Computational Statistics (732A90) - Lab 4 Report

Hariprasath Govindarajan (hargo729), Hector Plata (hecpl268)

05 February 2019

Contents

Question 1: Computations with Metropolis-Hastings

Considet the following probability density function:

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

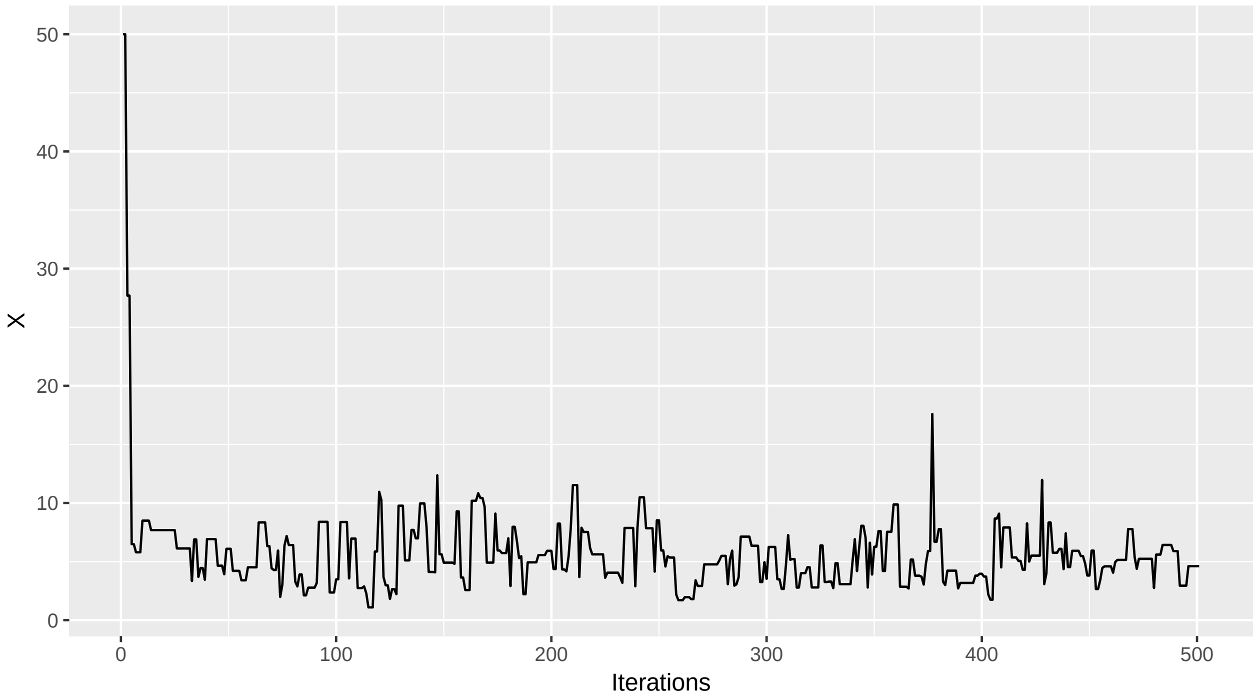
You can see that the distribution is known up to some constant of proportionality. If you are interested (**NOT** part of the lab) this constant can be found by applying integration by parts multiple times and equal to 120.

Task 1.1

Use Metropolis-Hastings algorithm to generate samples from this distribution by using proposal distribution as Log-normal $LN(X_t, 1)$, 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?

The results and the code for the Metropolis-Hastings sample can be seen below. We can see that the chain converges to values around 1.95 and 10.48 pretty qucikly. As for the burn-in period, the size could be around 10 iterations, since from this point we can see the samples moves around the bound described above.

```r
# Metropolis-Hasting initialization.
x = c(50)
Tmax = 500
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
  return((x^5) * exp(-x))
}

# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dlnorm(x, meanlog=log(y))) /
            (target_prop(x) * dlnorm(y, meanlog=log(x))))
  return(a)
}

# Generating the Markov Chain.
for (i in 1:Tmax)
{
  # Generating a candidate point Y.
  y = rlnorm(1, meanlog=log(x[i]), sdlog=1)

  # Calculating the probability of adding y
  # as the last state of the chain.
  alpha_y = alpha(x[i], y)

  # Sampling from a random uniform distribution.
  u = runif(1)

  # Updating rule.
  if (u < alpha_y)
  {
    x = c(x, y)
  }
  else
  {
    x = c(x, x[i])
  }
}

# Plotting the chain realized by the Metropolis-Hasting Sampler.
p = ggplot() +
    geom_line(aes(x=1:(Tmax + 1), y=x)) +
    labs(x="Iterations", y="X")

print(p)
```



```r
print(quantile(sort(x), probs=c(0.025, 0.975)))
```

```
##      2.5%     97.5%
##  1.959411 10.483670
```

```r
print(mean(x))
```

```
## [1] 5.695149
```

```r
# Saving the values for later.
step1_x = x
```

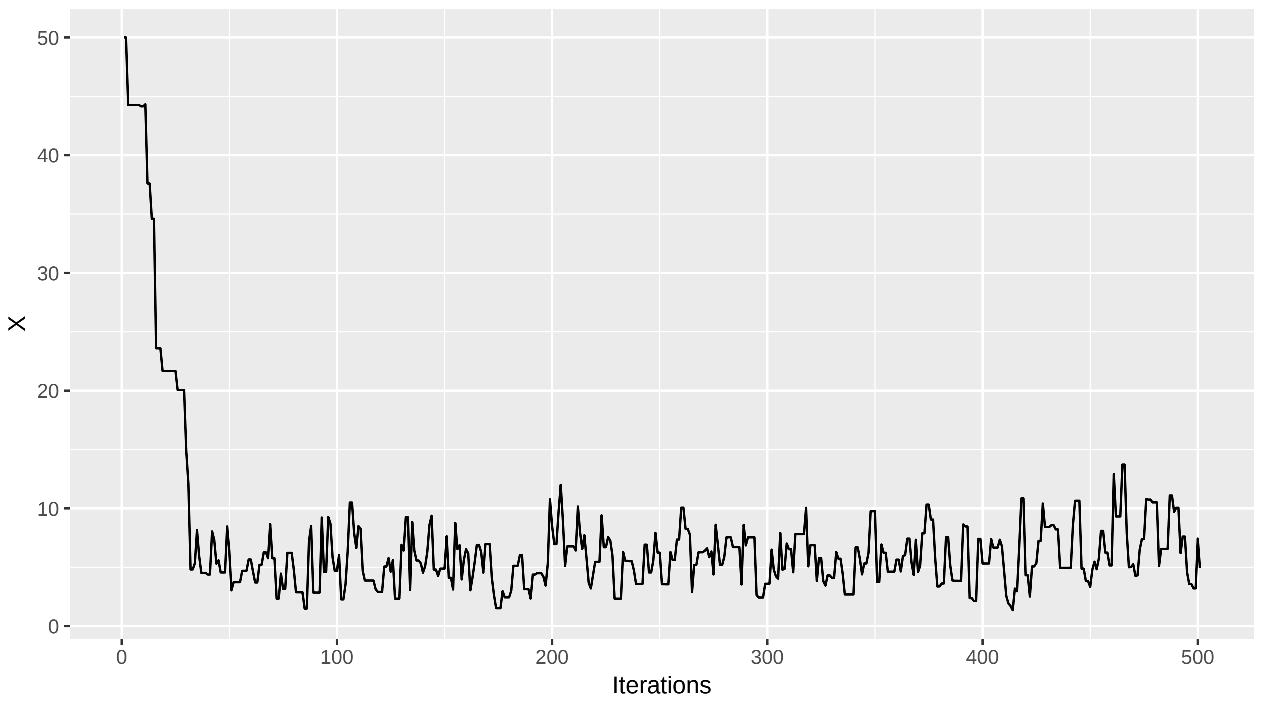
Task 1.2

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

The results from this experiment and the code can be seen below.

```r
# Metropolis-Hasting initialization.
x = c(50)
Tmax = 500
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
  return((x^5) * exp(-x))
}
```

```r
# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dchisq(x, df=floor(y + 1))) /
            (target_prop(x) * dchisq(y, df=floor(x + 1))))
  return(a)
}

# Generating the Markov Chain.
for (i in 1:Tmax)
{
  # Generating a candidate point Y.
  y = rchisq(1, df=floor(x[i]+ 1))

  # Calculating the probability of adding y
  # as the last state of the chain.
  alpha_y = alpha(x[i], y)

  # Sampling from a random uniform distribution.
  u = runif(1)

  # Updating rule.
  if (u < alpha_y)
  {
    x = c(x, y)
  }
  else
  {
    x = c(x, x[i])
  }
}

# Plotting the chain realized by the Metropolis-Hasting Sampler.
p = ggplot() +
    geom_line(aes(x=1:(Tmax + 1), y=x)) +
    labs(x="Iterations", y="X")

print(p)
```



```
print(quantile(sort(x), probs=c(0.025, 0.975)))
```

```
##      2.5%     97.5%
##  2.329737 36.098127
```

```
print(mean(x))
```

```
## [1] 7.341222
```

```
# Saving the values for later.
step2_x = x
```

Task 1.3

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

From the results of the previous steps we can see that the convergence takes more time for this proposal distribution. In this case (for step 2) the size of the burn-in period should be around 30. As for the distribution of the points. Both results have a similar lower bound, however the upper bound is much higher for the chi-square proposal than for the log-normal one. Also, the mean is higher for the chi-square than for the log-normal one.

Task 1.4

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

We see that the point estimate for the Gelman-Rubin factor is about 1 and the upper 95% confidence interval for this point estimate is 1.01. This suggests that our sequences converges

```
# Metropolis-Hasting initialization.
X = seq(1, 10)
Tmax = 500
```

```r
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
  return((x^5) * exp(-x))
}

# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dchisq(x, df=floor(y + 1))) /
            (target_prop(x) * dchisq(y, df=floor(x + 1))))
  return(a)
}

# List that is going to hold all of the sequences.
mcmc_runs = list()

for (x0 in X)
{
  # Initializing the sequence.
  x = c(x0)

  # Generating the Markov Chain.
  for (i in 1:Tmax)
  {
    # Generating a candidate point Y.
    y = rchisq(1, df=floor(x[i]+ 1))

    # Calculating the probability of adding y
    # as the last state of the chain.
    alpha_y = alpha(x[i], y)

    # Sampling from a random uniform distribution.
    u = runif(1)

    # Updating rule.
    if (u < alpha_y)
    {
      x = c(x, y)
    }
    else
    {
      x = c(x, x[i])
    }
  }
  # Saving the run into the mcmc_runs.
  mcmc_runs[[x0]] = coda::as.mcmc(x)
}

# Transforming the data to get the test done
# and printing the test results.
mcmc_runs = coda::as.mcmc.list(mcmc_runs)
res = coda::gelman.diag(mcmc_runs)
print(res)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1       1.01
```

Task 1.5

Estimate

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

Using the samples from step 1 and 2.

Given that the results seems to convergence, we can use both sequences as follows:

$$\mathbf{E}\left[x\right] = \int_0^\infty xf(x)dx \approx \frac{1}{n-m}\sum_{t=m+1}^{n} x_t$$

The results for both sequences can be seen below.

```
# Burn-in period.
m = 30
n = length(step1_x)

# Results of the integral.
print(mean(step1_x[(m + 1):n]))
```

```
## [1] 5.330902
```

```
print(mean(step2_x[(m + 1):n]))
```

```
## [1] 5.770096
```

Task 1.6

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 pdf of a gamma distribution is given by:

$$f\left(x|k,\theta\right) = \frac{1}{\Gamma\left(k\right)\theta^k}x^{k-1}e^{-\frac{x}{\theta}}$$

As stated on the the beginning of the example, the quantity $\frac{1}{\Gamma(k)}$ is equal to 120. The parameters k and θ can be infered from the approximation given at the start. Thus, $k = 6$ and $\theta = 1$.

The expected value of the gamma function is defined as follows:

$$\mathbf{E}[x] = k\theta = 6$$

From this result we see that the values are in some sense close to the real one. However, the values obtained given a log-normal distrubtion seem to be off by a reasonable amount compared to the ones generated by the chi-square distribution. This seems reasonable since the log-normal distribution is only associated with the normal distribution, thus, the proposal distribution is far away from the target one. While the relationship between the gamma and chi-square since the chi-square distribution is a special case of the gamma distribution.

Question 2: Gibbs sampling

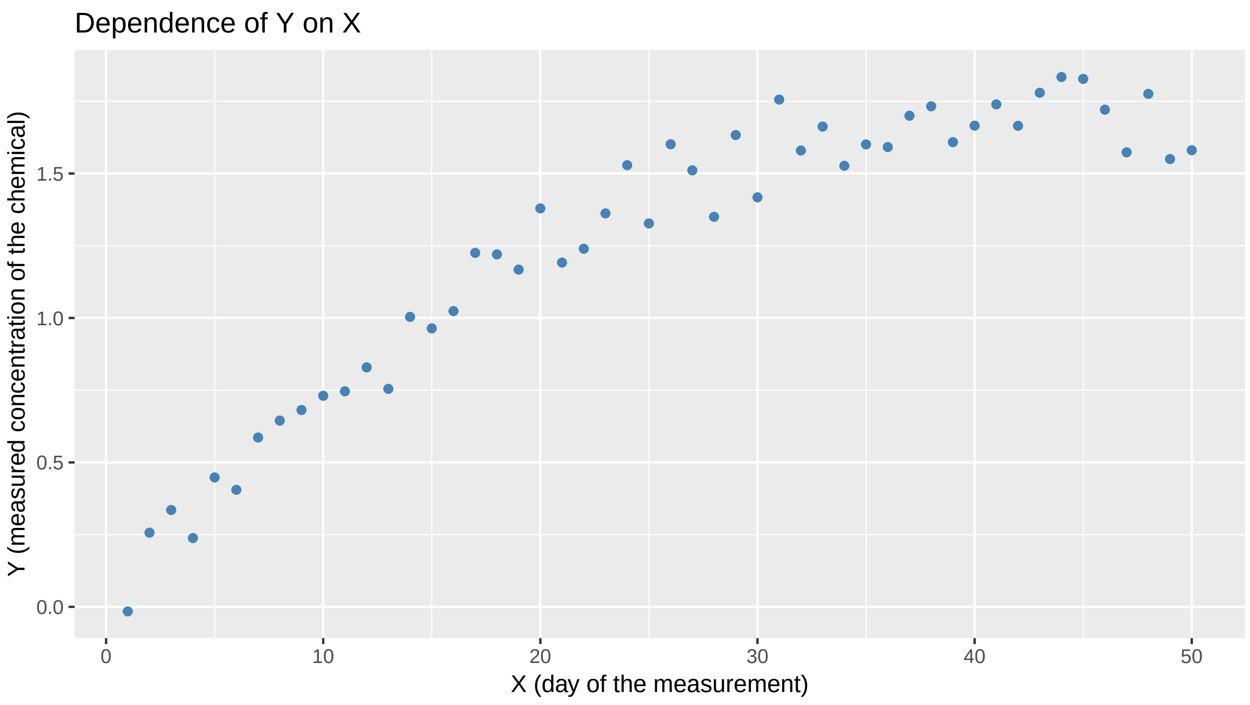
A concentration of a certain chemical was measured in a water sample, and the result was stored in the data `chemical.RData` having the following variables:

- `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. The purpose of this question is to restore the expected concentration values.

Task 2.1

We have imported the `chemical.RData` file which contains the `X` and `Y` variables and the following plot shows the dependence of Y on X.



From the plot, we see that the dependence is slightly curved. So, a linear regression with polynomial terms of degree 2 or 3 will be reasonable to model this data.

Task 2.2

We consider the following random-walk Bayesian model where n is the number of observations and $\vec{\mu} = (\mu_1, \mu_2, ..., \mu_n)$ are unknown parameters.

$$Y_i \sim \mathcal{N}(\mu_i, variance = 0.2), \quad i = 1, 2, ..., n$$

where the prior is:

$$p(\mu_1) = 1$$
$$p(\mu_{i+1}|\mu_i) = \mathcal{N}(\mu_i, 0.2)$$

Notations

$\mathcal{N}[m, v](x)$ - PDF of a normally distributed random variable x with mean (m) and variance (v).

$\vec{\mu}_{-i}$ - A vector containing all μ values except μ_i

σ^2 - Variance whose value is given as `0.2`

Formula for the prior: $p(\vec{\mu})$

We can use a chain rule to express this prior as follows:

$$p(\vec{\mu}) = p(\mu_1) \cdot p(\mu_2|\mu_1) \cdot p(\mu_3|\mu_2)...p(\mu_n|\mu_{n-1})$$
$$p(\vec{\mu}) = \mathcal{N}[\mu_1, \sigma^2](\mu_2) \cdot \mathcal{N}[\mu_2, \sigma^2](\mu_3)...\mathcal{N}[\mu_{n-1}, \sigma^2](\mu_n)$$
$$\boxed{Prior: \quad p(\vec{\mu}) = \prod_{i=1}^{n-1} \mathcal{N}[\mu_i, \sigma^2](\mu_{i+1})}$$

Formula for the likelihood: $p(\vec{Y}|\vec{\mu})$

Since each Y_i depends only on the corresponding μ_i according to the Bayesian model that we have assumed, we can express the likelihood as follows:

$$p(\vec{Y}|\vec{\mu}) = p(Y_1|\mu_1) \cdot p(Y_2|\mu_2)...p(Y_n|\mu_n) = \prod_{i=1}^{n} p(Y_i|\mu_i)$$
$$\boxed{Likelihood: \quad p(\vec{Y}|\vec{\mu}) = \prod_{i=1}^{n} \mathcal{N}[\mu_i, \sigma^2](Y_i)}$$

Task 2.3

2.3.1 Formula for posterior

By applying the Bayes theorem, we obtain:

$$p(\vec{\mu}|\vec{Y}) \propto p(\vec{Y}|\vec{\mu})p(\vec{\mu})$$

We want to get the posterior only upto a constant proportionality. So, we can get rid of any constant terms which might arise during this simplification process.

$$p(\vec{\mu}|\vec{Y}) \propto \left(\cancelto{(const.)}{\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^{2n-1}} \right) exp\left(-\frac{1}{2\sigma^2}\left[(\mu_2-\mu_1)^2+(\mu_3-\mu_2)^2+...+(\mu_n-\mu_{n-1})^2\right]\right) \times$$
$$exp\left(-\frac{1}{2\sigma^2}\left[(Y_1-\mu_1)^2+(Y_2-\mu_2)^2+...+(Y_{n-1}-\mu_{n-1})^2+(Y_n-\mu_n)^2\right]\right)$$

To simplify this expression, we use the property: $exp\left(-\frac{1}{d}((x-a)^2+(x-b)^2)\right) \propto exp\left(-\frac{[x-(a+b)/2]^2}{d/2}\right)$

The first exponent has $n-1$ terms and the second exponent has n terms. We can pair the $i-$th terms from the 2 exponents to apply this property. Note that the last term in the second exponent will not be paired to any term.

$$p(\vec{\mu}|\vec{Y}) \propto exp\left(-\frac{\left[\mu_1-\frac{\mu_2+Y_1}{2}\right]^2}{\sigma^2}-\frac{\left[\mu_2-\frac{\mu_3+Y_2}{2}\right]^2}{\sigma^2}-\frac{\left[\mu_3-\frac{\mu_4+Y_3}{2}\right]^2}{\sigma^2}-...-\frac{\left[\mu_{n-1}-\frac{\mu_n+Y_{n-1}}{2}\right]^2}{\sigma^2}-\frac{[\mu_n-Y_n]^2}{2\sigma^2}\right)$$

$$\boxed{Posterior: \quad p(\vec{\mu}|\vec{Y}) \propto exp\left(-\frac{[\mu_n-Y_n]^2}{2\sigma^2}-\sum_{i=1}^{n-1}\frac{\left[\mu_i-\frac{\mu_{i+1}+Y_i}{2}\right]^2}{\sigma^2}\right)}$$

2.3.2 Formulas for marginal distributions: $p(\mu_i|\vec{\mu}_{-i},\vec{Y})$

By observing the joint distribution containing μ_i where $i=1,2,3,...,n$ we can find the marginal functions for μ_i upto constant proportionality by finding only those terms containing the corresponding μ_i. From these marginal functions, we can figure out the marginal distribution based on the form of the expression.

Marginal for μ_1

Firstly, we want to get the formula for $p(\mu_1|\vec{\mu}_{-1},\vec{Y})$. So, we look for the terms in the posterior containing μ_1 and we see that only the first term in the exponent part of the posterior contains μ_1.

$$p(\mu_1|\vec{\mu}_{-1},\vec{Y}) \propto exp\left(-\frac{\left[\mu_1-\frac{\mu_2+Y_1}{2}\right]^2}{\sigma^2}\right)$$

Since the expression above resembles a normal distribution function, we can write it as a normal ditribution with appropriate parameters which will normalize this function.

$$\boxed{p(\mu_1|\vec{\mu}_{-1},\vec{Y}) = \mathcal{N}\left[\frac{\mu_2+Y_1}{2},\frac{\sigma^2}{2}\right](\mu_1)}$$

Since the product of 2 normal PDFs is proportional to a normal PDF, we can find m' and σ' such that $\mathcal{N}[m',\sigma'](x) \propto \mathcal{N}[m_1,\sigma_1^2](x) \times \mathcal{N}[m_2,\sigma_2^2](x)$. The new parameters can be found by the following formulas:

$$m' = \frac{m_1\sigma_2^2 + m_2\sigma_1^2}{\sigma_1^2+\sigma_2^2} \quad ; \quad \sigma'^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2} \tag{1}$$

We will use this property for finding the marginal distribution parameters for μ_n and μ_i.

Marginal for μ_n

To get the marginal for μ_n, we extract the last 2 terms in the exponent part of the posterior which are the only terms containing μ_n.

$$p(\mu_n|\vec{\mu}_{-n},\vec{Y}) \propto exp\left(-\frac{\left[\mu_{n-1}-\frac{\mu_n+Y_{n-1}}{2}\right]^2}{\sigma^2}\right) \times exp\left(-\frac{[\mu_n-Y_n]^2}{2\sigma^2}\right)$$

We rearrange the terms so that the 2 exponential expressions resemble normal PDFs of μ_n. The product of 2 normal PDFs can be written as a new normal function. We calculate the mean and variance of this new normal distribution using the formulas from (1).

$$p(\mu_n|\vec{\mu}_{-n},\vec{Y}) \propto exp\left(-\frac{[\mu_n-(2\mu_{n-1}-Y_{n-1})]^2}{4\sigma^2}\right) \times exp\left(-\frac{[\mu_n-Y_n]^2}{2\sigma^2}\right)$$

$$m_1 = (2\mu_{n-1}-Y_{n-1}); \quad m_2 = Y_n; \quad \sigma_1^2 = 2\sigma^2; \quad \sigma_2^2 = \sigma^2$$

$$m' = (2Y_n + 2\mu_{n-1} - Y_{n-1})/3; \quad \sigma'^2 = 2\sigma^2/5 = 0.08$$

$$\boxed{p(\mu_n|\vec{\mu}_{-n},\vec{Y}) = \mathcal{N}\left[\frac{1}{3}(2Y_n+2\mu_{n-1}-Y_{n-1}),\frac{2\sigma^2}{3}\right](\mu_n)}$$

Marginal for μ_i

To get the marginal for μ_i, we extract the terms in the exponent part of the posterior which contains μ_i and we see that every i and $(i-1)$th term contains μ_i.

$$p(\mu_i|\vec{\mu}_{-i},\vec{Y}) \propto exp\left(-\frac{\left[\mu_{i-1}-\frac{\mu_i+Y_{i-1}}{2}\right]^2}{\sigma^2}\right) \times exp\left(-\frac{\left[\mu_i-\frac{\mu_{i+1}+Y_i}{2}\right]^2}{\sigma^2}\right)$$

We rearrange the terms so that the 2 exponential expressions resemble normal PDFs of μ_i. The product of 2 normal PDFs can be written as a new normal function. We calculate the mean and variance of this new normal distribution using the formulas from (1).

$$p(\mu_i|\vec{\mu}_{-i},\vec{Y}) \propto exp\left(-\frac{[\mu_i-(2\mu_{i-1}-Y_{i-1})]^2}{4\sigma^2}\right) \times exp\left(-\frac{\left[\mu_i-\frac{\mu_{i+1}+Y_i}{2}\right]^2}{\sigma^2}\right)$$

$$m_1=(2\mu_{i-1}-Y_{i-1}); \quad m_2=(\mu_{i+1}+Y_i)/2; \quad \sigma_1^2=2\sigma^2; \quad \sigma_2^2=\sigma^2/2$$

$$m'=(2Y_i+2\mu_{i-1}+2\mu_{i+1}-Y_{i-1})/5; \quad \sigma'^2=2\sigma^2/5=0.08$$

$$\boxed{p(\mu_i|\vec{\mu}_{-i},\vec{Y})=\mathcal{N}\left[\frac{1}{5}(2Y_i+2\mu_{i-1}+2\mu_{i+1}-Y_{i-1}),\frac{2\sigma^2}{5}\right](\mu_i)}$$

To summarize, these are the marginal distributions for μ_i:

$$\boxed{p(\mu_i|\vec{\mu}_{-i},\vec{Y})=\begin{cases}\mathcal{N}\left[\frac{\mu_2+Y_1}{2},\frac{\sigma^2}{2}\right] & i=1\\ \mathcal{N}\left[\frac{1}{5}(2Y_i+2\mu_{i-1}+2\mu_{i+1}-Y_{i-1}),\frac{2\sigma^2}{5}\right] & 1<i<n\\ \mathcal{N}\left[\frac{1}{3}(2Y_n+2\mu_{n-1}-Y_{n-1}),\frac{2\sigma^2}{3}\right] & i=n\end{cases}}$$

Task 2.4

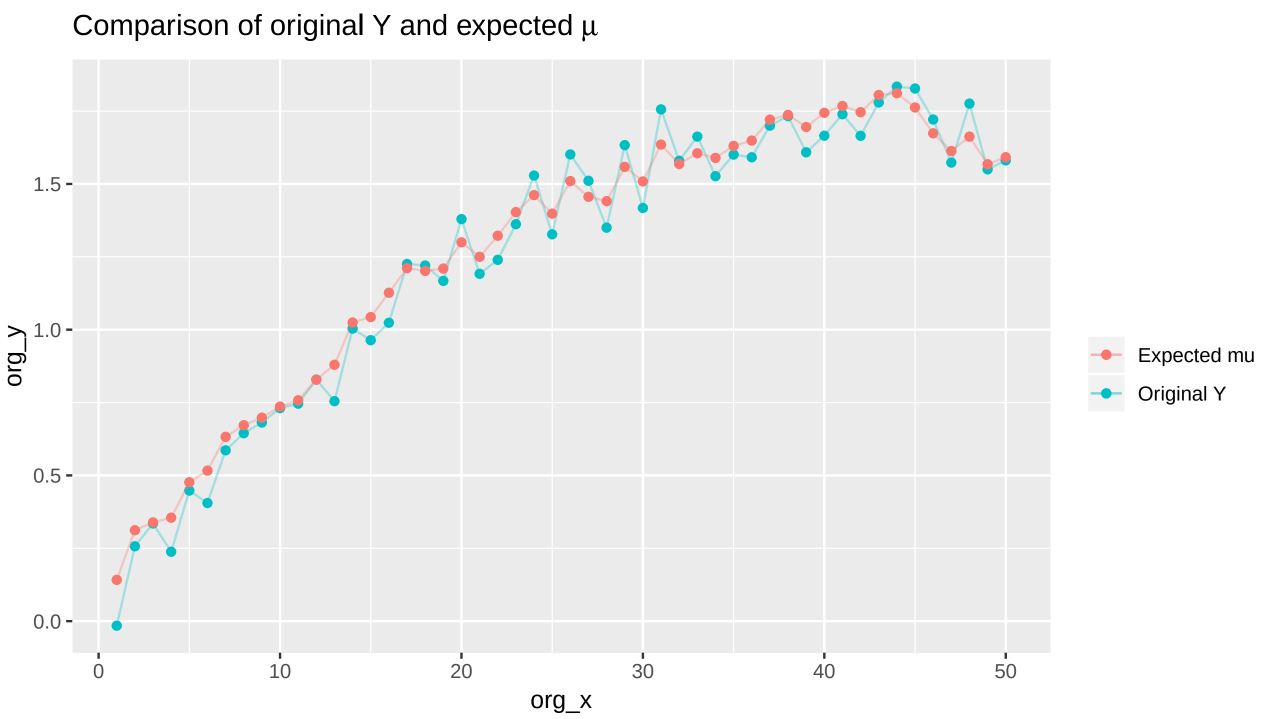
Using the marginal distributions derived for μ_i in the previous task, we implement a Gibbs sampler which uses values, $\vec{\mu}^0=(0,0,...,0)$ as a starting point. We run this Gibbs sampler to obtain 1000 values of $\vec{\mu}$ and then calculate the expected value of $\vec{\mu}$ using a Monte Carlo approach.

The function `generate_mu()` takes the parameters `i` - index of μ, `mu_now` - latest values of $\vec{\mu}$, `Y` - actual values of $\vec{Y}$ and returns a randomly generated value for μ_i based on the corresponding marginal distribution.

The function `mcmc_gibbs()` takes the parameters `Y` - actual values of $\vec{Y}$ and `num_iter` - the number of Gibbs samples to be generated and returns a matrix containing the samples of $\vec{\mu}$.

```r
# Function to generate values from marginal distributions
generate_mu = function(i, mu_now, Y){
  n = length(mu_now)

  mean_mu = 0
  var_mu = 0

  # Set mean and variance for mu_i based on i=1/n/others
  if(i == 1){
    mean_mu = (Y[1] + mu_now[2])/2
    var_mu = 0.1
  }
  else if(i == n){
    mean_mu = (2*Y[n] - Y[n-1] + 2*mu_now[n-1])/3
    var_mu = 2*0.2/3
  }
  else{
    mean_mu = (2*mu_now[i-1] + 2*mu_now[i+1] + 2*Y[i] - Y[i-1])/5
    var_mu = 0.08
  }
  # Generate mu_i using a normally distributed marginal distribution
  rnorm(1, mean_mu, sqrt(var_mu))
}

# Function to run Gibbs sampling
mcmc_gibbs = function(Y, num_iter){

  n = length(Y)
  mu0 = rep(0,length(Y))

  mu = matrix(0, nrow = num_iter, ncol = n)
  mu[1, ] = mu0

  for (i in 2:num_iter){
    latest_mu = mu[i-1, ]

    for(j in 1:n){
      latest_mu[j] = generate_mu(j, latest_mu, Y)
    }

    mu[i, ] = latest_mu
    }

  return(mu)
}

set.seed(7654321)
gibbs_res = mcmc_gibbs(Y, 1000)

exp_mu = colMeans(gibbs_res)
```

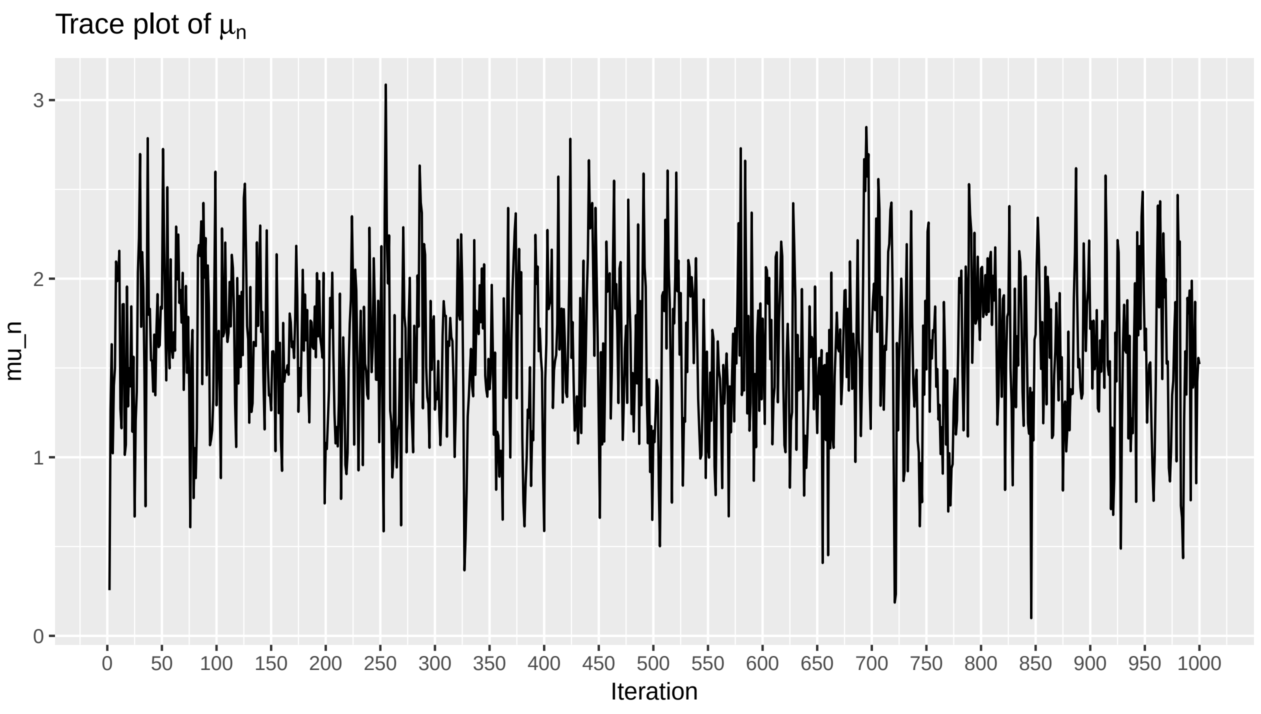
The following plot shows original Y vs X along with the expected values of $\vec{\mu}$ vs X.



By observing the plot, we surely see that the noise is reduced compared to the original values of Y but it is not completely removed. Though the plot of expected values of $\vec{\mu}$ is not very smooth, it closely follows the original values of Y and has reduced the noise to some extent. So, it seems like it has captured the true underlying dependence between Y and X reasonably well.

Task 2.5

The following is the traceplot of μ_n obtained from the 1000 Gibbs samples.



The mean value of μ_n is 1.5528 based on the 1000 samples. From the above plot, we can see from the first 200 samples that μ_n initially varies around a value close to 1.6 to 1.7. Only after 200 samples, we observe the μ_n values to oscillate around 1.5. We can hence say that the burn-in period was around 200 samples. Between 200 and 1000 samples, we see that the oscillations appear to be more reasonable and centered around the value 1.5. Hence, we can say that the Gibbs sampler has reached convergence starting from around 200 samples.

Appendix

```r
# Global options
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE, fig.width=8)

library(ggplot2)    # For plotting
library(latex2exp)

# ------------------------------------------------------------------------------
# Question 1: Computations with Metropolis-Hastings
# ------------------------------------------------------------------------------

# Metropolis-Hasting initialization.
x = c(50)
Tmax = 500
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
  return((x^5) * exp(-x))
}

# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dlnorm(x, meanlog=log(y))) /
            (target_prop(x) * dlnorm(y, meanlog=log(x))))
  return(a)
}

# Generating the Markov Chain.
for (i in 1:Tmax)
{
  # Generating a candidate point Y.
  y = rlnorm(1, meanlog=log(x[i]), sdlog=1)

  # Calculating the probability of adding y
  # as the last state of the chain.
  alpha_y = alpha(x[i], y)

  # Sampling from a random uniform distribution.
  u = runif(1)

  # Updating rule.
  if (u < alpha_y)
  {
    x = c(x, y)
  }
  else
  {
    x = c(x, x[i])
  }
}

# Plotting the chain realized by the Metropolis-Hasting Sampler.
p = ggplot() +
    geom_line(aes(x=1:(Tmax + 1), y=x)) +
    labs(x="Iterations", y="X")

print(p)

print(quantile(sort(x), probs=c(0.025, 0.975)))

print(mean(x))


# Saving the values for later.
step1_x = x

# Metropolis-Hasting initialization.
x = c(50)
Tmax = 500
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
  return((x^5) * exp(-x))
}

# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dchisq(x, df=floor(y + 1))) /
             (target_prop(x) * dchisq(y, df=floor(x + 1))))
  return(a)
}

# Generating the Markov Chain.
for (i in 1:Tmax)
{
  # Generating a candidate point Y.
  y = rchisq(1, df=floor(x[i]+ 1))

  # Calculating the probability of adding y
  # as the last state of the chain.
  alpha_y = alpha(x[i], y)

  # Sampling from a random uniform distribution.
  u = runif(1)

  # Updating rule.
  if (u < alpha_y)
  {
    x = c(x, y)
  }
  else
  {
    x = c(x, x[i])
  }
}

# Plotting the chain realized by the Metropolis-Hasting Sampler.
p = ggplot() +
    geom_line(aes(x=1:(Tmax + 1), y=x)) +
    labs(x="Iterations", y="X")

print(p)

print(quantile(sort(x), probs=c(0.025, 0.975)))

print(mean(x))

# Saving the values for later.
step2_x = x

# Metropolis-Hasting initialization.
X = seq(1, 10)
Tmax = 500
seed = 4

# Setting the seed :).
set.seed(seed)


# Defining the proportional target probability.
target_prop = function(x)
{
```

```r
  return((x^5) * exp(-x))
}

# Defining the probability alpha that a
# canditate Y is accepted as the next state
# of the chain.
alpha = function(x, y)
{
  a = min(1, (target_prop(y) * dchisq(x, df=floor(y + 1))) /
            (target_prop(x) * dchisq(y, df=floor(x + 1))))
  return(a)
}

# List that is going to hold all of the sequences.
mcmc_runs = list()

for (x0 in X)
{
  # Initializing the sequence.
  x = c(x0)

  # Generating the Markov Chain.
  for (i in 1:Tmax)
  {
    # Generating a candidate point Y.
    y = rchisq(1, df=floor(x[i]+ 1))

    # Calculating the probability of adding y
    # as the last state of the chain.
    alpha_y = alpha(x[i], y)

    # Sampling from a random uniform distribution.
    u = runif(1)

    # Updating rule.
    if (u < alpha_y)
    {
      x = c(x, y)
    }
    else
    {
      x = c(x, x[i])
    }
  }

  # Saving the run into the mcmc_runs.
  mcmc_runs[[x0]] = coda::as.mcmc(x)
}

# Transforming the data to get the test done
# and printing the test results.
mcmc_runs = coda::as.mcmc.list(mcmc_runs)
res = coda::gelman.diag(mcmc_runs)
```

```r
print(res)

# Burn-in period.
m = 30
n = length(step1_x)

# Results of the integral.
print(mean(step1_x[(m + 1):n]))
print(mean(step2_x[(m + 1):n]))

# ------------------------------------------------------------------------------
# Question 2: Gibbs sampling
# ------------------------------------------------------------------------------

load("chemical.RData")

ggplot() + geom_point(aes(x = X, y = Y), color = "steelblue") +
  ggtitle("Dependence of Y on X") +
  xlab("X (day of the measurement)") +
  ylab("Y (measured concentration of the chemical)")

# Function to generate values from marginal distributions
generate_mu = function(i, mu_now, Y){
  n = length(mu_now)

  mean_mu = 0
  var_mu = 0

  # Set mean and variance for mu_i based on i=1/n/others
  if(i == 1){
    mean_mu = (Y[1] + mu_now[2])/2
    var_mu = 0.1
  }
  else if(i == n){
    mean_mu = (2*Y[n] - Y[n-1] + 2*mu_now[n-1])/3
    var_mu = 2*0.2/3
  }
  else{
    mean_mu = (2*mu_now[i-1] + 2*mu_now[i+1] + 2*Y[i] - Y[i-1])/5
    var_mu = 0.08
  }

  # Generate mu_i using a normally distributed marginal distribution
  rnorm(1, mean_mu, sqrt(var_mu))
}

# Function to run Gibbs sampling
mcmc_gibbs = function(Y, num_iter){

  n = length(Y)
  mu0 = rep(0,length(Y))

  mu = matrix(0, nrow = num_iter, ncol = n)
  mu[1, ] = mu0

  for (i in 2:num_iter){
    latest_mu = mu[i-1, ]

    for(j in 1:n){
      latest_mu[j] = generate_mu(j, latest_mu, Y)
    }

    mu[i, ] = latest_mu
    }

  return(mu)
}

set.seed(7654321)
gibbs_res = mcmc_gibbs(Y, 1000)

exp_mu = colMeans(gibbs_res)

ggplot(data.frame(org_x = X, exp_mu = exp_mu, org_y = Y)) +
  geom_point(aes(x = org_x, y = org_y, color = "Original Y")) +
  geom_line(aes(x = org_x, y = org_y, color = "Original Y"), alpha = 0.3) +
  geom_point(aes(x = org_x, y = exp_mu, color = "Expected mu")) +
  geom_line(aes(x = org_x, y = exp_mu, color = "Expected mu"), alpha = 0.3) +
  labs(color="") + ggtitle(TeX("Comparison of original Y and expected $\\mu$"))

ggplot(data.frame(x = 2:nrow(gibbs_res),
                  y = gibbs_res[2:nrow(gibbs_res), ncol(gibbs_res)])) +
  geom_line(aes(x = x, y = y)) +
  xlab("Iteration") + ylab("mu_n") +
  ggtitle(TeX("Trace plot of $\\mu_n$")) +
  scale_x_continuous(breaks = seq(0,1000,50))
```