STAT340 – Project 2

André Victor Ribeiro Amaral

Pratik Nag

December 31, 2020

This project is based on the data set `Tokyo` from the `GMRF Book`. The details can be found on Section 4.3.4. We will start by stating the quantities of interest.

Part 1.

The model is defined as follows

$$Y_i|X_i = x_i \sim \text{Binomial}(m = 2,\ p(x_i)),\ \text{where } p(x_i) = \Phi(x_i) \text{ is the CDF of Normal}(0,\,1) \text{ at } x_i.$$

In addition to that, $X_1, X_2, \cdots, X_n \sim \text{RW2}(\kappa)$, where κ is the *precision*. The prior distribution for κ is such that $\frac{1}{\sqrt{\kappa}} \sim \text{Exponential}(\lambda = 1.55)$, which means that

$$\pi(\kappa) = 1.55\ \exp\left(-1.55\,\frac{1}{\sqrt{\kappa}}\right)\,\frac{1}{2\kappa^{\frac{3}{2}}}.$$

And

$$\pi(x|\kappa) \propto \kappa^{\frac{n-1}{2}}\ \exp\left(-\frac{1}{2}\kappa\sum_{i=1}^{n}(x_i - 2x_{i-1} + x_{i-2})^2\right),$$

where $x_0 = x_n$ and $x_{-1} = x_{n-1}$. In matrix notation, we have $\pi(x|\kappa) \propto \kappa^{\frac{n-1}{2}}\ \exp\left[-\frac{1}{2}\kappa\left(\boldsymbol{x}^{\mathrm{T}}R\,\boldsymbol{x}\right)\right]$.

Now, if we want to sample from $\pi(\kappa|x, y)$, we can say that, for an instant (t), $\kappa_\star^{(t+1)} = a \cdot \kappa^{(t)}$, such that

$$\pi(a) \propto 1 + \frac{1}{a},\ \text{for } a \in \left[\frac{1}{A},\ A\right] \text{ and zero otherwise} - \text{with } A > 0 \text{ (in particular, set } A = 2).$$

In this case, if $A = 2$, then $\pi(a) = \frac{1}{c}\left(1 + \frac{1}{a}\right)$, for $a \in \left[\frac{1}{2},\ 2\right]$, for $c = \frac{3}{2} + \ln(4)$. One way to sample from $\pi(a)$ is to compute $F_A^{-1}(U)$, such that $U \sim \text{Uniform}[0, 1]$. But instead of computing the exact form of $F_A^{-1}(\cdot)$, we can deal with an empirical inverse CDF, which works just fine for our case (one can do this using the `splinefun()` function; but instead, we will use the `my.scale.proposal()` implemented by Professor Rue).

And then, we will accept $\kappa_\star$ with probability $\alpha = \min\{1, R\}$, where

$$R = \frac{\pi(\kappa_\star|x, y)}{\pi(\kappa|x, y)} = \frac{\pi(\kappa_\star) \cdot \pi(x|\kappa_\star)}{\pi(\kappa) \cdot \pi(x|\kappa)};$$

otherwise, set $\kappa_\star^{(t+1)} = \kappa^{(t)}$. In practice, we will work with log-scale; i.e., $\alpha = \exp(\min\{0, \ln(R)\})$.

After that, we can update x_i; here, we will have that

$$\pi(x_i|\boldsymbol{x}_{-i},\kappa,y) \propto \kappa^{\frac{n-1}{2}} \exp\left[-\frac{1}{2}\kappa\left(\boldsymbol{x}^{\mathrm{T}}R\,\boldsymbol{x}\right)\right] \cdot \prod_{i=1}^{n} \Phi(x_i)^{y_i} \cdot (1-\Phi(x_i))^{m-y_i},\ \forall i \in \{1,\cdots,n\}.$$

Notice that, conditioned on $\boldsymbol{x}_{-i}$, κ and y, we can ignore the terms that does not depend on x_i.

And then, for an instant t, we can say that ${x_i}_\star^{(t+1)} = {x_i}^{(t)} + \epsilon_i$, such that $\epsilon_i \sim \text{Normal}(0,\sigma^2)$ (say, for instance, that $\sigma^2 = 1$); i.e., ${x_i}_\star^{(t+1)} \sim \text{Normal}({x_i}^{(t)}, \sigma^2 = 1)$, which can be seen as the *proposal kernel*.

Finally, we will accept $x_{i\star}$ with probability $\alpha = \exp(\min\{0, \ln(R)\})$, where

$$\ln(R) = \ln(\pi(x_{i\star}|\boldsymbol{x}_{-i},\kappa,y)) - \ln(\pi(x_i|\boldsymbol{x}_{-i},\kappa,y));$$

otherwise, ${x_i}_\star^{(t+1)} = {x_i}^{(t)}$

IMPORTANT: *in all solutions, we will first present the code (which was previously executed). Then, based on the data set that we have generated from it, we will present the results and their interpretations.*

```
tfile = tempfile()
download.file("http://hrue.r-inla-download.org/Tokyo.RData", destfile = tfile)
load(tfile)
str(Tokyo)
Tokyo = Tokyo[Tokyo$n == 2, ]
Tokyo$time = 1:nrow(Tokyo)
R = 67500 * toeplitz(c(6, -4, 1, rep(0, 360), 1, -4))
m = 2
n = 365
Y = Tokyo$y
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
logdensityX_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) - k *
    (X %*% (R) %*% X)))
a_values = seq(from = -39, to = 39, by = 0.1)
density = dnorm(a_values)
```

```
cumulative_density = cumsum(density)
cumulative_density = cumulative_density/max(cumulative_density)
normal_cdf = splinefun(a_values, cumulative_density)
gX = function(x, y, mm) {
    return(y * log(normal_cdf(x)) + (mm - y) * log(1 - normal_cdf(x)))
}
logDensityY_X = function(x, y, m) return(sum(gX(x, y, m)))
my.scale.proposal <- function(n, F = 2) {
    x <- numeric(n)
    if (F < 1)
        F = 1/F
    if (F == 1) {
        x[] <- 1
    } else {
        len = F - 1/F
        unif = (runif(n) < len/(len + 2 * log(F)))
        m <- sum(unif)
        x[unif] <- runif(m, min = 1/F, max = F)
        x[!unif] <- F^runif(n - m, min = -1, max = 1)
    }
    return(x)
}
main = function(k, X, Y, R, m, n) {
    value = TRUE
    count = 0
    p1 = c()
    p2 = c()
    p3 = c()
    k_iter = c()
    while (value) {
        k_old = k
        k_new = my.scale.proposal(1, F = 2) * k
        k_new_posterior = log(density_k(k_new, 1.55)) + logdensityX_k(k_new,
```

```
            X, R, n)
        k_posterior = log(density_k(k, 1.55)) + logdensityX_k(k, X, R, n)
        logR = k_new_posterior - k_posterior
        a = exp(min(0, logR))
        if (runif(1) < a)
            k = k_new
        X_new = X
        for (i in 1:n) {
            X_old = X
            x_new = X_old[i] + rnorm(1, mean = 0, sd = 1)
            x_old = X_old[i]
            X_posterior = logdensityX_k(k, X_old, R, n) + logDensityY_X(x_old,
                Y[i], m)
            X_old[i] = x_new
            X_new_posterior = logdensityX_k(k, X_old, R, n) + logDensityY_X(x_new,
                Y[i], m)
            logR = X_new_posterior - X_posterior
            a = exp(min(0, logR))
            if (runif(1) < a)
                X_new[i] = x_new
        }
        X = X_new
        p1 = append(p1, X[130])
        p2 = append(p2, X[3])
        p3 = append(p3, X[300])
        k_iter = append(k_iter, k)
        count = count + 1
        if (count >= 10000) {
            X_final = X_final + X
            k_final = k_final + k
        } else {
            X_final = X
            k_final = k
```

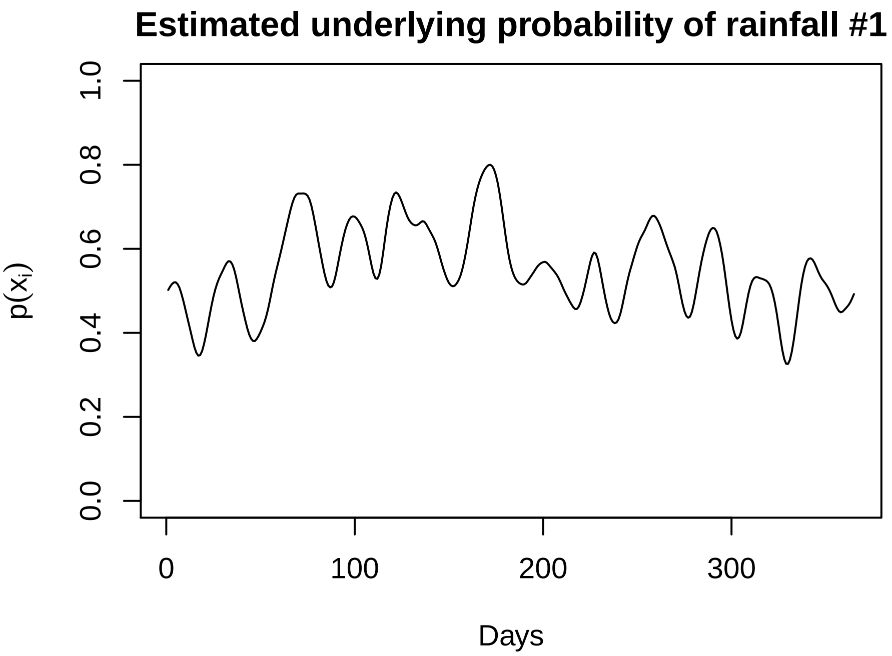
```r
        }
        if (count == 50000) {
            X_final = X_final/40000
            k_final = k_final/40000
            break
        }
    }
    return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final,
        k_iter = k_iter))
}
X = runif(n)
k = 1
U = main(k, X, Y, R, m, n)
```

In order to keep the consistency among the results, we will always consider the average of the posterior sample, with some burn-in, for all 365 *timepoints*, and we will also check the sample from the 3^{rd}, 130^{th} and 300^{th} days.

So now, we can load and analyze our results.

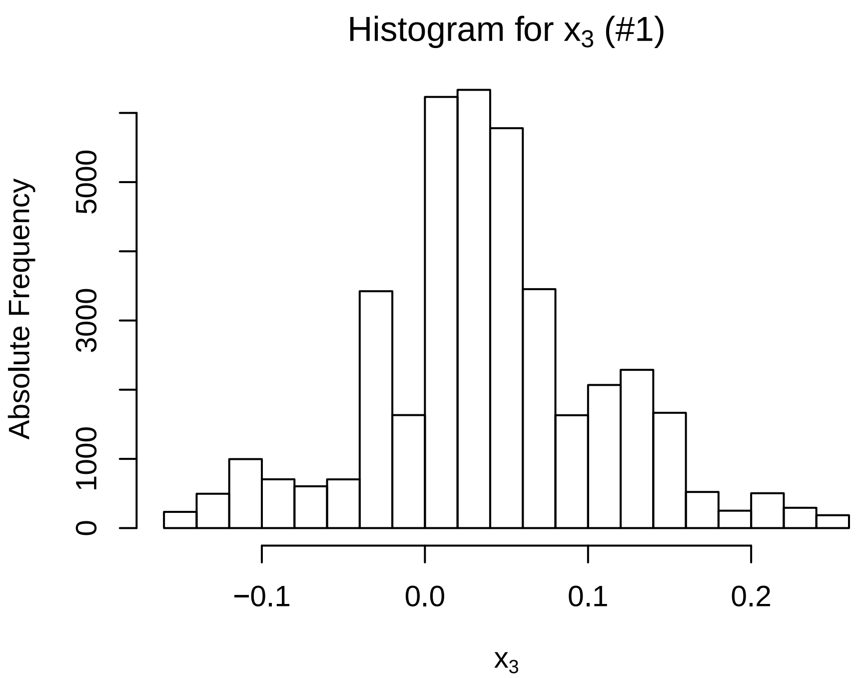
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 365\}$, with a burn-in of size $10,000$ (out of $50,000$ observations).

```r
load("data/problem1.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #1")
```

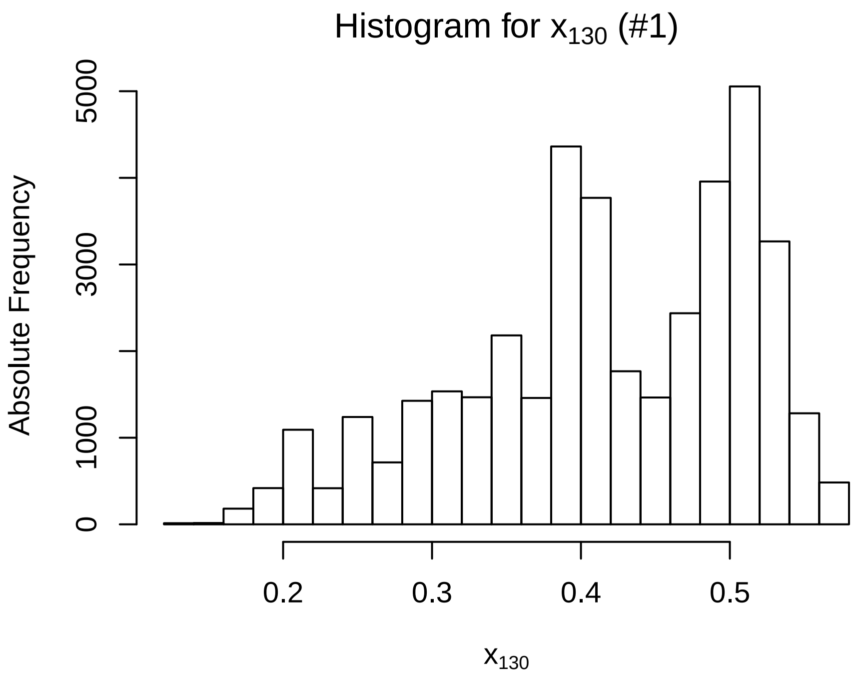


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

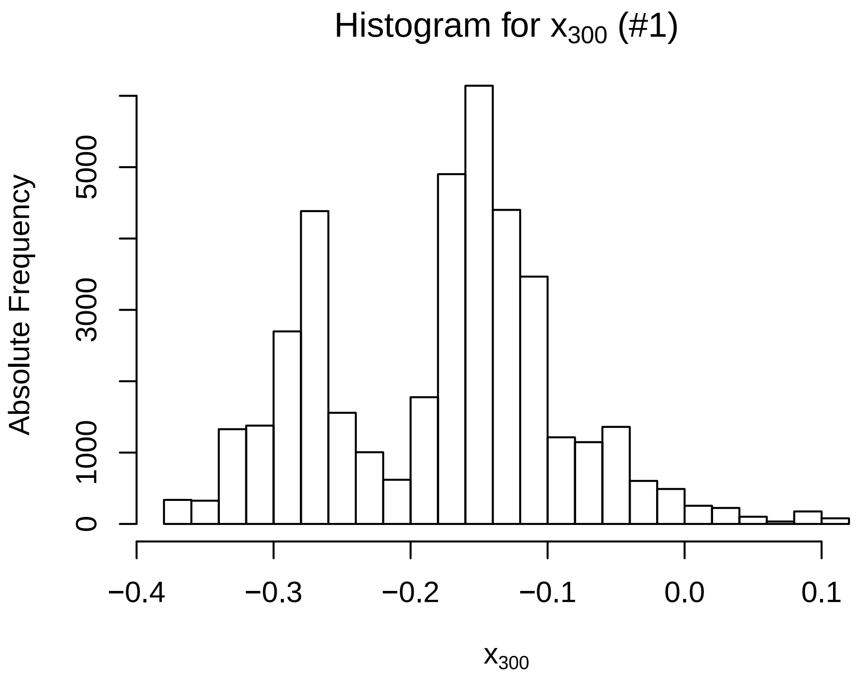
```
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 40000), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[3] * " (#1)"))
```



```r
hist(tail(U$x130, 40000), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[130] * " (#1)"))
```




```r
hist(tail(U$x300, 40000), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[300] * " (#1)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem1.RData`.

Remark: Although in the code we are showing that we have iterated through the loop $50,000$ times, if we wanted better results, we were supposed to run it for a longer time; i.e., we had to stop it earlier due to the "slow" running time. As a consequence of it, and since we did not expect a rapid convergence for this (and the next one) sampler, the results obtained for this "Part 1" (as well as "Part 2") are not reliable. From sampler #3 on, the results are much more accurate, though.

Part 2.

In this second part, the difference from what we have done so far depends on an approximation for $\pi(x_i|\boldsymbol{x}_{-i},\kappa,y)$, such that it is normally distributed. To do this, we have to approximate the term $\pi(y_i|x_i)$ in a way that it can be written as a polynomial in x_i. Start by recalling that

$$\pi(x_i|\boldsymbol{x}_{-i},\kappa,y) \propto \exp\left[-\frac{1}{2}\kappa\left(a_i {x_i}^2 + b_i x_i\right)\right] \cdot \Phi(x_i)^{y_i} \cdot (1-\Phi(x_i))^{m-y_i}.$$

Now, take the log of it.

$$\ln \pi(x_i|\boldsymbol{x}_{-i},\kappa,y) \propto -\frac{1}{2}\kappa\left(a_i {x_i}^2 + b_i x_i\right) + \left[y_i \ln(\Phi(x_i)) + (m-y_i)\ln(1-\Phi(x_i))\right]. \tag{1}$$

Then, let $g(x_i) = y_i \ln(\Phi(x_i)) + (m-y_i)\ln(1-\Phi(x_i))$ and write the Taylor's expansion of $g(x_i)$ around ${x_i}^\star$ in the following way

$$\begin{aligned}
g(x_i) &\approx g({x_i}^\star) + g'({x_i}^\star)(x_i - {x_i}^\star) + \frac{1}{2}\,g''({x_i}^\star)(x_i - {x_i}^\star)^2 \\
&\approx g'({x_i}^\star)(x_i - {x_i}^\star) + \frac{1}{2}\,g''({x_i}^\star)(x_i - {x_i}^\star)^2 \\
&= a_i' x_i - a_i' {x_i}^\star + \frac{1}{2}\left(a_i'' {x_i}^2 - 2a_i'' x_i {x_i}^\star + a_i'' {x_i}^{\star 2}\right), \text{ where } a_i^{(k)} = g^{(k)}({x_i}^\star) \text{ for } k \in \{1,2\} \\
&\approx a_i' x_i + \frac{1}{2}\left(a_i'' {x_i}^2 - 2a_i'' x_i {x_i}^\star\right) \\
&= \frac{1}{2}\left[a_i'' {x_i}^2 + 2(a_i' - a_i'' {x_i}^\star)x_i\right] \\
&= -\frac{1}{2}(c_i {x_i}^2 + d_i x_i), \text{ where } c_i = -a_i'' \text{ and } d_i = -2(a_i' - a_i'' {x_i}^\star).
\end{aligned}$$

Thus, using the above approximation for $g(x_i)$, we can re-write Equation (1).

$$\begin{aligned}
\ln \pi_{\mathrm{G}}(x_i|\boldsymbol{x}_{-i},\kappa,y) &\propto -\frac{1}{2}\,\kappa\left(a_i {x_i}^2 + b_i x_i\right) - \frac{1}{2}\,\kappa\left(\frac{c_i {x_i}^2 + d_i x_i}{\kappa}\right) \\
&= -\frac{1}{2}\,\kappa\left[{x_i}^2\left(a_i + \frac{c_i}{\kappa}\right) + x_i\left(b_i + \frac{d_i}{\kappa}\right)\right].
\end{aligned}$$

Implying that

$$\pi_{\mathrm{G}}(x_i|\boldsymbol{x}_{-i},\kappa,y) \propto \exp\left\{-\frac{1}{2}\,\kappa\left[{x_i}^2\left(a_i + \frac{c_i}{\kappa}\right) + x_i\left(b_i + \frac{d_i}{\kappa}\right)\right]\right\}, \tag{2}$$

which can be viewed as the core of a Normal distribution.

For a vector $\boldsymbol{x}$, we can re-write Equation (2) in a matrix notation as

$$\pi_{\mathrm{G}}(\boldsymbol{x}|\kappa, y) \propto \exp\left\{-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\left[\kappa\, R + \mathrm{diag}(\boldsymbol{c})\right]\boldsymbol{x} + \boldsymbol{d}^{\mathrm{T}}\boldsymbol{x}\right\} \tag{3}$$

where $\boldsymbol{x}|\kappa, y \sim \mathcal{N}_C(\boldsymbol{c}, \kappa\, R + \mathrm{diag}(\boldsymbol{c}))$.

And then, we can compute the acceptance rate for $x_{i\star}$, such that $x_{i\star} \sim \pi_{\mathrm{G}}(x_i|\boldsymbol{x}_{-i}, \kappa, y)$, in the following way

$$\begin{aligned}\ln(R) = {} & \ln(\pi(x_{i\star}|\boldsymbol{x}_{-i}, \kappa, y)) + \ln(\pi_{\mathrm{G}}(x_i|\boldsymbol{x}_{-i}, x_{i\star}, \kappa, y)) \\ & - \ln(\pi(x_i|\boldsymbol{x}_{-i}, \kappa, y)) - \ln(\pi_{\mathrm{G}}(x_{i\star}|\boldsymbol{x}_{-i_\star}, x_i, \kappa, y)).\end{aligned}$$

From a practical point of view, there are some additional details that we have to consider; for instance, we can compute $g'(x_i{}^\star)$ and $g''(x_i{}^\star)$ numerically (e.g., using the `numDeriv` package) – without having to find an analytic solution for it. Additionally, for a fixed x_i, we have to optimize $g(x_i) = g(x_i, x_i{}^\star)$ for $x_i{}^\star$; this can also be done by setting $x_i{}^\star$ equal to the previous mean of $\pi_{\mathrm{G}}(\boldsymbol{x}|\kappa, y)$, which is the approach that we will take here.

Finally, taking these changes into account, we can implement our *single site sampler with a Gaussian-approximation proposal for the spline* in a similar way to what we have done for "Part 1.".

```
library(numDeriv)
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
logdensityX_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) - k *
    (X %*% (R) %*% X)))
logdensityX = function(X, k, index) {
    if (index == 2) {
        e = (6 * X[index]^2) + (2 * X[n] - 8 * X[index - 1]) * X[index]
    } else if (index == 1) {
        e = (6 * X[index]^2) + (2 * X[n - 1] - 8 * X[n]) * X[index]
    } else {
        e = (6 * X[index]^2) + (2 * X[index - 2] - 8 * X[index - 1]) * X[index]
    }
    return(-0.5 * k * e * 67500)
```

```
}
a_values = seq(from = -39, to = 39, by = 0.1)
density = dnorm(a_values)
cumulative_density = cumsum(density)
cumulative_density = cumulative_density/max(cumulative_density)
normal_cdf = splinefun(a_values, cumulative_density)
gX = function(x, y, mm) {
    return(y * log(normal_cdf(x)) + (mm - y) * log(1 - normal_cdf(x)))
}
b_coefficient = function(X, index) {
    if (index == 2)
        e = (2 * X[n] - 8 * X[index - 1]) else if (index == 1)
        e = (2 * X[n - 1] - 8 * X[n]) else e = (2 * X[index - 2] - 8 * X[index - 1])
    return(e)
}
gausian_approx = function(k, X, y, m, n, index) {
    x_old = X[index]
    single_deriv = grad(gX, x = X[index], y = y, mm = m)
    double_deriv = drop(hessian(gX, x = X[index], y = y, mm = m))
    r = k * 6 - double_deriv
    b = k * b_coefficient(X, index) + single_deriv - double_deriv * X[index]
    mean = b/r
    for (i in 1:n) {
        if (mean < -20 || mean > 7)
            mean = x_old
    }
    return(list(X_star = mean, variance = (1/r)))
}
main = function(k, X, Y, R, m, n) {
    value = TRUE
    count = 0
    p1 = c()
    p2 = c()
```

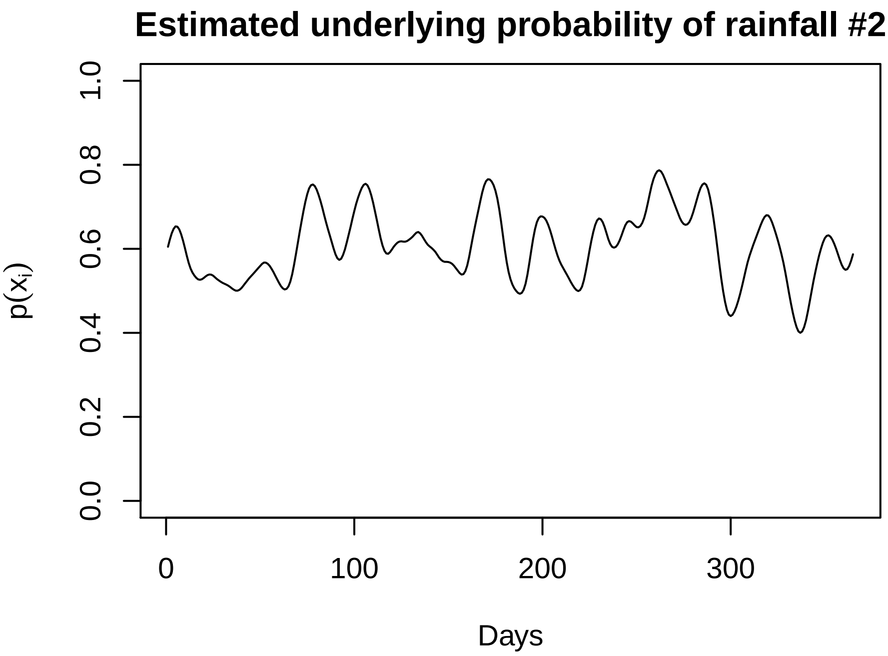
```
p3 = c()
while (value) {
    k_old = k
    k_new = my.scale.proposal(1, F = 2) * k
    k_new_posterior = log(density_k(k_new, 1.55)) + logdensityX_k(k_new,
        X, R, n)
    k_posterior = log(density_k(k, 1.55)) + logdensityX_k(k, X, R, n)
    logR = k_new_posterior - k_posterior
    a = exp(min(0, logR))
    if (runif(1) < a)
        k = k_new
    X_new = X
    for (i in 1:n) {
        # density x_old
        X_ = X
        X_oldDensity = logdensityX(X, k, i) + gX(X[i], Y[i], m)
        gausianApprox = gausian_approx(k, X, Y[i], m, n, i)
        x_new1 = rnorm(1, gausianApprox$X_star, sqrt(gausianApprox$variance))
        X_[i] = x_new1
        gausianApprox_new = gausian_approx(k, X_, Y[i], m, n, i)
        # density of x_old gaussian approximation
        X_old_Gausian = (-0.5) * ((X - gausianApprox_new$X_star)^2)/gausianApprox_new$vari
        # density x_new
        X_newDensity = logdensityX(X_, k, i) + gX(X_[i], Y[i], m)
        # density of x_new gausian approximation
        X_new_Gausian = (-0.5) * ((X_ - gausianApprox$X_star)^2)/gausianApprox$variance
        logR = X_newDensity - X_oldDensity + X_old_Gausian - X_new_Gausian
        a = exp(min(0, logR))
        if (runif(1) < a)
            X_new[i] = x_new1
    }
    X = X_new
    p1 = append(p1, X[130])
```

```r
        p2 = append(p2, X[3])
        p3 = append(p3, X[300])
        count = count + 1
        if (count >= 10000) {
            X_final = X_final + X
            k_final = k_final + k
        } else {
            X_final = X
            k_final = k
        }
        if (count == 50000) {
            X_final = X_final/40000
            k_final = k_final/40000
            break
        }
    }
    return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final))
}
X = runif(n, 3, 5)
k = 0.5
U = main(k, X, Y, R, m, n)
```

Now, we can load and analyze our results.

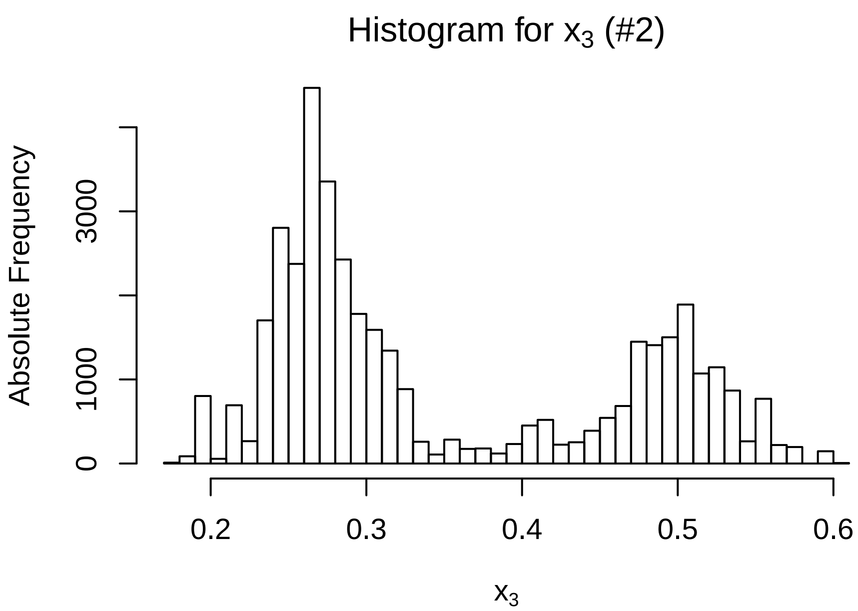
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 365\}$, with a burn-in of size $10,000$ (out of $50,000$ observations).

```r
load("data/problem2.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #2")
```

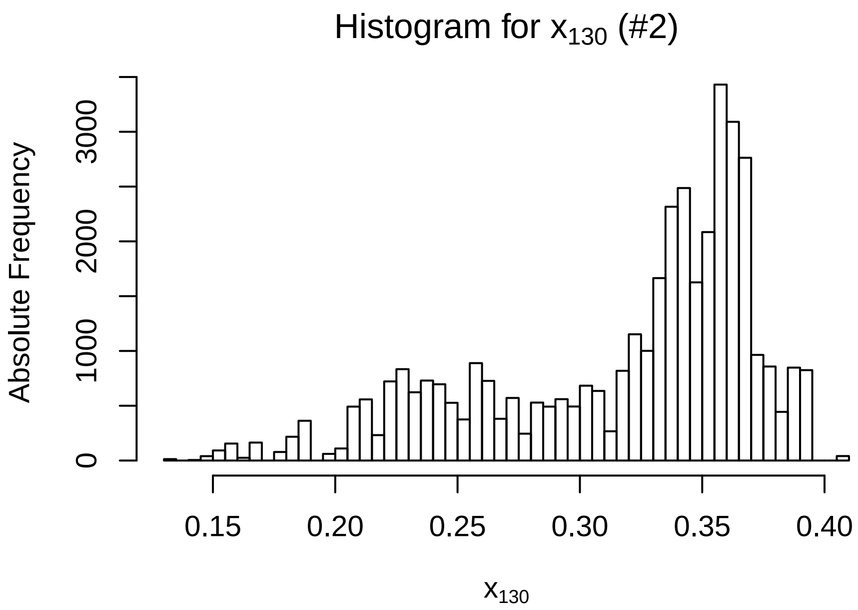


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

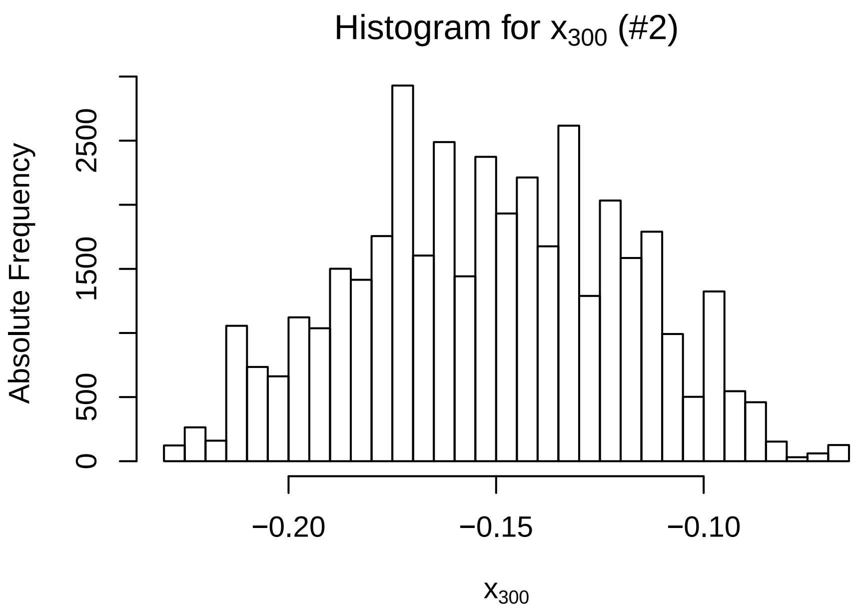
```
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 40000), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[3] * " (#2)"))
```



```r
hist(tail(U$x130, 40000), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[130] * " (#2)"))
```




```r
hist(tail(U$x300, 40000), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[300] * " (#2)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem2.RData`.

Remark: Similarly to "Part 1", the slow convergence rate that we have observed in this sampler, and the "low" number of iterations that we used here, do **not** give us reliable results.

Part 3.

This third part will use most of what we have developed for "Part 2". However, instead of working with Equation (2), as before, we will sample $\boldsymbol{x}_\star$ from Equation (3), meaning that, for some accepted $\kappa_\star$, we will updated all values of x_i, for $i \in \{1, \cdots, 365\}$, at once. Here, notice that we still have to go through the procedure of accepting $\kappa_\star$ with probability α – it will not be the case for "Part 4.".

Then, in a very similar way as we have done before, we can compute

$$\ln(R) = \ln(\pi(\boldsymbol{x}_\star|\kappa, y)) + \ln(\pi_{\mathrm{G}}(\boldsymbol{x}|\boldsymbol{x}_\star, \kappa, y)) - \ln(\pi(\boldsymbol{x}|\kappa, y)) - \ln(\pi_{\mathrm{G}}(\boldsymbol{x}_\star|\boldsymbol{x}, \kappa, y)).$$

Finally, we can implement the code, which will be very similar to the "Part 2." solution.

```
library(numDeriv)
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
logdensityX_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) - k *
    (X %*% (R) %*% X)))
logX_k = function(X, R, n) return((-0.5) * (X %*% (R) %*% X))
a_values = seq(from = -39, to = 39, by = 0.1)
density = dnorm(a_values)
cumulative_density = cumsum(density)
cumulative_density = cumulative_density/max(cumulative_density)
normal_cdf = splinefun(a_values, cumulative_density)
gX = function(x, y, mm) {
    return(y * log(normal_cdf(x)) + (mm - y) * log(1 - normal_cdf(x)))
}
logDensityY_X = function(x, y, m) return(sum(gX(x, y, m)))
gausian_approx = function(k, X_star, Y, m, n) {
    single_deriv = rep(NA, n)
    double_deriv = rep(NA, n)
    X_star_old = X_star
    for (i in 1:n) {
        single_deriv[i] = grad(gX, x = X_star[i], y = Y[i], mm = m)
        double_deriv[i] = drop(hessian(gX, x = X_star[i], y = Y[i], mm = m))
```

```
    }
    double_deriv_matrix = diag((-1) * double_deriv)
    R_ = k * R + double_deriv_matrix
    b = (single_deriv - (double_deriv * X_star))
    L = t(chol(R_))
    X_star = backsolve(t(L), forwardsolve(L, b))
    for (i in 1:n) {
        if (X_star[i] < -20 || X_star[i] > 7)
            X_star[i] = X_star_old[i]
    }
    return(list(X_star = X_star, R_ = R_, L = L, b = b, derivative = double_deriv))
}
main = function(k, X, Y, R, m, n) {
    value = TRUE
    count = 0
    p1 = c()
    p2 = c()
    p3 = c()
    while (value) {
        k_old = k
        k_new = my.scale.proposal(1, F = 2) * k
        k_new_posterior = log(density_k(k_new, 1.55)) + logdensityX_k(k_new,
            X, R, n)
        k_posterior = log(density_k(k, 1.55)) + logdensityX_k(k, X, R, n)
        logR = k_new_posterior - k_posterior
        a = exp(min(0, logR))
        if (runif(1) < a)
            k = k_new
        gausianApprox = gausian_approx(k, X, Y, m, n)
        X_old = X
        X_new = gausianApprox$X_star + backsolve(t(gausianApprox$L), rnorm(n))
        # density x_old
        X_oldDensity = logdensityX_k(k, X_old, R, n) + logDensityY_X(X_old,
```

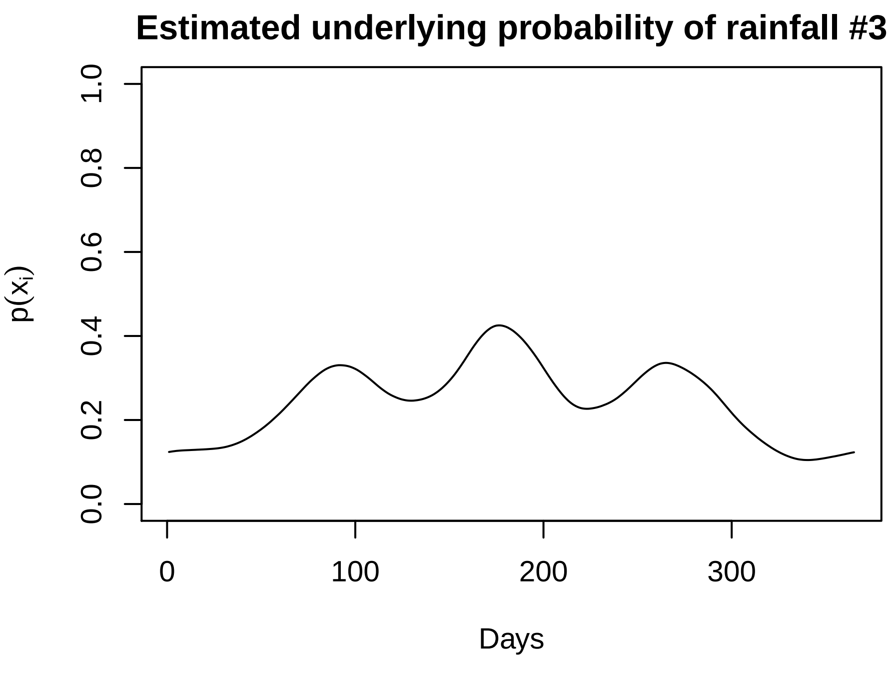
```
        Y, m)
    # density of x_old gaussian approximation
    gausianApprox_new = gausian_approx(k, X_new, Y, m, n)
    X_old_Gausian = (-0.5) * ((X_old - gausianApprox_new$X_star) %*%
        (gausianApprox_new$R_) %*% (X_old - gausianApprox_new$X_star))
    # density x_new
    X_newDensity = logdensityX_k(k, X_new, R, n) + logDensityY_X(X_new,
        Y, m)
    # density of x_new gausian approximation
    X_new_Gausian = (-0.5) * ((X_new - gausianApprox$X_star) %*% (gausianApprox$R_) %*%
        (X_new - gausianApprox$X_star))
    logR = X_newDensity - X_oldDensity + X_old_Gausian - X_new_Gausian
    a = exp(min(0, logR))
    if (runif(1) < a)
        X = X_new
    p1 = append(p1, X[130])
    p2 = append(p2, X[3])
    p3 = append(p3, X[300])
    count = count + 1
    if (count >= 100) {
        X_final = X_final + X
        k_final = k_final + k
    } else {
        X_final = X
        k_final = k
    }
    if (count == 1000) {
        X_final = X_final/900
        k_final = k_final/900
        break
    }
}
return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final))
```

```
}
X = runif(n, 1, 3)
k = 0.7
U = main(k, X, Y, R, m, n)
```

Now, we can load and analyze our results.

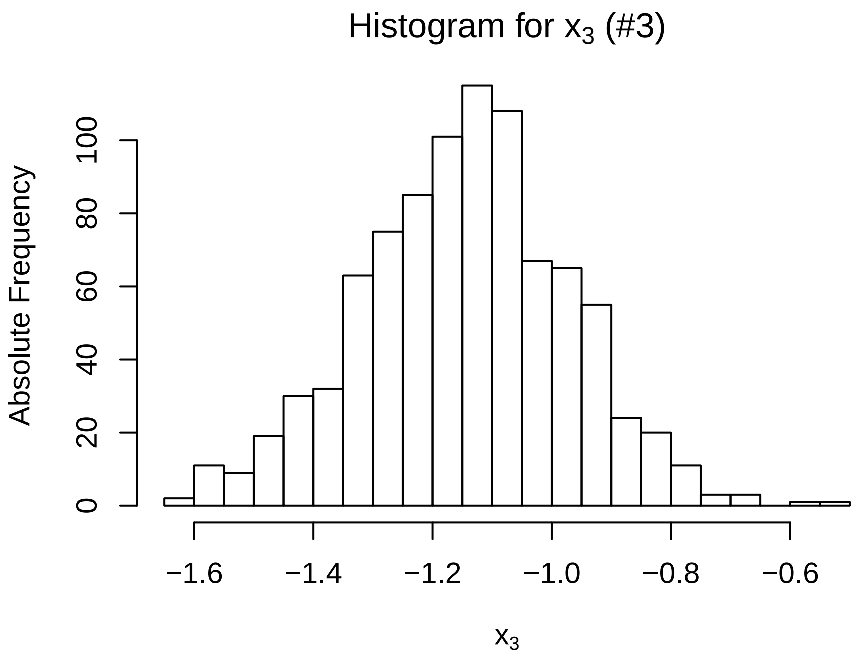
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 364, 365\}$, with a burn-in of size 100 (out of 1,000 observations).

```
load("data/problem3.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #3")
```

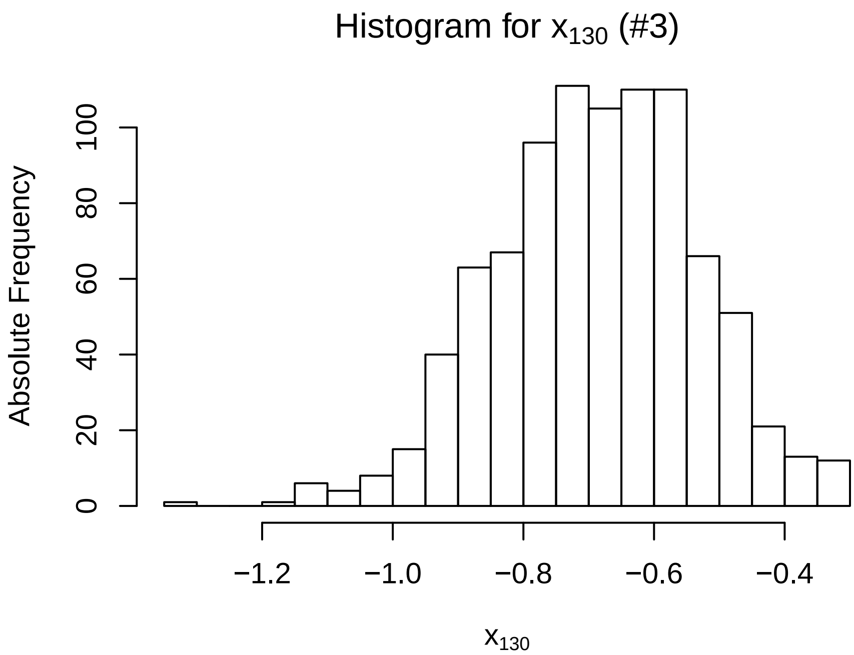


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

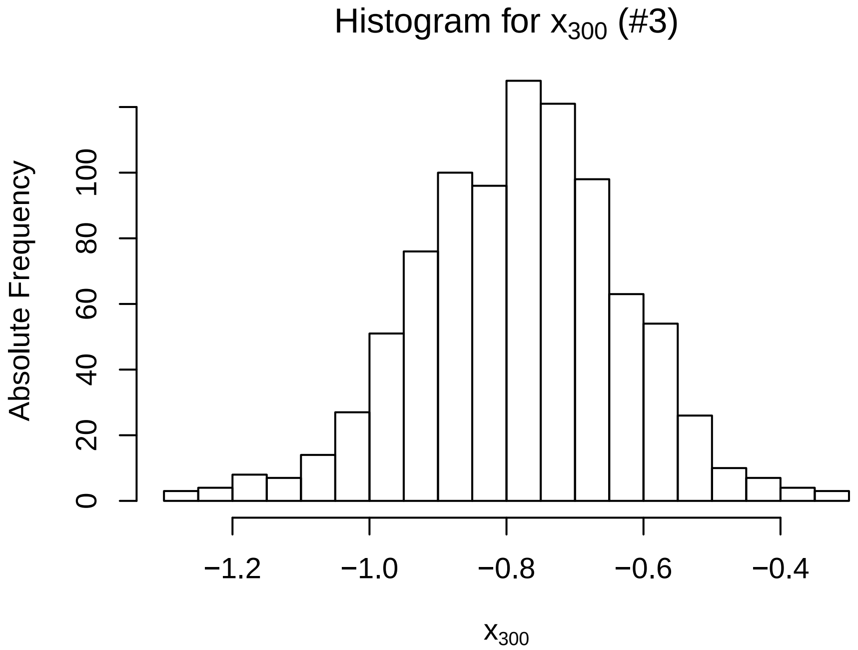
```
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 900), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[3] * " (#3)"))
```




```
hist(tail(U$x130, 900), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[130] * " (#3)"))
```




```
hist(tail(U$x300, 900), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[300] * " (#3)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem3.RData`.

Part 4.

In this fourth part, we will update $\kappa_\star$ and $\boldsymbol{x}_\star$ jointly, which means that we will get some $\kappa_\star = \kappa_\star^{(t+1)} = a \cdot \kappa^{(t)}$, such that $a \sim \pi(a)$, and then compute $\pi_{\mathrm{G}}(\boldsymbol{x}|\kappa_\star, y)$ as before. Once again, $\boldsymbol{x}_\star \sim \pi_{\mathrm{G}}(\boldsymbol{x}|\kappa_\star, y)$. Finally, we can compute

$$\ln(R) = \ln(\pi(\boldsymbol{x}_\star, \kappa_\star|y)) + \ln(\pi_{\mathrm{G}}(\boldsymbol{x}|\boldsymbol{x}_\star, \kappa, y)) - \ln(\pi(\boldsymbol{x}, \kappa|y)) - \ln(\pi_{\mathrm{G}}(\boldsymbol{x}_\star|\boldsymbol{x}, \kappa_\star, y)),$$

such that $\pi(\boldsymbol{x}, \kappa|y) \propto \pi(\kappa) \cdot \pi(\boldsymbol{x}|\kappa) \cdot \pi(y|\boldsymbol{x})$. With these small changes, we can implement a similar code to "Parts 2. and 3.".

```
library(numDeriv)
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
logdensityX_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) - k *
    (X %*% (R) %*% X)))
a_values = seq(from = -39, to = 39, by = 0.1)
density = dnorm(a_values)
cumulative_density = cumsum(density)
cumulative_density = cumulative_density/max(cumulative_density)
normal_cdf = splinefun(a_values, cumulative_density)
gX = function(x, y, mm) {
    return(y * log(normal_cdf(x)) + (mm - y) * log(1 - normal_cdf(x)))
}
logDensityY_X = function(x, y, m) return(sum(gX(x, y, m)))
gausian_approx = function(k, X_star, Y, m, n) {
    single_deriv = rep(NA, n)
    double_deriv = rep(NA, n)
    X_star_old = X_star
    for (i in 1:n) {
        single_deriv[i] = grad(gX, x = X_star[i], y = Y[i], mm = m)
        double_deriv[i] = drop(hessian(gX, x = X_star[i], y = Y[i], mm = m))
    }
    double_deriv_matrix = diag((-1) * double_deriv)
    R_ = k * R + double_deriv_matrix
```

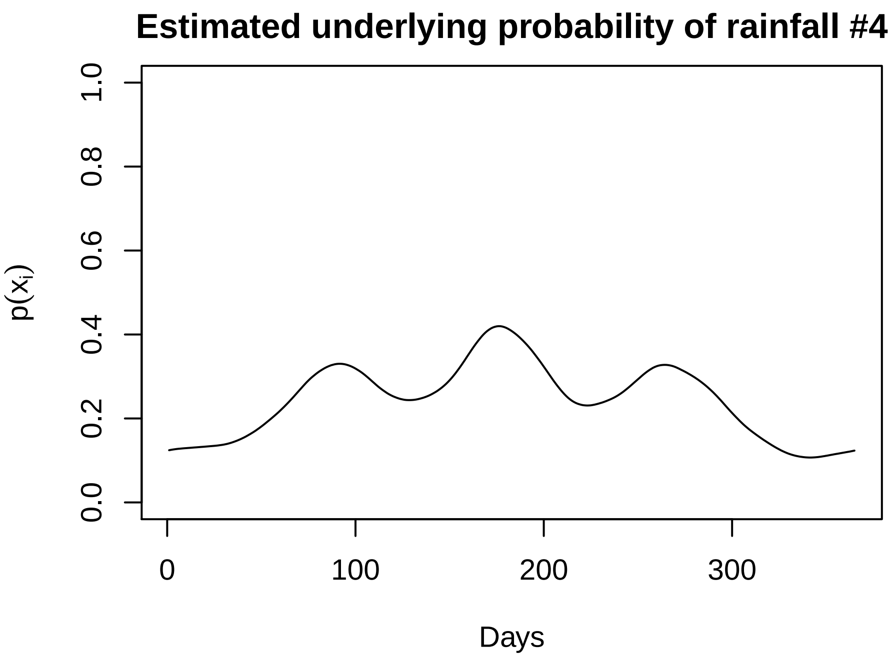
```
    b = (single_deriv - (double_deriv * X_star))
    L = t(chol(R_))
    X_star = backsolve(t(L), forwardsolve(L, b))
    for (i in 1:n) {
        if (X_star[i] < -20 || X_star[i] > 7)
            X_star[i] = X_star_old[i]
    }
    return(list(X_star = X_star, R_ = R_, L = L, b = b, derivative = double_deriv))
}
main = function(k, X, Y, R, m, n) {
    value = TRUE
    count = 0
    p1 = c()
    p2 = c()
    p3 = c()
    while (value) {
        k_new = my.scale.proposal(1, F = 2) * k
        gausianApprox = gausian_approx(k_new, X, Y, m, n)
        X_new = gausianApprox$X_star + backsolve(t(gausianApprox$L), rnorm(n))
        Xk_oldDensity = logdensityX_k(k, X, R, n) + logDensityY_X(X, Y, m) +
            log(density_k(k, 1.55))
        gausianApprox_new = gausian_approx(k, X_new, Y, m, n)
        w = determinant(gausianApprox_new$R_)
        X_old_Gausian = (-0.5) * ((X - gausianApprox_new$X_star) %*% (gausianApprox_new$R_) %*
            (X - gausianApprox_new$X_star)) + (0.5) * w$modulus
        # density x_new
        Xk_newDensity = logdensityX_k(k_new, X_new, R, n) + logDensityY_X(X_new,
            Y, m) + log(density_k(k_new, 1.55))
        # density of x_new gausian approximation
        w = determinant(gausianApprox$R_)
        X_new_Gausian = (-0.5) * ((X_new - gausianApprox$X_star) %*% (gausianApprox$R_) %*%
            (X_new - gausianApprox$X_star)) + (0.5) * w$modulus
        logR = Xk_newDensity - Xk_oldDensity + X_old_Gausian - X_new_Gausian
```

```
        a = exp(min(0, logR))
        if (runif(1) < a) {
            X = X_new
            k = k_new
        }
        p1 = append(p1, X[130])
        p2 = append(p2, X[3])
        p3 = append(p3, X[300])
        count = count + 1
        if (count >= 100) {
            X_final = X_final + X
            k_final = k_final + k
        } else {
            X_final = X
            k_final = k
        }
        if (count == 1000) {
            X_final = X_final/900
            k_final = k_final/900
            break
        }
    }
    return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final))
}
X = runif(n, 1, 3)
k = 0.7
U = main(k, X, Y, R, m, n)
```

Now, we can load and analyze our results.

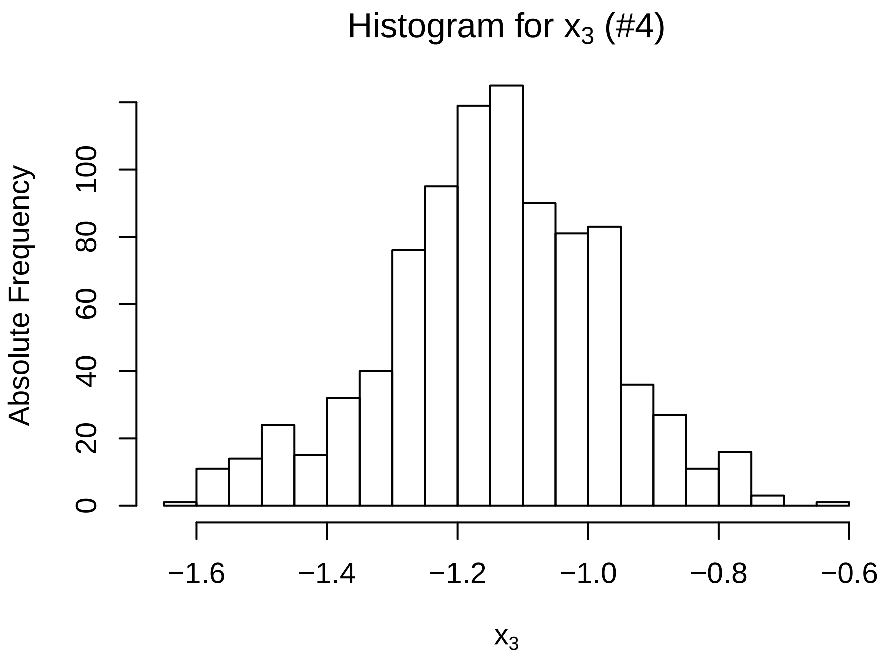
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 364, 365\}$, with a burn-in of size 100 (out of $1,000$ observations).

```
load("data/problem4.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #4")
```

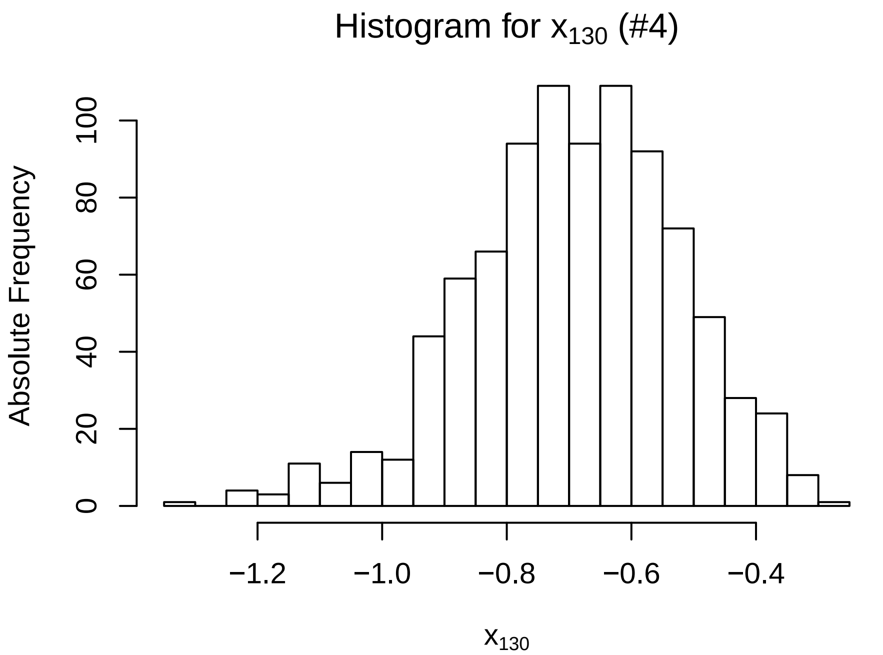


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

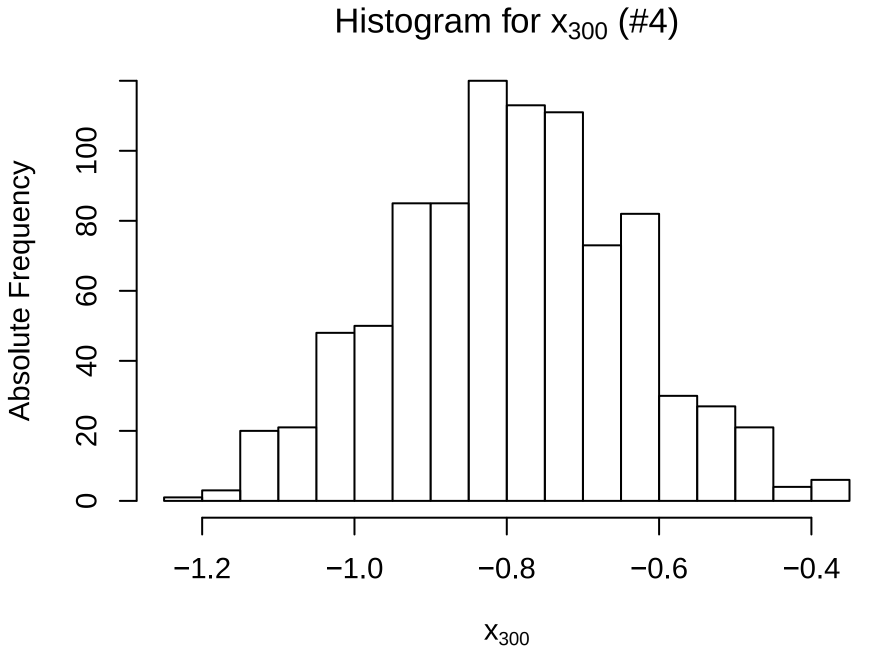
```
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 900), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[3] * " (#4)"))
```




```
hist(tail(U$x130, 900), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[130] * " (#4)"))
```




```
hist(tail(U$x300, 900), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 20, main = expression("Histogram for " * x[300] * " (#4)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem4.RData`.

Part 5. This fifth part (and the next one) will use a slightly different approach if compared to what we have done so far. We will introduce some auxiliary random variables to fit the model.

Let X'_{i1} and X'_{i2} be two auxiliary variables, such that $X'_{ij} = X_i + \epsilon_{ij}$, with $\epsilon_{ij} \sim \text{Normal}(0, 1)$. In this case, for $Y_i = Y_{i1} + Y_{i2}$, we have that

$$Y_{ij} = \begin{cases} 0 & , \text{ if } X'_{ij} \geq 0 \\ 1 & , \text{ if } X'_{ij} < 0 \end{cases}, \text{ for all } j \in \{1, 2\}.$$

Assume for a moment that $y_{i1} = 1$ and $y_{i2} = 0$, then we will have that $\pi(x'_{i1}, x'_{i2} | x_i, y_{i1}, y_{i2}) \propto \exp\left[-\frac{1}{2}(x'_{i1} - x_i)^2 - \frac{1}{2}(x'_{i2} - x_i)^2 \cdot \mathbb{I}_{\{x'_{i1} \geq 0\}} \cdot \mathbb{I}_{\{x'_{i2} < 0\}}\right]$, where $\mathbb{I}_A$ is the indicator function for some set A; i.e., it is a Truncated Normal distribution. Therefore, notice that the previous distribution can be factorized into $\pi(x'_{i1} | x_i, y_{i1}) \cdot \pi(x'_{i2} | x_i, y_{i2})$.

In this case, the κ value will be updated in the same way as the previous parts. The *spline* will be updated, for some accepted *precision* value, following a Normal distribution, and the auxiliary variables will come from a Truncated Normal distribution, as we have just explained.

```
library(truncnorm)
Y1 = rep(NA, n)
Y2 = rep(NA, n)
for (i in 1:n) {
    if (Y[i] == 0) {
        Y1[i] = 0
        Y2[i] = 0
    } else if (Y[i] == 1) {
        Y1[i] = 1
        Y2[i] = 0
    } else {
        Y1[i] = 1
        Y2[i] = 1
    }
}
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
```

```
logdensityEta_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) -
    k * (t(X) %*% (R) %*% X)))
dist_eta = function(k, R, eta1, eta2, n) {
    I = diag(1, n, n)
    Q = k * R + 2 * I
    b = -2 * (t(eta1) %*% I + t(eta2) %*% I)
    L = t(chol(Q))
    mean = backsolve(t(L), forwardsolve(L, t(b)))
    return(list(mean = mean, L = L))
}
truncated_normal = function(eta, Y1, Y2) {
    eta1 = rep(NA, n)
    eta2 = rep(NA, n)
    for (i in 1:n) {
        if (Y1[i] == 1)
            eta1[i] = rtruncnorm(1, b = 0, mean = eta[i], sd = 1)
        if (Y1[i] == 0)
            eta1[i] = rtruncnorm(1, a = 0, mean = eta[i], sd = 1)
        if (Y2[i] == 1)
            eta2[i] = rtruncnorm(1, b = 0, mean = eta[i], sd = 1)
        if (Y2[i] == 0)
            eta2[i] = rtruncnorm(1, a = 0, mean = eta[i], sd = 1)
    }
    return(list(eta1 = eta1, eta2 = eta2))
}
main = function(k, eta, Y1, Y2, R, m, n, eta1, eta2) {
    value = TRUE
    count = 0
    p1 = c()
    p2 = c()
    p3 = c()
    while (value) {
        k_new = my.scale.proposal(1, F = 2) * k
```

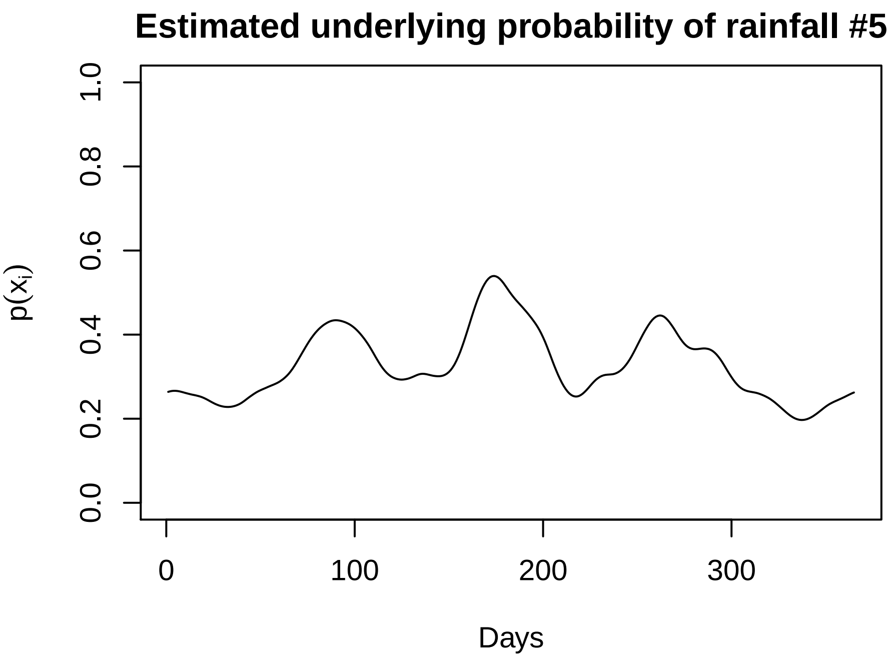
```
        k_old_density = log(density_k(k, 1.55)) + logdensityEta_k(k, eta,
            R, n)
        k_new_density = log(density_k(k_new, 1.55)) + logdensityEta_k(k_new,
            eta, R, n)
        logR = k_new_density - k_old_density
        a = exp(min(0, logR))
        if (runif(1) < a)
            k = k_new
        distbn_eta = dist_eta(k, R, eta1, eta2, n)
        eta = distbn_eta$mean + backsolve(t(distbn_eta$L), rnorm(n))
        auxiliary_eta = truncated_normal(eta, Y1, Y2)
        eta1 = auxiliary_eta$eta1
        eta2 = auxiliary_eta$eta2
        p1 = append(p1, eta[130])
        p2 = append(p2, eta[3])
        p3 = append(p3, eta[300])
        count = count + 1
        if (count >= 300) {
            X_final = X_final + eta
            k_final = k_final + k
        } else {
            X_final = eta
            k_final = k
        }
        if (count == 3000) {
            X_final = X_final/2700
            k_final = k_final/2700
            break
        }
    }
    return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final))
}
eta = runif(n, 1, 3)
```

```r
eta1 = eta + rnorm(n)
eta2 = eta + rnorm(n)
k = 3
U = main(k, eta, Y1, Y2, R, m, n, eta1, eta2)
```

Now, we can load and analyze our results.

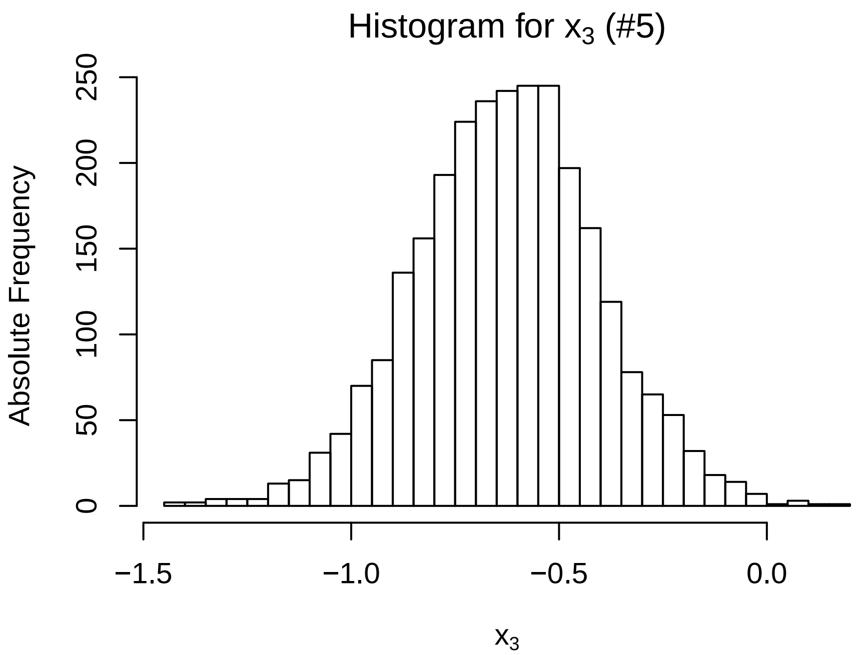
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 364, 365\}$, with a burn-in of size 300 (out of 3,000 observations).

```r
load("data/problem5.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #5")
```

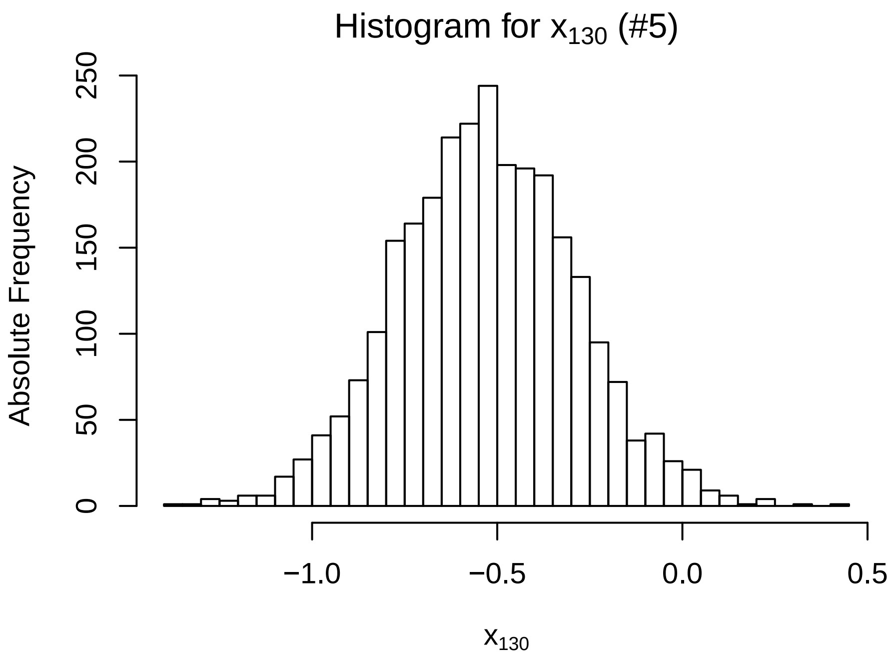


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

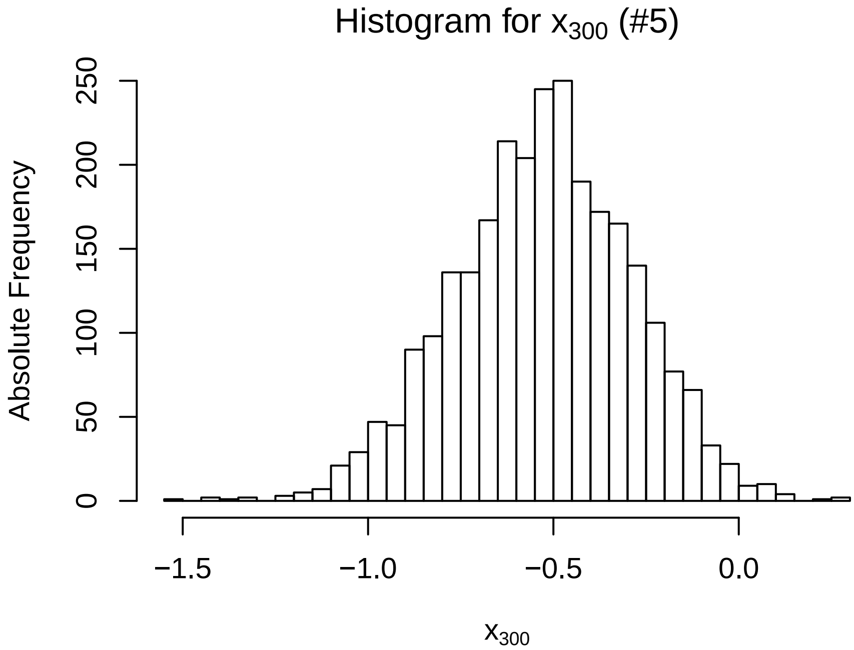
```r
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 2700), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[3] * " (#5)"))
```




```
hist(tail(U$x130, 2700), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[130] * " (#5)"))
```




```
hist(tail(U$x300, 2700), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[300] * " (#5)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem5.RData`.

Part 6. For this final sampler, we want to do something similar to "Part 4", except that we will update it in two blocks: the precision and the spline jointly, and the auxiliary variables. To do so, notice that, we want to compute

$$\begin{aligned}\pi(x, \kappa | x_1', x_2', y) &\propto \pi(x, \kappa, x_1', x_2', y) \\ &= \pi(y | x_1', x_2') \cdot \pi(x_1', x_2' | x) \cdot \pi(x | \kappa) \cdot \pi(\kappa) = h(\kappa, x).\end{aligned}$$

In this case, we will accept $\kappa_\star$ and $x_\star$ with a probability $\alpha = \exp(\min\{0, \ln(R)\})$, such that

$$\ln(R) = \ln(h(\kappa_\star, x_\star)) + \ln(\pi(x|\kappa, x')) - \ln(h(\kappa, x)) - \ln(\pi(x_\star|\kappa_\star, x')).$$

Furthermore, the new $x'_{1\star}$ and $x'_{2\star}$ will be sample from the Truncated Normal distribution, as before.

```
library(mvtnorm)
library(truncnorm)
density_k = function(k, lambda) return(lambda * exp(-lambda/sqrt(k)) * ((k^(-3/2))/2))
logdensityEta_k = function(k, X, R, n) return(0.5 * (((n - 1) * log(k)) -
    k * (X %*% (R) %*% X)))
dist_eta = function(k, R, eta1, eta2, n) {
    I = diag(1, n, n)
    Q = k * R + 2 * I
    b = -(eta1 + eta2)
    L = t(chol(Q))
    mean = backsolve(t(L), forwardsolve(L, b))
    return(list(mean = mean, L = L, Q = Q))
}
truncated_normal = function(eta, Y1, Y2) {
    eta1 = rep(NA, n)
    eta2 = rep(NA, n)
    for (i in 1:n) {
        if (Y1[i] == 1)
            eta1[i] = rtruncnorm(1, b = 0, mean = eta[i], sd = 1)
```

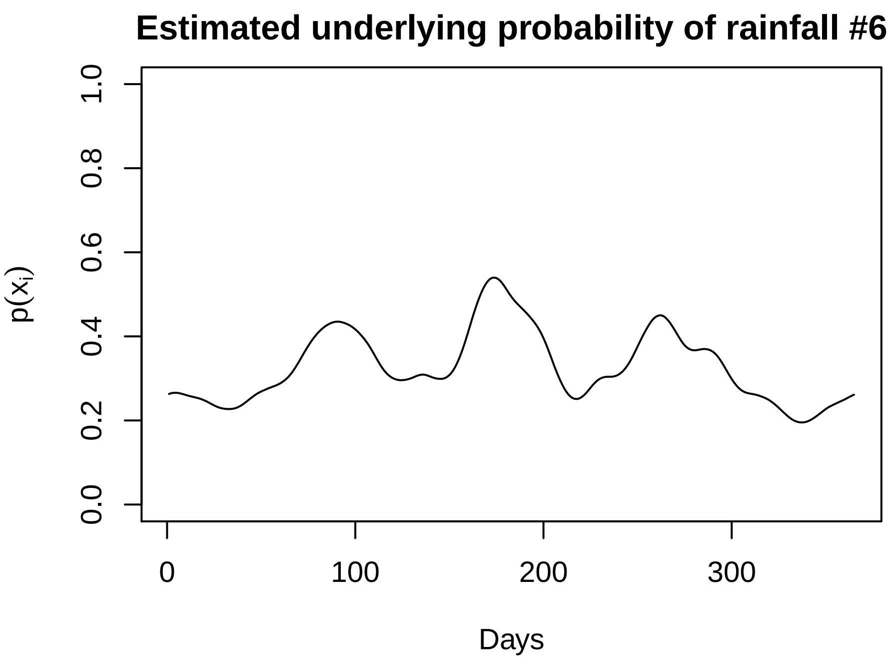
```
        if (Y1[i] == 0)
            eta1[i] = rtruncnorm(1, a = 0, mean = eta[i], sd = 1)
        if (Y2[i] == 1)
            eta2[i] = rtruncnorm(1, b = 0, mean = eta[i], sd = 1)
        if (Y2[i] == 0)
            eta2[i] = rtruncnorm(1, a = 0, mean = eta[i], sd = 1)
    }
    return(list(eta1 = eta1, eta2 = eta2))
}
main = function(k, eta, Y1, Y2, R, m, n, eta1, eta2) {
    value = TRUE
    count = 0
    I = diag(1, n, n)
    p1 = c()
    p2 = c()
    p3 = c()
    while (value) {
        k_new = my.scale.proposal(1, F = 2) * k
        distbn_eta_new = dist_eta(k_new, R, eta1, eta2, n)
        eta_new = distbn_eta_new$mean + backsolve(t(distbn_eta_new$L), rnorm(n))
        etak_old_density = log(density_k(k, 1.55)) + logdensityEta_k(k, eta,
            R, n) - 0.5 * ((sum((eta1 - eta)^2) + sum((eta2 - eta)^2)))
        etak_new_density = log(density_k(k_new, 1.55)) + logdensityEta_k(k_new,
            eta_new, R, n) - 0.5 * (sum((eta1 - eta_new)^2) + sum((eta2 -
            eta_new)^2))
        distbn_eta_old = dist_eta(k, R, eta1, eta2, n)
        w1 = determinant(distbn_eta_old$Q)
        proposal_old = (-0.5) * ((eta - distbn_eta_old$mean) %*% (distbn_eta_old$Q) %*%
            (eta - distbn_eta_old$mean)) + (0.5) * w1$modulus
        w2 = determinant(distbn_eta_new$Q)
        proposal_new = (-0.5) * ((eta_new - distbn_eta_new$mean) %*% (distbn_eta_new$Q) %*%
            (eta_new - distbn_eta_new$mean)) + (0.5) * w2$modulus
        logR = etak_new_density - etak_old_density + proposal_old - proposal_new
```

```
        a = exp(min(0, logR))
        if (runif(1) < a) {
            k = k_new
            eta = eta_new
        }
        auxiliary_eta = truncated_normal(eta, Y1, Y2)
        eta1 = auxiliary_eta$eta1
        eta2 = auxiliary_eta$eta2
        p1 = append(p1, eta[130])
        p2 = append(p2, eta[3])
        p3 = append(p3, eta[300])
        count = count + 1
        if (count >= 300) {
            X_final = X_final + eta
            k_final = k_final + k
        } else {
            X_final = eta
            k_final = k
        }
        if (count == 3000) {
            X_final = X_final/2700
            k_final = k_final/2700
            break
        }
    }
    return(list(x130 = p1, x3 = p2, x300 = p3, k = k_final, X = X_final))
}
eta = runif(n, 1, 3)
eta1 = eta + rnorm(n)
eta2 = eta + rnorm(n)
k = 4
U = main(k, eta, Y1, Y2, R, m, n, eta1, eta2)
```

Now, we can load and analyze our results.

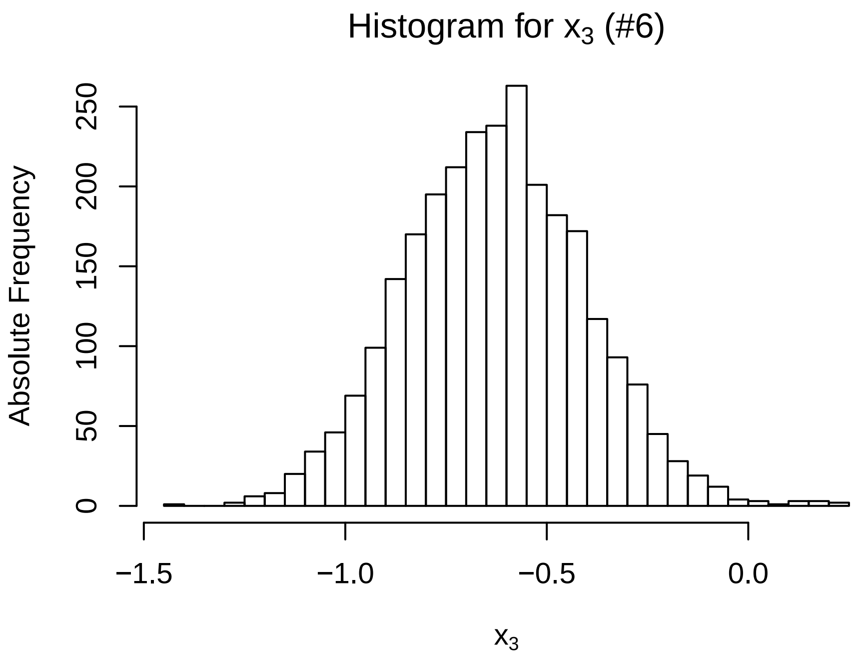
Let's start with the estimated probability of rainfall along the 365 days, based on the estimated x_i, for $i \in \{1, 2, \cdots, 364, 365\}$, with a burn-in of size 300 (out of 3,000 observations).

```
load("data/problem6.Rdata")
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
plot(pnorm(U$X), type = "l", ylim = c(0, 1), xlab = "Days", ylab = expression(p(x[i])),
    main = "Estimated underlying probability of rainfall #6")
```

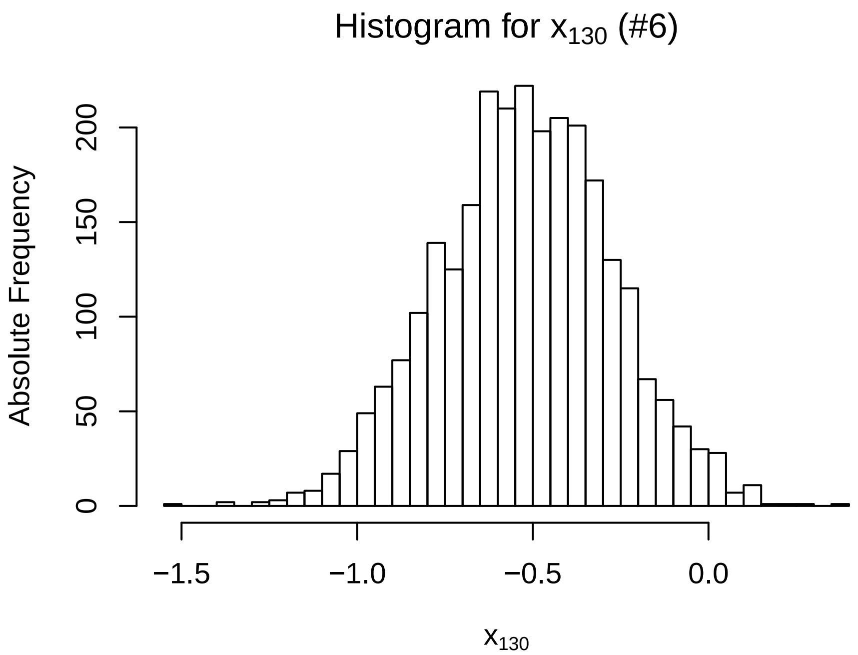


Now, let's take a look at three particular days, and plot the sample we have obtained from them.

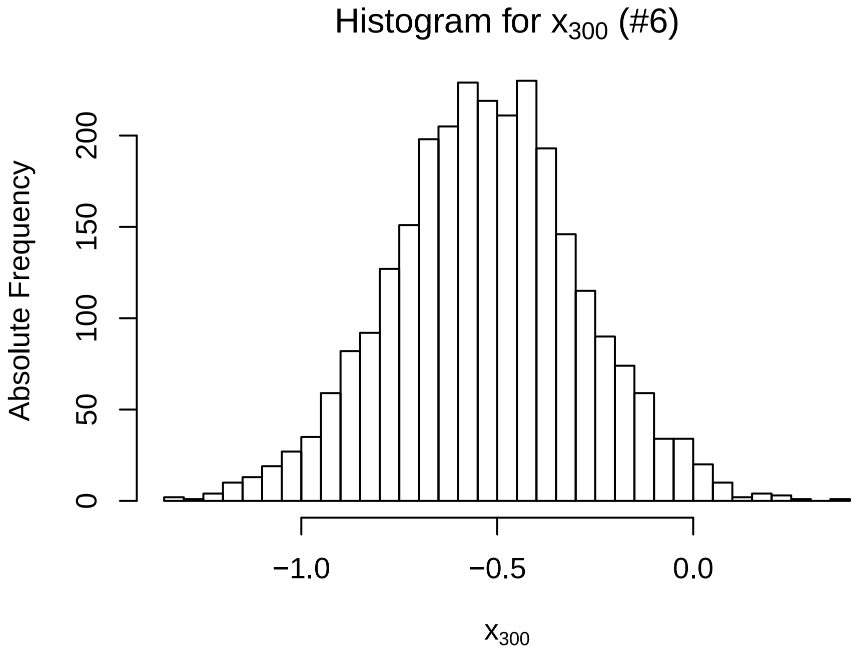
```
par(mar = c(4.1, 4.1, 2.4, 2.1), cex = 0.875)
hist(tail(U$x3, 2700), xlab = expression(x[3]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[3] * " (#6)"))
```




```
hist(tail(U$x130, 2700), xlab = expression(x[130]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[130] * " (#6)"))
```




```
hist(tail(U$x300, 2700), xlab = expression(x[300]), ylab = "Absolute Frequency",
    breaks = 50, main = expression("Histogram for " * x[300] * " (#6)"))
```



For reference, the data set that we used to generate these plot is in `/data/problem6.RData`.