Lab4

Andreas Stasinakis & Jesper Lindberg

February 6, 2019

Contents

Contributors: Jesper Lindberg & Andreas Stasinakis **1**

Question 1: Computations with Metropolis - Hastings(Jesper Lindberg) **2**
1.1 Metropolis Hastings, time series plot. 2
1.2 Same function for Chi -square as proposal distribution 4
1.3 Comparing the two samples 6
1.4 Convergence monitoring with Gelman - Rubin method. 6
1.5 Estimate an integral of a Gamma distribution 7
1.6 Gamma distribution, estimation of the integral and conclusions. 8

Question 2 Gibbs Sampling, Bayessian Theorem and Trace plots. **9**
2.1 Import(load) Rdata and plot the dependence of Y on X. 9
2.2 Bayessian model, Likelihood and Prior of Normal distriubtion. 9
2.3 Bayes Theorem, calculating the Posterior distribution 10
2.4 Gibbs sampler, Monte carlo approach and Bayes Theorem 12
2.5 Trace plot, burn in period and convergence. 14

Contributors: Jesper Lindberg & Andreas Stasinakis

Question 1: Computations with Metropolis - Hastings(Jesper Lindberg)

Consider the following probability density function:

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

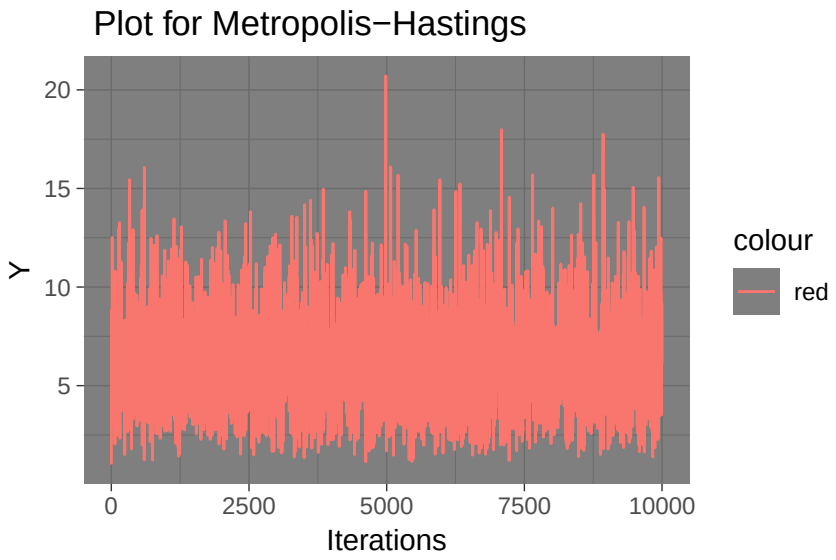
You can see that the distribution is known up to some constant of proportionality. If you are interested (NOT part of the Lab) this constant can be found by applying integration by parts multiple times and equals 120.

1.1 Metropolis Hastings, time series plot.

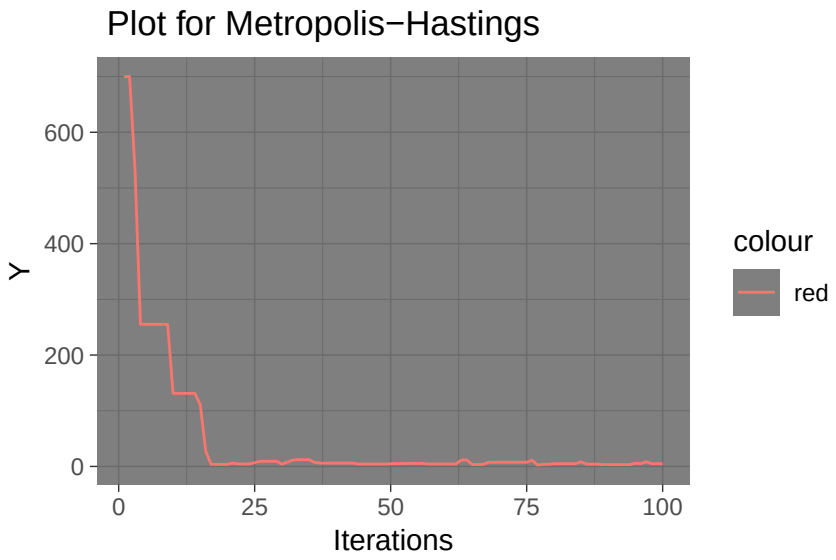
Use Metropolis-Hastings algorithm to generate samples from this distribution by using proposal distribution as log-normal $LN(Xt, 1)$, take some starting point. Plot the chain you obtained as a time series plot. What can you guess about the convergence of the chain? If there is a burn-in period, what can be the size of this period?

```
library(ggplot2)
# target pdf of the distribution we want to sample
target_pdf = function(x){
  return(x^5*exp(-x))
}

#function for random sample using Metropolis - Hastings
# target distribution is the pdf above
#Log normal distribution as a proposal distribution
# starting point  and number of steps

Metro_Hest = function(target, stat_point, steps){

  # vector to store the sample
  X = rep(stat_point,steps)

  # loop for all steps
  for (i in 2:steps) {

    # generate one random value from the proposal distribution
    # this value depends on the previous value
    Y = rlnorm(n = 1,meanlog = log(X[i-1]), sdlog = 1)

    # random value between 0 and 1
    u = runif(n = 1)

    #ratio for the target dist
    r1 = target(Y)/target(X[i-1])

    #ratio for the proposal distribution, conditional prob
    r2 = dlnorm(x = X[i-1] , meanlog = log(Y),
                sdlog = 1 )/dlnorm(x = Y, meanlog = log(X[i-1]), sdlog = 1 )

    # Calculate the acceptance probability
    # is a probability so have to be betwwen 0 and 1

    acceptance = min(1, r1*r2 )

    #the acceptance condition
```

```
    if (u<=acceptance) {
      X[i] = Y
    }else X[i] = X[i-1]
  }

  return(X)

}

set.seed(12345)
X_log  = Metro_Hest(target = target_pdf,stat_point = 1,steps = 10000)
df = data.frame(x = c(1:10000), y = X_log)

plot1 = ggplot(df) +
  geom_line(mapping = aes(x = df$x,y= df$y, color = "red"))+
  labs(title = " Plot for Metropolis-Hastings", x = "Iterations", y = "Y")+
  theme_dark()

plot1
```




```
# sample to comment the burn in period
# choose an "unusual" starting point

X_log2  = Metro_Hest(target = target_pdf,stat_point = 700,steps = 100)
df = data.frame(x = c(1:100), y = X_log2)

plot2 = ggplot(df) +
  geom_line(mapping = aes(x = df$x,y= df$y, color = "red"))+
  labs(title = " Plot for Metropolis-Hastings", x = "Iterations", y = "Y")+
  theme_dark()

plot2
```



In this task we implement the Metropolis-Hastings in order to generate samples from the above distribution using Markov Chains. We also need a proposal distribution which in this case is the Log - normal with logmean equal to a starting point and the log standard deviation equal to 1. Given a target pdf, a starting point and a number of iterations, a point Y from the proposal distribution is generated. In order to decide if we will keep this point to our sample we have to calculate an acceptance probability. This probability can be found from the formula above:

$$a(X_t, Y) = min\Big\{1, \frac{\pi(Y)q(X_t/Y)}{\pi(X_t)q(Y/X_t)}\Big\}$$

, where $\pi(X)$ is the *pdf* target distribution and q is the proposal. Moreover we generate a random number u between 0 and 1, and if $u < a$ then $X_{t+1} = Y$ otherwise $X_{t+1} = \text{X\_t}$. It is obvious that this method uses Markov Chains because each X_t depends only on the previous point X_{t-1}.

In general, a chain converges if after some iterations, the chain has "forgotten" its starting values and the outputs seem identical. So in order to comment the convergence of the chain we should take a look to the first plot above. In order for the chain to converge we need a stationery pattern and not a random walk pattern. So in this case that the chain converges after not that many iterations, with a mean value close to 6.

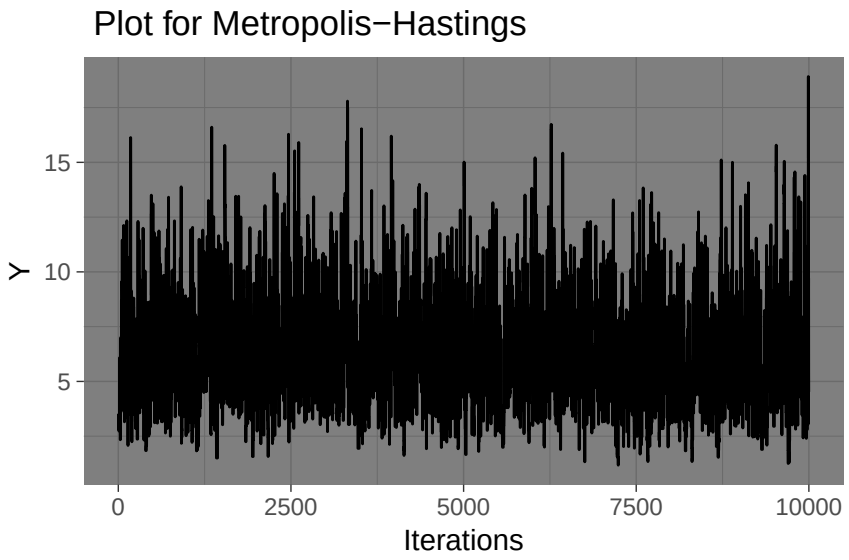
It is obvious that if we do not choose a "good" starting point, the metropolis algorithm may need some iterations in order to generate numbers which are representative of the distribution we want. Also the point of each iteration are highly dependent to the starting point. We can solve those two issues by just removing the n first observations, which called burn in period. The remaining part of the chain has converge in the distribution or at least has reached the stationery phase. In order to decide the n we sample with the a unusual starting point (500) and we plot that sample. It is clear that the algorithm find the correct path pretty soon despite the fact that the starting point is really far from the mean of the distribution. Although, it seems that we do not need to remove any point of the sample, we can choose $n = 15$ for the burn in period because we have to choose one.

1.2 Same function for Chi -square as proposal distribution

Perform Step 1 by using the chi-square distribution $X^2([X_t + 1])$ as a proposal distribution, where $[x]$ is the floor function, meaning the integer part of x for positive x, i.e. $[2.95] = 2$.

```
#we use the same function with different proposal distribution

Metro_Hest_ch = function(target, stat_point, steps){

  # vector to store the sample
```

```
  X = rep(stat_point,steps)

  # loop for all steps
  for (i in 2:steps) {

    # generate one random value from the proposal distribution
    # this value depends on the previous value
    Y = rchisq(n = 1,floor(X[i-1]))

    # random value between 0 and 1
    u = runif(n = 1)

    #ratio for the target dist
    #r1 = target(Y)/target(X[i-1])

    #ratio for the proposal distribution, conditional prob
    #r2 = dlnorm(x = X[i-1] , meanlog = Y, sdlog = 1 )/dlnorm(x = Y, meanlog = X[i-1], sdlog = 1 )


    a = min(1, (target(Y)*dchisq(X[i-1],
                                 df = floor(Y)))/
              (target(X[i-1])*dchisq(Y,df = floor(X[i-1]))))

    # Calculate the acceptance probability
    # is a probability so have to be betwwen 0 and 1

    #acceptance = min(1, r1*r2 )

    #the acceptance condition
    if (u<=a) {
      X[i] = Y
    }else X[i] = X[i-1]
  }

  return(X)

}

X_chi  = Metro_Hest_ch(target = target_pdf,stat_point = 3.1,steps = 10000)
df = data.frame(x = c(1:10000), y = X_chi)

ggplot(df) +
  geom_line(mapping = aes(x = df$x,y= df$y))+
  labs(title = " Plot for Metropolis-Hastings", x = "Iterations", y = "Y")+
  theme_dark()
```



1.3 Comparing the two samples

Compare the results of Steps 1 and 2 and make conclusions.

For the two previous tasks we implement the Metropolis Hasting algorithm using two different proposal distributions, Log normal and Chi - square. In order to compare their result we generate a 10000 sample for both of them and we plot it. As mentioned before, both of them converge and it seems that they produce a really accurate estimation. One difference we can mention is that the implementation with the Chi - square as proposal seems to converge later than the one with the Log Normal. We can also comment on the variance of each sample. The log- Normal sample has one observation more 20 while for the Chi square the highest value is close to 18.

1.4 Convergence monitoring with Gelman - Rubin method.

Generate 10 MCMC sequences using the generator from Step 2 and starting points 1,2,3...10. Use the Gelman-Rubin method to analyze convergence of these sequences.

```
library(coda)
# a for loop to generate sequences.

set.seed(12345)
steps = c(50,100,500,100,5500,10000,100000)
con_value = c()
for (i in 1:7){

  chains = list()
  for (k in 1:10) {
    chains[[k]] = as.mcmc((Metro_Hest_ch(target = target_pdf,
                                         stat_point = k,steps = steps[i])))
  }
  # add all the chains in one mcmc list
  #gelman.diag works only with mmm objects
  MCMC = mcmc.list(chains)
  gelman = gelman.diag(MCMC)
  con_value[i] = gelman$psrf[2]
}
```

```r
df = data.frame(steps = steps, Upper_CI = con_value )

knitr::kable(x = df, caption = "Compare the Upper CI values")
```

Table 1: Compare the Upper CI values

steps	Upper_CI
50	1.845484
100	1.143434
500	1.017135
100	1.052396
5500	1.004310
10000	1.001029
100000	1.000102

In this task we use Gelman-Rubin method in order to analyze convergence of 10 MCMC sequences using sample from task 2. In order for the chain to converge the value for the upper limit should be approximately close to 1. We can also make conclusions about the specific number of iterations the chain needs in order to converge. Therefore, we run this method for many different steps value and the results can be seen in the table above. It is obvious that the algorithm need more than 500 iterations in order to converge and after 10000 the value for the upper Confidence interval does not change that much. Therefore, a value close to 10000 for the total number of steps seems reasonable because the chain converges and the algorithm is not computational heavy also.

1.5 Estimate an integral of a Gamma distribution

Estimate $\int_0^\infty xf(x)dx$ *using the sample from steps 1 and 2.*

```r
# estimation for the integral
# just the mean of each sample

# estimate for the Log Normal
int_log = mean(X_log)

#estimate for the Chi square
inte_chi = mean(X_chi)
```

In this task we want to estimate a definite integral in the general form of $\int_D h(x)dx$. Let refer to it as θ. So, in particular, we want to estimate

$$\theta = \int_0^\infty xf(x)dx$$

.

Let's also define $h(x) = g(x)f(x)$, where in this case $f(x)$ is a known distribution and $g(x) = x$. We can also use the formula below:

$$\theta = E[f(x)] = \int_D g(x)f(x)dx$$

Finally an estimator for this expected value for $f(x)$ can be found by this formula:

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

, which is the mean of each sample in this task.

For the first sample, which is distributed as Log - Normal we have the above estimation: 6.0372186

For the second sample, Chi- square distribution, the estimation is 6.0332922.

1.6 Gamma distribution, estimation of the integral and conclusions.

The distribution generated is in fact a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

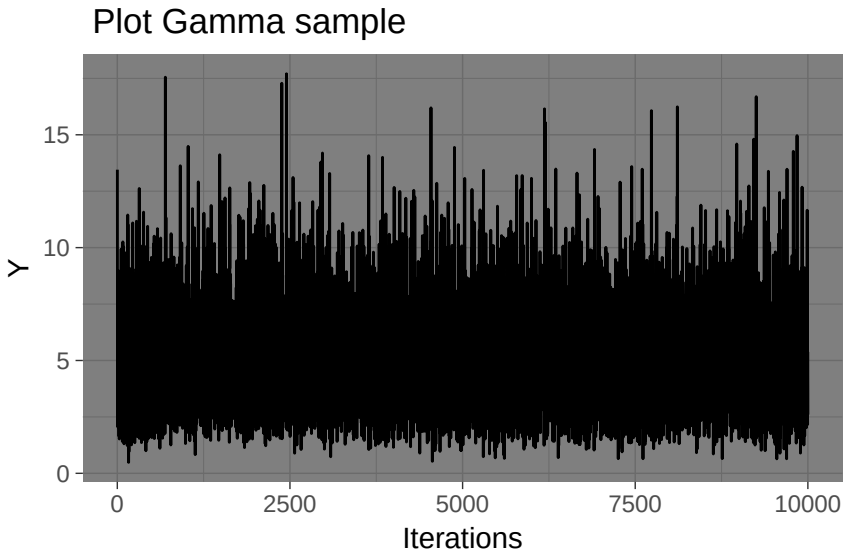
The integral we have to calculate is :

$$\int_0^\infty xx^5e^{-x}dx = \int_0^\infty xf(x;\alpha=6,\beta=1)dx = E[X] = \frac{\alpha}{\beta} = \frac{6}{1} = 6$$

, which is the integral generated by a Gamma distribution. So the real value of this interval is 6. From the task above we obtain similar values. More specific, using the log Normal, we obtain the estimation 6.0372186 while for the Chi square the value 6.0332922. Those estimations are really close to the real value 6 which is another proof that the two algorithms provide accurate samples. Of course, we can expect that those values can be exactly 6 because we are trying to estimate a gamma distribution, so every time the sample generated from a Gamma will be more accurate than our estimator.

We also generate a sample from the Gamma distribution with parameters $\alpha = 6, \beta = 1$ and we plot it. It seems like both samples generate a similar sample, and that Log - normal capture the "outliers".

```
# gamma sample
df_g = data.frame(x = c(1:10000), y = rgamma(10000,shape = 5,1))

ggplot(df_g) +
  geom_line(mapping = aes(x = df_g$x,y= df_g$y))+
  labs(title = " Plot Gamma sample", x = "Iterations", y = "Y")+
  theme_dark()
```



Question 2 Gibbs Sampling, Bayessian Theorem and Trace plots.

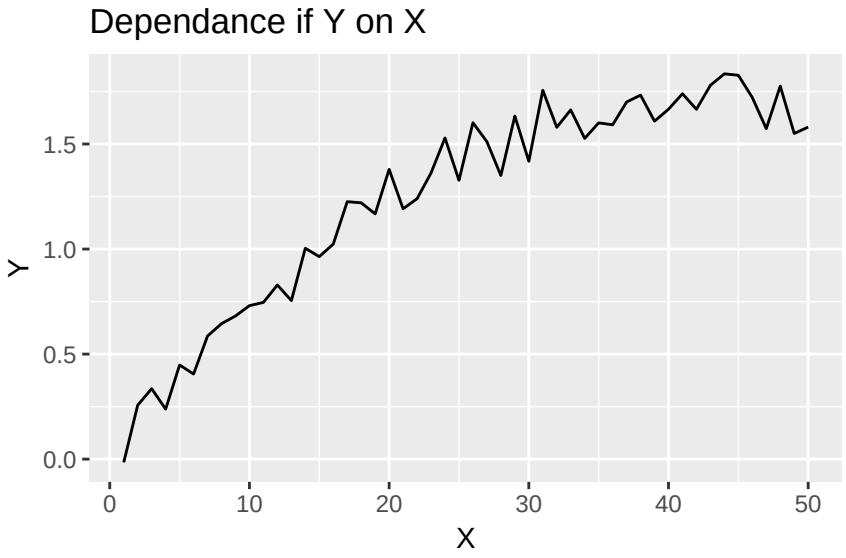
A concentration of a certain chemical was measured in a water sample, and the result was stored in the data chemical.RData having the following variables: X: day of the measurement Y: measured concentration of the chemical.

The instrument used to measure the concentration had certain accuracy; this is why the measurements can be treated as noisy. Your purpose is to restore the expected concentration values.

2.1 Import(load) Rdata and plot the dependence of Y on X.

Import the data to R and plot the dependence of Y on X. What kind of model is reasonable to use here?

```
library(ggplot2)
# load the data
load("../chemical.RData")

# data frame for the plot
df = data.frame(x = X, y = Y)

#plot the dependance
ggplot(df) +
  geom_line(mapping = aes(x = df$x, y = df$y))+
  labs(title = "Dependance if Y on X", x = "X", y = "Y")
```



As mentioned to the instructions the data is really noisy. More specific, we can observe that there are many fluctuations between the points and the accuracy of most of the models may not be high. If we have to choose a model for this data, a quadratic linear regression may capture the trend of the data but still can not provide really accurate predictions.

2.2 Bayessian model, Likelihood and Prior of Normal distriubtion.

A researcher has decided to use the following (random- walk) Bayesian model (n=number of observations, $\mu = (\mu_1, ..., \mu_n)$ are unknown parameters):

$$Y_i \sim N(\mu_i, 0.2), i = 1, ..., n$$

, *where the prior is*

$$P(\mu_1) = 1$$
$$P(\mu_{i+1}|\mu_i) = N(\mu_i, 0.2), i = 1, ..., n-1$$

Present the formulae showing the likelihood $p(Y|\mu)$ *and the prior* $p(\mu)$*. Hint: a chain rule can be used here* $p(\mu) = p(\mu_1)p(\mu_2|\mu_1)...p(\mu_n|\mu_{n-1})$*.*

First we start with the likelihood which is given by taking the product of the PDF for each parameter. The PDF for each normal distribution is as follows:

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\vec{y_i}-\vec{\mu_i})^2}{2\sigma^2}}$$

So, that gives us the following formula for the likelihood.

$$P(\vec{Y}|\vec{\mu}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\vec{y_i}-\vec{\mu_i})^2}{2\sigma^2}}$$

Which can be rewritten as:

$$P(\vec{Y}|\vec{\mu}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(\vec{y_i}-\vec{\mu_i})^2}$$

, using the properties of the product.

The prior can be written as:

$$P(\vec{\mu}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n-1}(\vec{\mu_{i+1}}-\vec{\mu_i})^2}$$

2.3 Bayes Theorem, calculating the Posterior distribution

Use Bayes' Theorem to get the posterior up to a constant proportionality, and then fnd out the distributions of $(\mu_i\mu_{-i}, Y)$*, where* $\mu_{-i}i$ *is a vector containing all* μ *values except of* μ_i

Hint A: consider for separate formulae for $(\mu_1|\mu_{-1}, Y), (\mu_n|\mu_{-n}, Y)$ *and then a formula for all remaining* $(\mu_i|\mu_{-i}, Y)$*.*

Hint B:

$$exp\bigg(-\frac{1}{d}\Big((x-a)^2 + (x-b)^2\Big)\bigg) \propto exp\bigg(-\frac{\Big(x-(a+b)/2\Big)^2}{d/2}\bigg)$$

Hint C:

$$exp\bigg(-\frac{1}{d}\big((x-a)^2 + (x-b)^2 + (x-c)^2\big)\bigg) \propto exp\bigg(-\frac{\big(x-(a+b+c)/3\big)^2}{d/3}\bigg)$$

Given the likelihood and the prior distribution above we can now get the posterior distribution for each i using the Bayes Theorem: More specific the Bayes Theorem is the one below:

$$P(\theta \backslash D) \propto P(D \backslash \theta)P(\theta)$$

Using the above formula we get the formula for the posterior distribution:

$$P(\vec{\mu}\backslash\vec{Y}) \propto \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(\vec{y_i}-\vec{\mu_i})^2}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n-1}(\vec{\mu_{i+1}}-\vec{\mu_i})^2} \Rightarrow$$

$$P(\vec{\mu}\backslash\vec{Y}) \propto \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^{2n} e^{-\frac{1}{2\sigma^2}\left[\sum_{i=1}^{n}(\vec{y_i}-\vec{\mu_i})^2+\sum_{i=1}^{n-1}(\vec{\mu_{i+1}}-\vec{\mu_i})^2\right]}$$

The posterior distribution does not depend on the constant so from now on we will refer to any constant using the leter C. Therefore the final posterior we get is:

$$P(\vec{\mu}\backslash\vec{Y}) \propto Ce^{-\frac{1}{2\sigma^2}\left[\sum_{i=1}^{n}(\vec{y_i}-\vec{\mu_i})^2+\sum_{i=1}^{n-1}(\vec{\mu_{i+1}}-\vec{\mu_i})^2\right]}$$

Now we have to use the posterior in order to find the distribution of each μ_i. We have identify which distribution is given by this posterior formula. The problem now is that the formula is not a known pdf so we have to do the maths and try to find out the distribution which is behind. In order to achieve that we have different cases. Each i contributes only to the $i-1$ and to the $i+1$ element. So we need to find the marginal for each μ_i. But for the special cases, $i=1$ and $i=n$ we have to think differently. More specific, for $i=1$ we care only about the next point because we do not have any previous, while for $i=n$ we do not have $=n+1$. The explanation below will make this statement more clear.

Firstly of all, we analyze the formula above in order to be more understandable. So

$$P(\vec{\mu}\backslash\vec{Y}) \propto e^{-\frac{1}{2\sigma^2}\left[(y_1-\mu_1)^2+(y_2-\mu_2)^2+\ldots+(y_{n-1}-\mu_{n-1})^2+(y_n-\mu_n)^2+(\mu_2-\mu_1)^2+(\mu_3-\mu_2)^2+\ldots+(\mu_n-\mu_{n-1})^2\right]}$$

Now, for $i=1$ we are only interesting in the parts of the above sum which depends on this specific i. All the other parts of the sum can be treated like constants. So

$$P(\mu_1|\mu_{-1},Y) \propto C_1e^{-\frac{1}{2\sigma^2}\left[(y_1-\mu_1)^2+(\mu_2-\mu_1)^2\right]} = e^{-\frac{1}{2\sigma^2}\left[(\mu_1-y_1)^2+(\mu_1-\mu_2)^2\right]}$$

, because of the property $(a-b)^2=(b-a)^2$.

So now, our target is to find the distribution for this pdf. We use the Hint A in which as x we have μ_1, $\alpha=y_1$, and $\beta=\mu_2$.

Therefore,

$$P(\mu_1|\mu_{-1},Y) \propto C_1e^{-\frac{1}{\sigma^2}\left(\mu_1-\frac{(y_1+\mu_2)}{2}\right)^2} \sim \mathcal{N}(\frac{y_1+\mu_2}{2},\ \frac{\sigma^2}{2})\,.$$

For $i=n$, we will not have any parameter for $n+1$ so we only need $i=n,n-1$. Therefore the general formula can now be written as:

$$P(\mu_n|\mu_{-n},Y) \propto C2e^{-\frac{1}{2\sigma^2}\left[(y_n-\mu_n)^2)+(\mu_n-\mu_{n-1})^2\right]}$$

Our target here is to find a distribution for μ_n as random variable, so we can use the Hint B using as random variable μ_n. Using the same properties as before and hint B, we obtain the result above:

$$P(\mu_n|\mu_{-n},Y) \propto C_2e^{-\frac{1}{\sigma^2}\left[\mu_n-\frac{(y_n+\mu_{n-1})^2}{2}\right]} \Rightarrow$$

$$\mu_n|\mu_{-n} \sim \mathcal{N}(\frac{\mu_{n-1}+y_n}{2},\ \frac{\sigma^2}{2})\,.$$

Finally, we have to find the distribution which every μ_i, except for $\mu_n$and μ_1, follows. As we did before we only care about the cases in which i appears. Therefore we do not need all the indexed of the sum, only the $i, i-1, i+1$.

$$P(\mu_i|\mu_{-i}, Y) \propto C_3 e^{-\frac{1}{2\sigma^2}\left[(y_i-\mu_i)^2)+(\mu_i-\mu_{i-1})^2+(\mu_{i+1}-\mu_i)^2\right]} \Rightarrow$$

In this case we can use the Hint C, again having as random variable $x = \mu_i$. Therefore :

$$P(\mu_i|\mu_{-i}, Y) \propto C e^{-\frac{3}{2\sigma^2}\left[\mu_i-\frac{(\mu_{i-1}+y_i+\mu_{i+1})}{3}\right]^2} \Rightarrow$$

$$\mu_i|\mu{-}i \sim \mathcal{N}(\frac{1}{3}(\mu_{i+1}+y_i+\mu_{i-1}),\ \frac{\sigma^2}{3})\,.$$

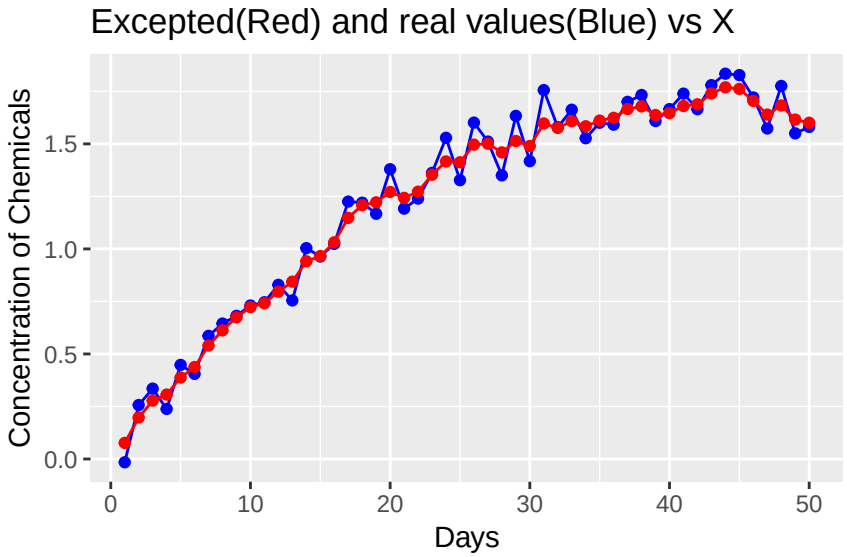
We have also to mention that for all calculations the rest of the elements we do not need are constants. That is the reason why we have those $C's$ in every posterior.

2.4 Gibbs sampler, Monte carlo approach and Bayes Theorem

Use the distributions derived in Step 3 to implement a Gibbs sampler that uses $\vec{\mu^0} = (0, ..., 0)$ *as a starting point. Run the Gibbs sampler to obtain 1000 values of the vector* μ *and then compute the expected value of vector* μ *by using a Monte Carlo approach. Plot the expected value of vector* μ *versus* X *and* Y *versus* X *in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of vector* μ *can catch the true underlying dependence between* Y *and* X*?*

```
#function for Gibbs sampler
Gibbs = function(steps,X0,Y,var){
  # steps: number of iterations we want
  # X0 : The vector of the initial values of the sample
  #Y a vector of the real values we want to sample
  # the variance of the distribution, in this case all of them is 0.2

  d = length(X0) # length of each vector
  sample = matrix(0,nrow = steps,ncol = d)# store all the sample
  sample[1,] = X0 #store the initial vector in the first row

  for (i in 2:steps) {

    X = sample[i-1 , ] # we need the previous vector in the calculations
    mu = rep(x = 0, d)# Store each vector

    # using the formula for i=1 above
    mu[1] = rnorm(1, mean = (Y[1]+ X[2])/2,sd =  sqrt(var/2))

    # for loop for all other i's except the first and the last mu
    for (j in 2:(d-1)) {
      mu[j] = rnorm(1,mean = (X[j+1] + Y[j] + mu[j-1])/3 ,sd = sqrt(var/3))
    }

    # final point of the vector
    mu[d] = rnorm(1,mean = (1/2)*(mu[d-1] + Y[d]),sd = sqrt(var/2))
    sample[i,] = mu

  }
```

```
  return(sample)

}

#Starting points in a vector
X0 = rep(0,50)
all_mu = Gibbs(steps = 1000, X0 = X0,Y = Y,var = 0.2)

# for each parrameter we take the mean of all iterations
mu = apply(all_mu,2,mean)


df = data.frame(x = X, y = Y, mu = mu)

ggplot(df) +
  geom_line(mapping = aes(x = df$x, y = df$y),color = "blue")+
  geom_point(mapping = aes(x = df$x, y = df$y),color = "blue")+
  geom_line(mapping = aes(x = df$x, y = df$mu), color = "red")+
  geom_point(mapping = aes(x = df$x, y = df$mu), color = "red")+
  labs(title = "Excepted(Red) and real values(Blue) vs X",
       x = "Days", y = "Concentration of Chemicals")
```



In this task we implement the Gibbs algorithm in order to generate a sample and restore the expected concentration values. We can give a short description of the algorithm. We want to estimate a vector of 50 parameters, so we use a vector of 50 starting points all of them equal to 0. We run the algorithm for 1000 iterations and for each iterations we estimate all the parameters using the values obtain from previous iterations. The output is a matrix of 50 columns(as the length of the vector we want to estimate) and 1000 rows(iterations). Finally, we calculate the mean of each column which will give us the expected values stored in a vector of length 50.

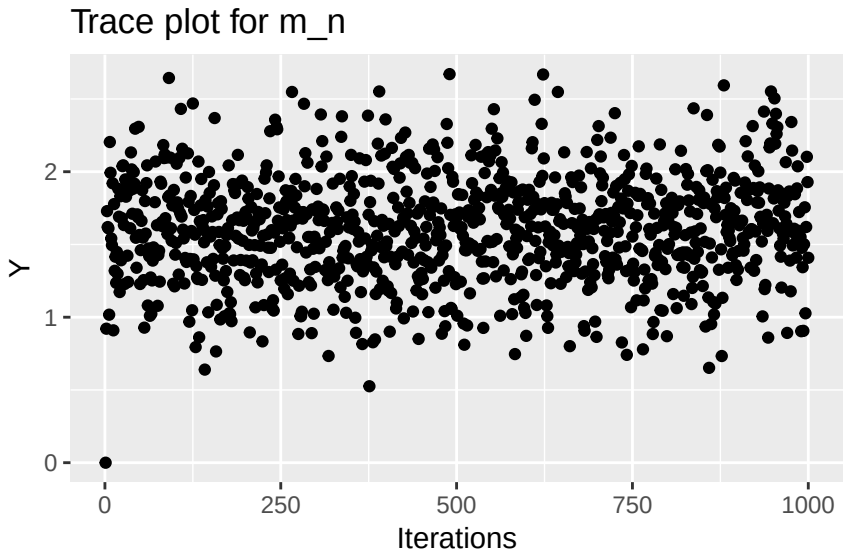
We also plot those values and the real ones against the number of days. From the plot we can observe that the noisy we had before in our data is reduced. Of course there are still fluctuations but, the variance between them is not that important which means that one suitable model can capture those fluctuations and give accurate predictions. One can also say that the expected values follow the same trend as the real ones. More specific, both values are increasing exponential until close to the 40th day when they start decreasing. In

conclusion, it seems that the algorithm does a descent job, generating an accurate sample which reduces the noisy and does not change the distribution of the real data.

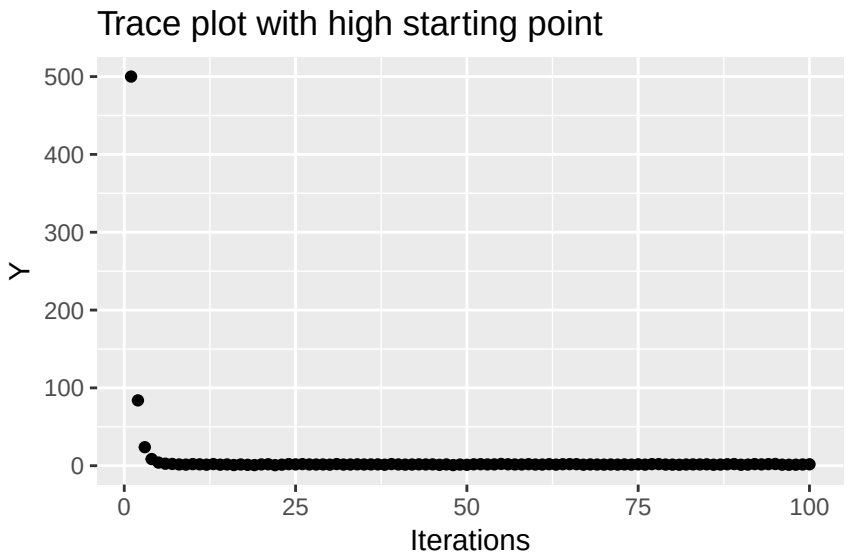
2.5 Trace plot, burn in period and convergence.

Make a trace plot for μ_n and comment on the burn-in period and convergence.

```
# trace plot for the final mu
df1 = data.frame(x = c(1:1000), y = all_mu[,50])

ggplot(df1) +
  geom_point(mapping = aes(x = df1$x, y =df1$y))+
  labs(title = "Trace plot for m_n", x = "Iterations", y= "Y")
```




```
X0 = c(rep(0,49),500)
all_mu = Gibbs(steps = 1000, X0 = X0,Y = Y,var = 0.2)


df2 = data.frame(x = c(1:100), y = all_mu[1:100,50])

ggplot(df2) +
  geom_point(mapping = aes(x = df2$x, y =df2$y))+
  labs(title = "Trace plot with high starting point", x = "Iterations", y= "Y")
```



We also present a trace plot for μ_{50} in order to comment on the burn in period and in the convergence of this chain. In order to have a better picture for the burn in period we run again the algorithm but we use as starting point for the μ_{50} a really high number (500). It is obvious that the algorithm reaches the stationery phase fast enough so we can choose as a burn in period just 10 observation. We may need to run the algorithm more times but it can be said that the chain converges with a mean value close to 1.5.