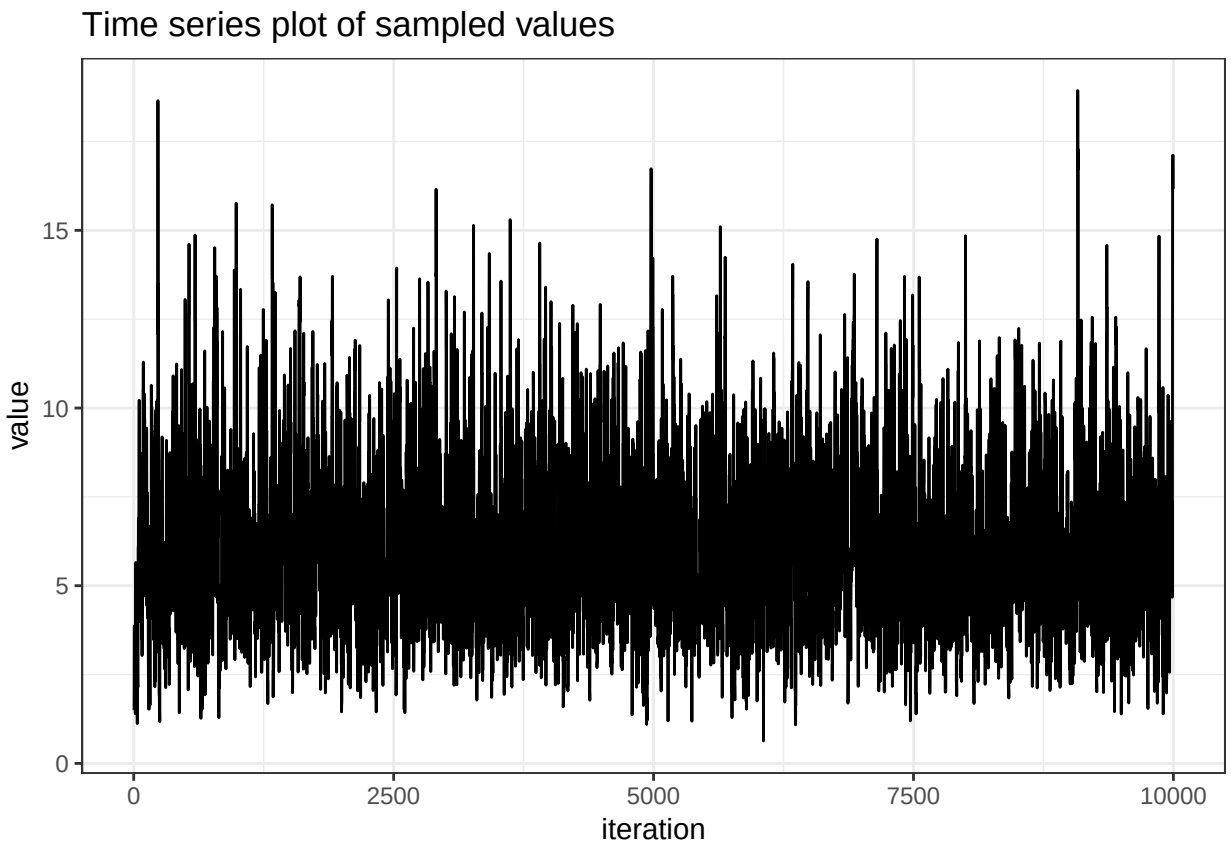
The created function from step 1 can be used to generate the samples. The starting point (2) and number of iterations (10,000) are kept from the previous simulation.

```
# Performing mh algortihm.
samples_chi_squared = mh_algorithm(
  # Given density function f(x) will be defined as pdf_original.
  pdf_original = function(x) x^5*exp(-x),
  # Pdf of defined chi-square distribution q(x) will be used for pdf_proposal
```

```
  # using dchisq().
  pdf_proposal = function(x, y) dchisq(x = x, df = floor(y+1)),
  # Candidate point y will be sampled from proposal function with df = floor(x+1)
  # using rchisq().
  sample_f_proposal = function(x) rchisq(n = 1, df = floor(x+1)),
  # Starting point 2 will be chosen.
  x0 = starting_point,
  # Number of iterations will be set to 10000.
  n = iterations)
```

Again, the obtained chain will be plotted.

```
plotting_samples(samples_chi_squared)
```



1.3: Comparing obtained chains

Comparing the two plots, it is obvious that with the choice of the chi-square distribution as a proposal function, the chain converges in contrast to the chain generated by using the log-normal distribution as a proposal function. Visually, it seems as if the second chain is stationary and that the chain does converge to this stationary distribution. The burn-in-period cannot defined by looking at this plot since the chain convergese very fast. Consequently, the simulation usign the chi-square distribution as a proposal function seems to deliver an reasonable result and is therefore the better choice.

1.4: Analysing convergence using 10 MCMC sequences

Ten MCMC sequences using the generator from step 2 and starting points 1-10 are created and stored in a `mcmc.list()`-object.

```r
library(coda)
my_mcmc = mcmc.list()
for (k in 1:10) {
  # Performing mh algortihm.
  samples_chi_squared = mh_algorithm(
    # Given density function f(x) will be defined as pdf_original.
    pdf_original = function(x) x^5*exp(-x),
    # Pdf of defined chi-square distribution q(x) will be used for pdf_proposal
    # using dchisq().
    pdf_proposal = function(x, y) dchisq(x = x, df = floor(y+1)),
    # Candidate point y will be sampled from proposal function with df = floor(x+1)
    # using rchisq().
    sample_f_proposal = function(x) rchisq(n = 1, df = floor(x+1)),
    # Starting point 2 will be chosen.
    x0 = k,
    # Number of iterations will be set to 10000.
    n = iterations)
  # Storing samples as mcmc objects in my_mcmc
  my_mcmc[[k]] = as.mcmc(samples_chi_squared)
}
```

Using this list of different sequences, the *Gelman-Rubin factor* can be calculated.

```r
gelman.diag(my_mcmc)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

Since both values the point estiamtor and also the upper confidence interval show a value of 1 it can be concluded that convergence has been reached.

1.5: Estimating integral

For the sample from the first step (log-normal distribution as proposal distribution), the integral can be estimated by:

$$\frac{1}{n}\sum_{t=1}^{n}(X_t)$$

As a result, the estiamted integral is 4.2216864.

In contrast, for the sample obtained within the second step (chi-square distribution as proposal distribution), the first $k-1$ samples should be removed from the collected samples which refer to the *burn-in period*. As described in 1.4, k cannot be identified visually with the help of the plot, since the chain converges too fast. As a result, the first 1% (100 samples) of the samples are removed.

The modified formula for the integral estimation is

$$\frac{1}{n-100}\sum_{t=100+1}^{n}(X_t)$$

As a result, the estiamted integral from the samples from the second step is 5.9209507.

1.6: Comparing obtained to actual integral

The pdf of a Gamma distribution is

$$\frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

Since $\frac{\beta^{\alpha}}{\Gamma(\alpha)}$ is a constant, $x^{\alpha-1}e^{-\beta x}$. can be kept. Compared to the target distribution $f(x) \propto x^5 e^{-x}, x > 0$ it follows that $\alpha = 6$ and $\beta = 1$.

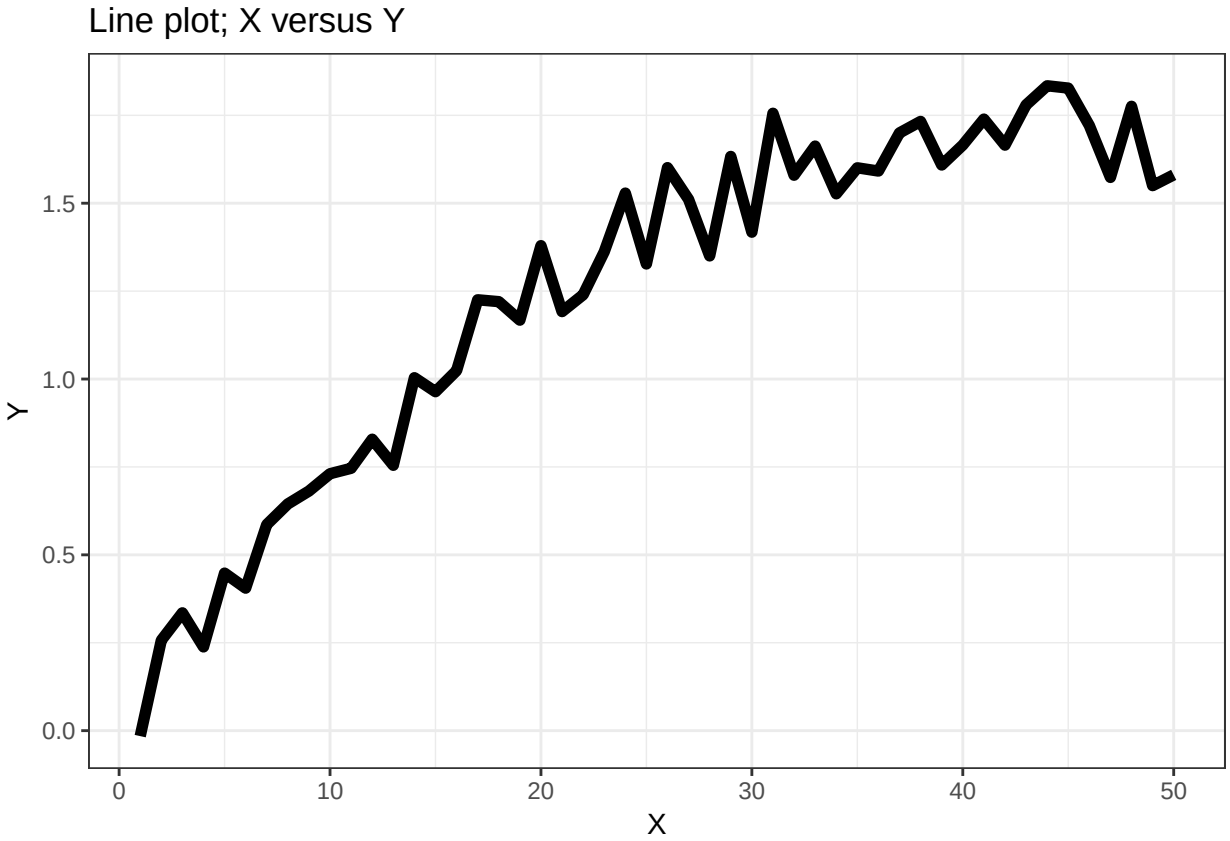
Since the mean of a Gamma distribution is $E(X) = \frac{\alpha}{\beta}$, the actual integral is 6.

Finally, it confirms that the solution from step 2 (chi-square distribution as proposal distribution) is the better choice because the estimated integral is very close to the actual integral.

Question 2: Gibbs sampling

2.1: Plot of data

```
library(dplyr)
library(reshape2)
library(ggplot2)
load("chemical.Rdata")

theme_set(theme_bw())

data.frame(X, Y) %>% ggplot(., aes(X, Y)) + geom_line(size = 2) +
  labs(title = "Line plot; X versus Y")
```



2.2: Prior and likelihood

In this assignment we skip the first fraction of the normal distribution: $k = \frac{1}{\sqrt{2\pi\sigma^2}}$ and make use of the proportianality properties.

The prior beliefs is that the measurement value of one day is dependent of the measurement value on the previous day. These priors can be rewritten as noral distribution expressions:

$$
\begin{aligned}
p(\mu_1) &= 1 \\
p(\mu_2) &= p(\mu_2|\mu_1) && \propto e^{\frac{1}{2\sigma^2}(\mu_2-\mu_1)^2} \\
&\vdots \\
p(\mu_n) &= p(\mu_n|\mu_{n-1}) && \propto e^{\frac{1}{2\sigma^2}(\mu_n-\mu_{n-1})^2},
\end{aligned}
$$

The product of all these leads to the following result:

$$
\begin{aligned}
p(\vec{\mu}) = \prod_{i=1}^{n} p(\mu_i) \propto \left(1\right) \cdot \left(e^{-\frac{1}{2\sigma^2}(\mu_2-\mu_1)^2}\right) \cdot \left(e^{-\frac{1}{2\sigma^2}(\mu_3-\mu_2)^2}\right) \cdot \ldots \cdot \left(e^{-\frac{1}{2\sigma^2}(\mu_n-\mu_{n-1})^2}\right) = \\
= 1 \cdot e^{-\frac{1}{2\sigma^2}\sum_{i=2}^{n}(\mu_i-\mu_{i-1})^2}
\end{aligned}
$$

And that's the priors defined. We continue with definitions of the likelihood, each one at a time to the find a general expression:

$$
\begin{aligned}
p(\vec{Y}|\mu_1) &\propto e^{-\frac{1}{2\sigma^2}(y_1-\mu_1)^2} \\
p(\vec{Y}|\mu_2) &\propto e^{-\frac{1}{2\sigma^2}(y_2-\mu_2)^2} \\
&\vdots \\
p(\vec{Y}|\mu_n) &\propto e^{-\frac{1}{2\sigma^2}(y_n-\mu_n)^2}
\end{aligned}
$$

The product can be written as:

$$
p(\vec{Y}|\vec{\mu}) = \prod_{i=1}^{n} p(y_i|\mu_i) \propto e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu_i)^2}
$$

We write them again, just for the purpose of comparing these expressions to find a general formula.

$$
\begin{aligned}
p(\vec{Y}|\vec{\mu}) &\propto \left(e^{-\frac{1}{2\sigma^2}(\mu_1-y_1)^2}\right)\left(e^{-\frac{1}{2\sigma^2}(\mu_2-y_2)^2}\right) \cdot \ldots \cdot \left(e^{-\frac{1}{2\sigma^2}(\mu_n-y_n)^2}\right) \\
p(\vec{\mu}) &\propto \left(1\right)\left(e^{-\frac{1}{2\sigma^2}(\mu_2-\mu_1)^2}\right) \cdot \ldots \cdot \left(e^{-\frac{1}{2\sigma^2}(\mu_n-\mu_{n-1})^2}\right)
\end{aligned}
$$

These are to be combined below...

2.3: Posterior

The posterior in any bayes model can be calculated with bayes theorem, a vectorized solution is simply the product of several indices:

$$
p(\vec{\mu}|\vec{Y}) = \prod p(\vec{Y}|\vec{\mu}) \cdot p(\vec{\mu}) = \left(e^{-\frac{1}{2\sigma^2}[(-\mu_1+y_1)^2+\ldots+(-\mu_n+y_n)^2]}\right) \cdot \left((1) \cdot e^{-\frac{1}{2\sigma^2}[(-\mu_1+\mu_2)^2+\ldots+(-\mu_{n-1}+\mu_n)^2]}\right)
$$

We rewrite expressions from the priors and likelihoods into one exponent and make use of **Hint B**. By evaluating the expressions it is possible to see that the products (with exception when $i = 1$) are two normal distributions with different means. It is of interest to isolate μ_i on the left hand side in the exponent and adjust the exponent so that the denominator is $2\sigma^2$. From there, it is possible to use hint B once again to obtain a combined normal.

$$
\begin{aligned}
p(\mu_1|\vec{Y}) &\propto exp\left(\frac{1}{\sigma^2}(\mu_1 - (y_1 + \mu_2)/2)^2\right) = \mathbf{N}\left(\mathbf{\frac{1}{2}(y_1 + \mu_2)}, \mathbf{\frac{1}{2}\sigma^2}\right) \\
p(\mu_i|\vec{Y}) &\propto exp\left(\frac{1}{\sigma^2}(\mu_{i-1} - (y_{i-1} + \mu_i)/2)^2\right) \cdot exp\left(\frac{1}{\sigma^2}(\mu_i - (y_i + \mu_{i+1})/2)^2\right) = ... \propto \\
&\propto \mathbf{N}\left(\mathbf{\frac{1}{5}(2\mu_{i-1} - y_{i-1} + 2\mu_{i+1} + 2y_i)}, \mathbf{\frac{1}{10}\sigma^2}\right) \\
p(\mu_n|\vec{Y}) &\propto exp\left(\frac{1}{\sigma^2}(\mu_{n-1} - (y_{n-1} + \mu_n)/2)^2\right) \cdot exp\left(\frac{1}{2\sigma^2}(\mu_n - y_n)^2\right) \propto \\
&\propto \mathbf{N}\left(\mathbf{\frac{1}{3}(2\mu_{n-1} - y_{n-1} + 2y_n)}, \mathbf{\frac{1}{6}\sigma^2}\right)
\end{aligned}
$$

The expressions in **bold** are programmer-friendly expressions, as we can use the same syntax in R.

2.4: Gibbs sampler

With all formulas defined, it is possible to write code that will sample from the specific distributions derived above.

```
post_sample_fun = function(i, y, mu, sigma){
  if(i == 1){
    return(
      rnorm(n    = 1,
            mean = (y[i] + mu[i+1])/2,
            sd   = sqrt((1/2) * sigma)))
  } else if(i == length(y)){
    return(
      rnorm(n    = 1,
            mean = (2*mu[i-1] - y[i-1] + 2*y[i])/3,
            sd   = sqrt((1/6) * sigma)))
  } else (
    return(
      rnorm(n    = 1,
            mean = (2*mu[i-1] - y[i-1] + 2*mu[i+1] + 2*y[i])/5,
            sd   = sqrt((1/10) * sigma)))
  )
}
```

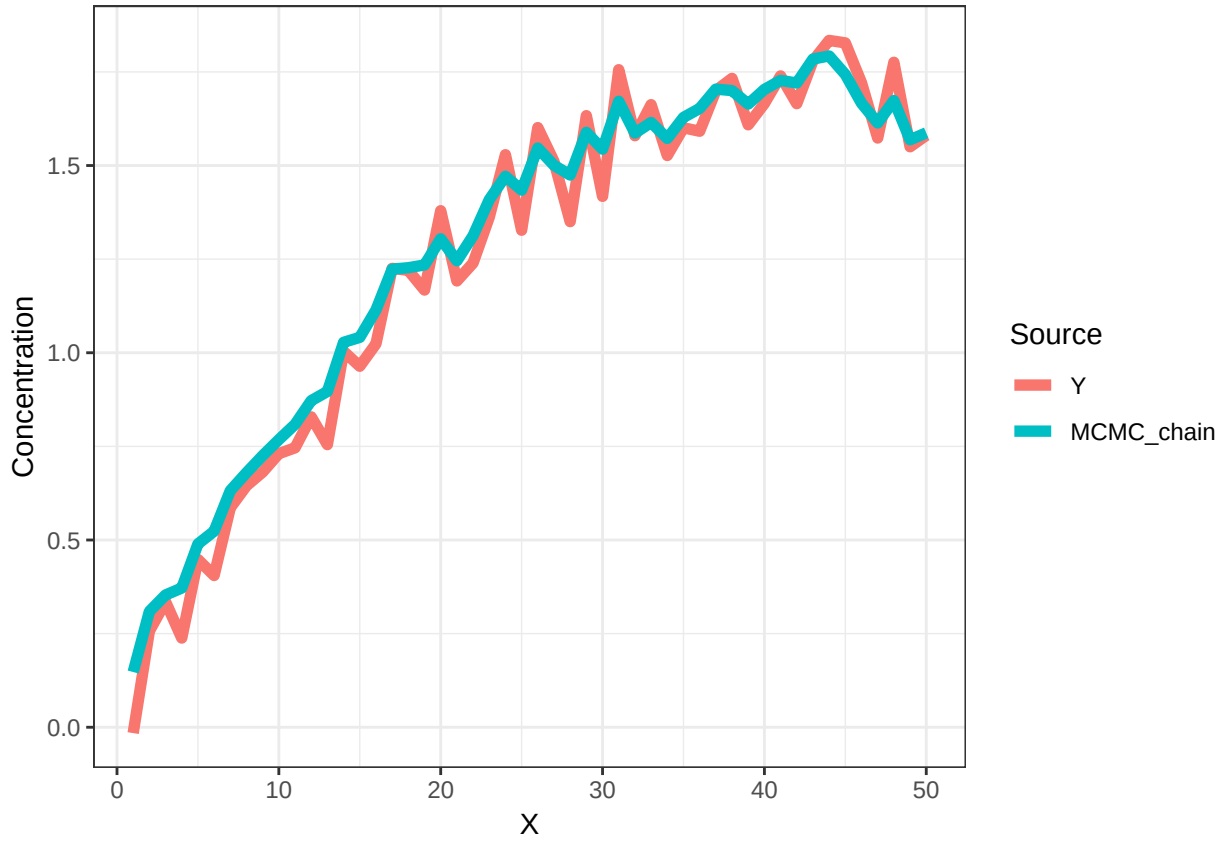
The function is defined with default arguments.

```
MCMC.Gibbs = function(n_iter = 1000, mean_vec = Y, var_ = 1/5, Plot, mu_0 = 0){
  #set-up:
  l        = length(mean_vec)
  mean_mat = matrix(data = 0,
                    nrow = n_iter,
                    ncol = l)
  mu_      = rep(x     = mu_0,
                 times = l)

  # sampling:
  for(i in 1:n_iter){
    for(j in 1:l){
      mu_[j] = post_sample_fun(i = j, y = mean_vec, mu = mu_, sigma = var_)
    }
```

```
    mean_mat[i,] = mu_
  }

  # return values:
  if(Plot == "xy"){
    colMeans(mean_mat)
  } else if (Plot == "trace"){
    mean_mat[,1]
  }
}
```

The values are plotted below.

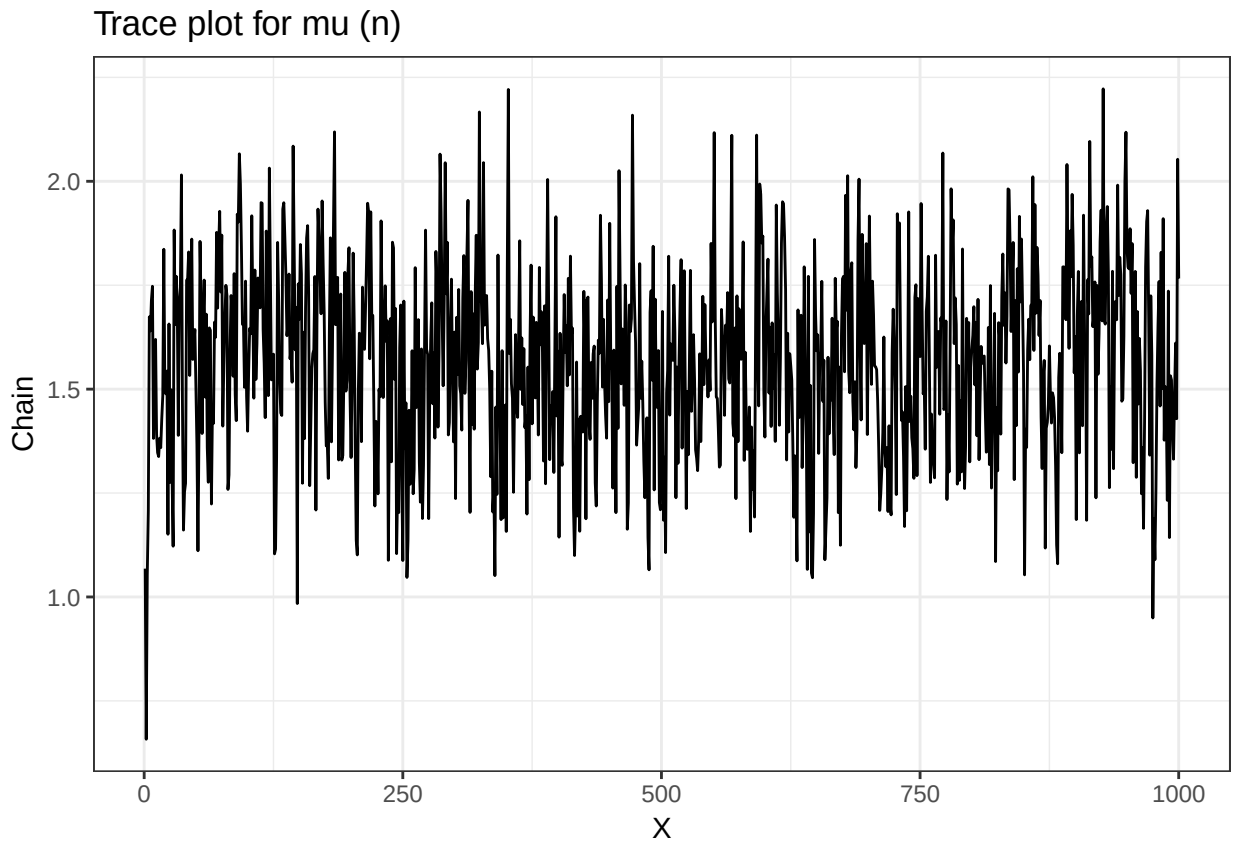
```
set.seed(123456)

data.frame("X" = X, "Y" = Y, "MCMC_chain" = MCMC.Gibbs(Plot = "xy")) %>%
  melt(., id.vars = "X", variable.name = "Source", value.name = "Concentration") %>%
  ggplot(., aes(X, Concentration, color = Source)) + geom_line(size =  2)
```



It seems as the noise is removed with this approach, as the MCMC line is more stable yet follows the trend of the original measurements. The true underlying might be just the blue plotted line, or not... It is hard to say, but if we assume that the error around the 'true value' indeed is normally distributed then this approach catches the underlying dependence very well, especially as X increases.

2.5: Trace plot

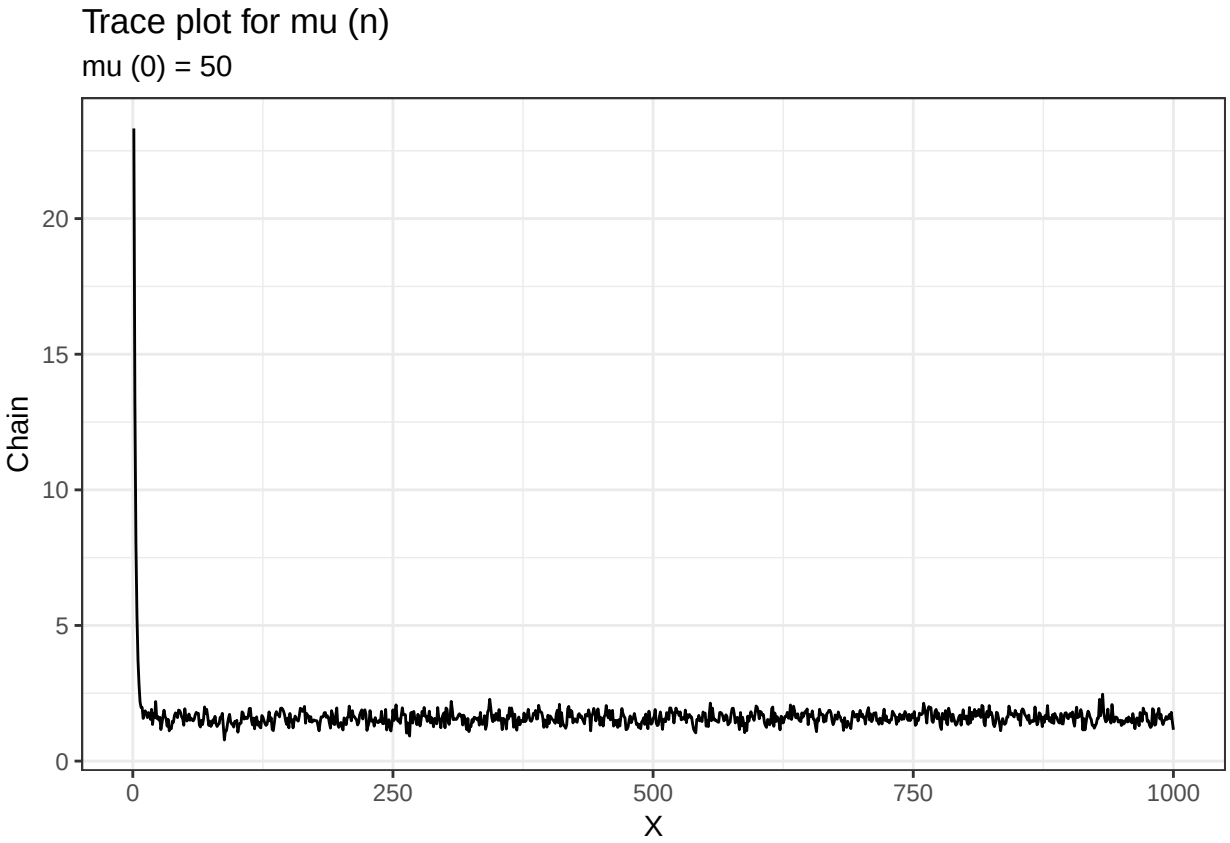
```
data.frame("Chain" = MCMC.Gibbs(Plot = "trace"),
           "X"     = 1:1000) %>%
  ggplot(., aes(X, Chain)) + geom_line() +
  labs(title = "Trace plot for mu (n)")
```



It only takes a couple of iterations before mu_n converges. The burn-in period is therefore almost non-existing. This phenomenon does not change with a different starting value either.

The plot below shows the trace plot for μ_n with a starting value of 50. and it still converges rapidly.

```
data.frame("Chain" = MCMC.Gibbs(n_iter = 1000, Plot = "trace", mu_0 = 50),
           "X"     = 1:1000) %>%
  ggplot(., aes(X, Chain)) + geom_line() +
  labs(title = "Trace plot for mu (n)",
       subtitle = "mu (0) = 50")
```


Trace plot for mu (n)
mu (0) = 50
Chain
20
15
10
5
0
0
250
500
750
1000
X

Appendix

All code is shown in the report.