Statistical Programming — Worksheet 6

1. Numbers. Dr. Winkel has 200 square tiles with which to decorate a wall of the kitchen in the Department of Statistics. 20 of the tiles are red, 30 blue, and the rest are white. Write down a formula for the number of distinct patterns he can create.

This is

$$\frac{200!}{150! \cdot 30! \cdot 20!}$$

How many digits does this number have? 200! *is too large for R to calculate, so use* `lfactorial()`*:*

```
exp(lfactorial(200) - lfactorial(150) - lfactorial(20) - lfactorial(30))

[1] 2.138999e+61
```

giving 2.1×10^{61}*, so there are 62 digits in the number. (You can also do this by replacing* $200!/150!$ *with* $151 \times \cdots \times 200$*.)*

How many digits does 1000! have? *To do this we need the number in base 10, so divide the natural log by* $\log(10)$*.*

```
lfactorial(1000)/log(10)

[1] 2567.605
```

so (rounding up) gives 2568 digits.

2. Metropolis Hastings. Suppose that $X_1, \ldots, X_n \overset{\text{i.i.d.}}{\sim} \text{Gamma}(\alpha, \beta)$, and let α and β have independent Exponential(1) priors.

(a) Write a function to evaluate the log-posterior of α and β given a vector of data $\boldsymbol{x}$. The function should have arguments `x`, `alpha` and `beta`.

```
## Log-Posterior distribution for Gamma x : vector of data alpha, beta :
## parameter values
log_post <- function(x, alpha, beta) {
    log_prior <- dexp(alpha, 1, log = TRUE) + dexp(beta, 1, log = TRUE)
    log_lik <- sum(dgamma(x, alpha, beta, log = TRUE))
    out <- log_prior + log_lik
    return(out)
}
```

(b) Write a function to perform a single Metropolis-Hastings step to explore the posterior above. Use a proposal

$$\alpha' = \alpha + \sigma Z_1 \qquad \beta' = \beta + \sigma Z_2$$

for Z_1, Z_2 independent standard normals (i.e. $q(\alpha' \mid \alpha) \sim N(\alpha, \sigma^2)$.) It should take as arguments `x`, `alpha`, `beta` and `sigma`.

```
## Function to perform one step of M-H x : vector of data alpha, beta :
## current parameter values sigma : s.d. of proposal distribution
mh_step <- function(x, alpha, beta, sigma) {
    new_pt <- c(alpha, beta) + rnorm(2, sd = sigma)
    if (any(new_pt <= 0))
        return(c(alpha, beta))  # proposed point not valid

    U <- runif(1)
    logalpha <- log_post(x, new_pt[1], new_pt[2]) - log_post(x, alpha, beta)

    if (log(U) < logalpha)
        return(new_pt) else return(c(alpha, beta))
}
```

(c) Write a function to run the Metropolis-Hastings algorithm for N steps and return an $N \times 2$ matrix of the parameter values. It should take as input the data `x`, number of steps `N`, starting values `alpha` and `beta`, and proposal standard deviation `sigma`.

```
## Function to perform the M-H x : vector of data N : number of samples
## alpha, beta : starting values sigma : s.d. of proposal distribution
run_mh <- function(x, N, alpha = 1, beta = 1, sigma = 2/sqrt(length(x))) {
    params <- matrix(0, N, 2)
    cur = c(alpha, beta)

    # iterate through MH steps
    for (i in 1:N) {
        cur = mh_step(x, cur[1], cur[2], sigma = sigma)
        params[i, ] = cur
    }

    return(params)
}
```

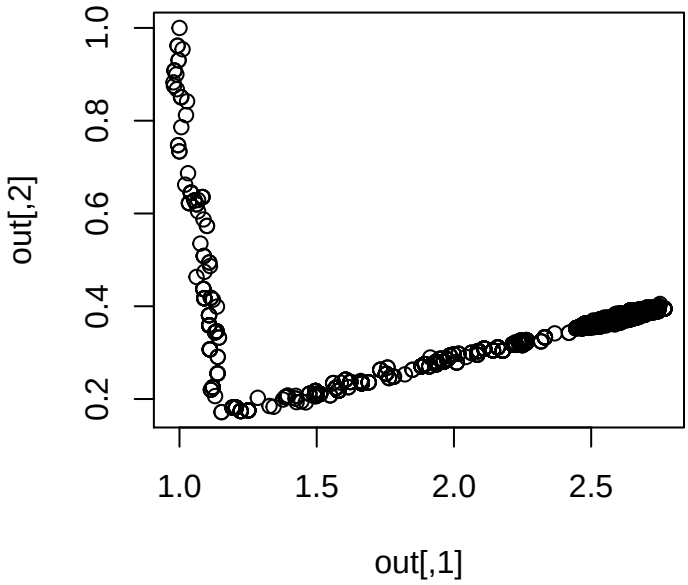
(d) The file `airpol.txt` (on the class website) contains daily PM2.5 readings taken from various measuring stations around Seattle during 2015. Read in the data as a vector and plot it in a histogram.

```
x <- scan("airpol.txt")  # note use of scan(), not read.table()
hist(x, breaks = 100, freq = FALSE)
```

Model the data as i.i.d. Gamma distributed observations using the model above. Run your Metropolis-Hastings algorithm for 5,000 steps with starting point $\alpha = 1$, $\beta = 1$. Plot your output with `plot()` and investigate different values of $\sigma \in \{0.01, 0.02, 0.05\}$.

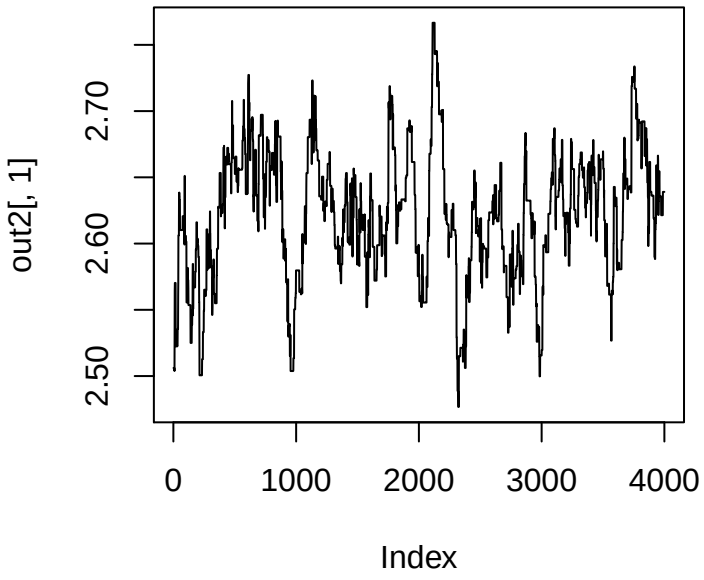
There is a compromise in how often we want the chain to move and how quickly it should be able to explore the space. 0.01 moves slowly, 0.05 rarely moves at all, but 0.02 works quite well.

```
out <- run_mh(x, 5000, alpha = 1, beta = 1, sigma = 0.02)
plot(out)
```

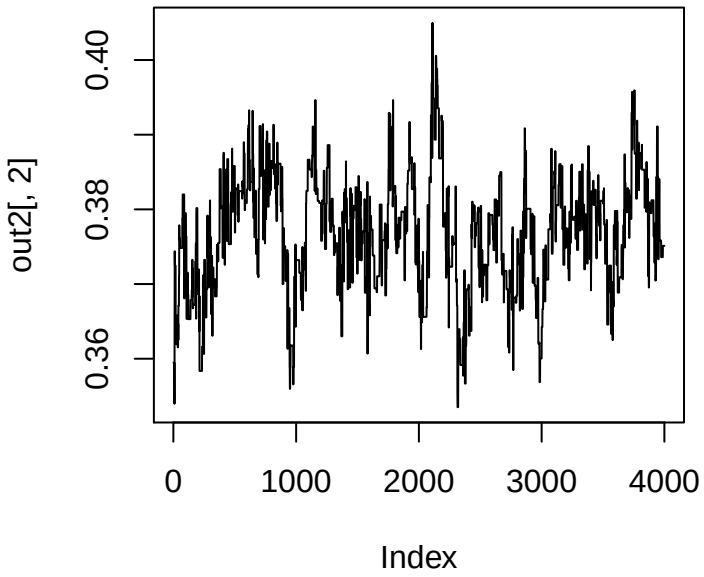


Of course, the chain takes some time to settle down (burn-in), so it is sensible to discard the first 1,000 or so observations for inference. This gives a clearer picture.

```
out2 <- out[-c(1:1000), ]
plot(out2[, 1], type = "l")
```

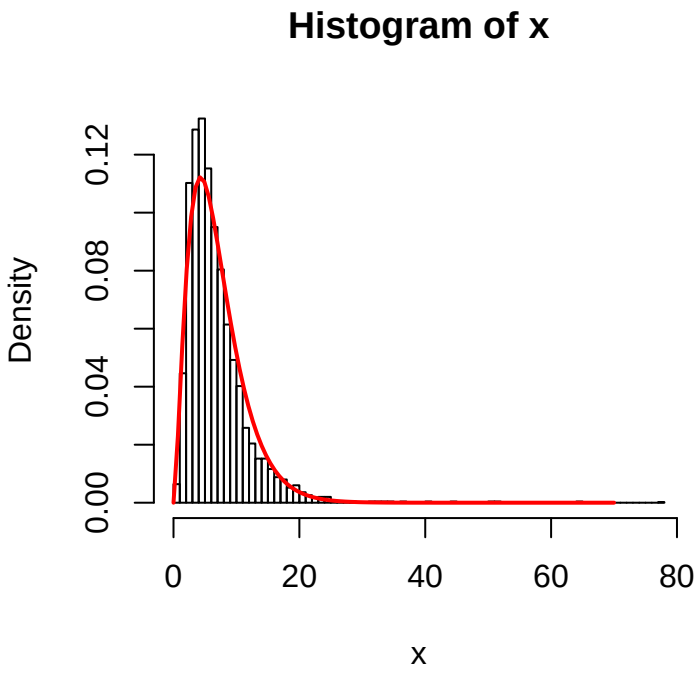



```
plot(out2[, 2], type = "l")
```



(e) Find the posterior means for α and β. Plot the density of the corresponding Gamma distribution over the histogram of the data.

```
hist(x, breaks = 100, freq = FALSE)
alphahat <- mean(out2[, 1])
betahat <- mean(out2[, 2])
f <- function(y) dgamma(y, alphahat, betahat)
plot(f, 0, 70, add = TRUE, col = 2, lwd = 2)
```



This looks like a reasonable fit, although actually the outliers are probably not modelled well by this distribution.

3. Image Reconstruction. Let the $n \times n$ matrix $Y = (y_{ij})$ of ± 1s follow the distribution of the Ising model with parameter θ, so that

$$\pi(Y) \propto \exp\left\{\theta \sum_{(i,j)\sim(i'j')} y_{ij}y_{i'j'}\right\}$$

where $(i,j) \sim (i'j')$ if either $i = i' \pm 1$ and $j = j'$, or vice versa (i.e. they differ by exactly one column or one row, but not both).

(a) Let $\widetilde{Y} = Y$ except that $\widetilde{y}_{ij} = 1 - y_{ij}$ (so they are equal except for a single entry). Show that

$$\log \pi(\widetilde{Y}) - \log \pi(Y) = \theta(d_{i,j} - 2a_{i,j})$$

where $d_{i,j}$ is the number of pixels adjacent to i,j, and $a_{i,j}$ is the number of adjacent pixels which have the same value as y_{ij}.

We will construct a Metropolis-Hasting algorithm to target π.

(b) First, look at the function `mh_step()` in the file `MHcode.R` on the website. The function performs one M-H step by proposing to flip `Y[r,c]`.

Complete the function by replacing the questions marks with code to calculate $\log \alpha$. Comment the code to show you understand what the rest of the function is doing.

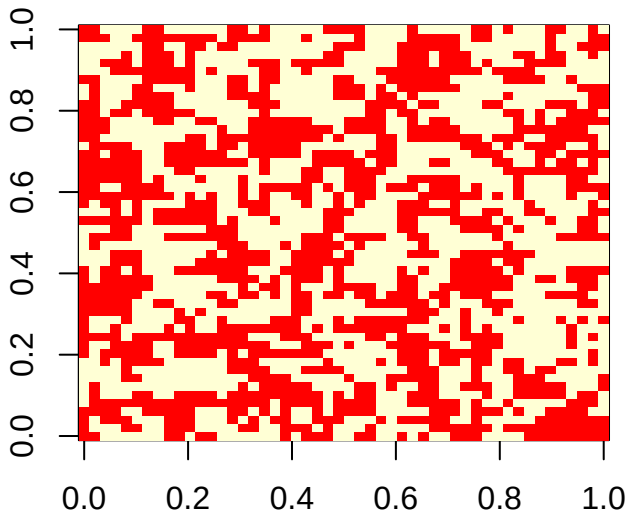
```
## Perform one M-H step Y - current state of chain r,c - row/column of bit to
## try to flip
mh_step <- function(Y, r, c, theta) {

    ## work out indicies of adjacent rows and columns
    r_adj = r + c(-1, 1)  # one row above and below
    c_adj = c + c(-1, 1)  # columns

    ## if on the boundary then adjust indices appropriately
    if (r == 1)
        r_adj = r_adj[-1] else if (r == nrow(Y))
        r_adj = r_adj[-2]
    if (c == 1)
        c_adj = c_adj[-1] else if (c == ncol(Y))
        c_adj = c_adj[-2]

    ## change in log(pi)
    n_agree = sum(Y[r, c] == c(Y[r_adj, c], Y[r, c_adj]))
    n_nbrs = length(r_adj) + length(c_adj)
    log_alpha = theta * (n_nbrs - 2 * n_agree)

    ## if U < acceptance ratio then flip entry otherwise keep as is
    if (log(runif(1)) < log_alpha) {
        Y[r, c] = 1 - Y[r, c]  # flip this bit
    }

    return(Y)
}
```

(c) Now create a function with arguments `n`, `N` and `theta` which creates an $n \times n$ matrix with random entries 0 or 1, and then performs N M-H steps by calling `mh_step()`. When finished, it should return the state of the chain.

```
mh_ising <- function(n, theta = 0.8, N = 1e+05) {
    ## generate totally random matrix to start
    X <- matrix(rbinom(n^2, 1, 0.5), n, n)

    for (i in seq_len(N)) {
        ## generate random row and column
        r <- sample(n, 1)
        c <- sample(n, 1)

        ## perform M-H step on (r,c)
        X <- mh_step(X, r, c, theta = theta)
    }

    X
}
```

(d) Run the function for $n = 50$ and values $\theta = 0.2, 0.5, 0.8$ (you'll probably need $N > 10^5$ to get reasonable convergence). You can plot your solution using the `image()` function:

```
> out <- mh_ising(50, theta=0.5, N=1e5)
> image(out)
```



(e) Consider an $n \times n$ matrix $X = (x_{ij})$ of independent Bernoulli random variables, where

$$P(x_{ij} = 1) = \begin{cases} 1 - p & \text{if } y_{ij} = 0 \\ p & \text{if } y_{ij} = 1 \end{cases}$$

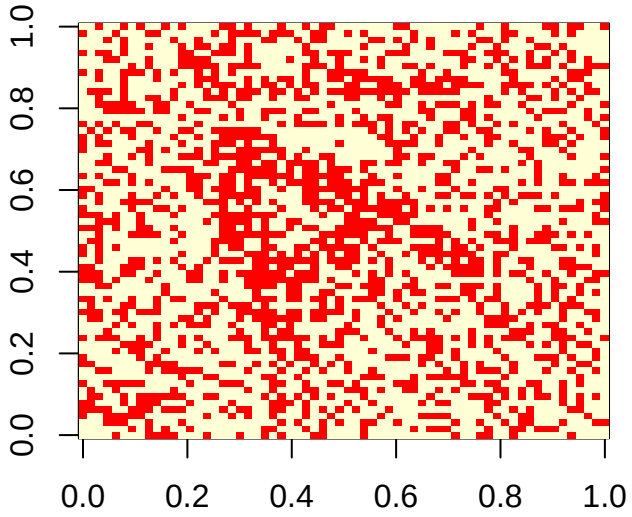
for an unknown matrix of numbers $Y = (y_{ij})$. Defining $\widetilde{Y}$ as in (a), show that

$$\log L(\widetilde{Y}; X) - \log L(Y; X) = \begin{cases} +\log \frac{p}{1-p} & \text{if } y_{ij} \neq x_{ij} \\ -\log \frac{p}{1-p} & \text{if } y_{ij} = x_{ij} \end{cases},$$

where $L(Y; X)$ is the likelihood for the unknown parameter Y given X.

(f) Read in the data and look at it:

```
X <- as.matrix(read.table("image_noisy.txt"))
image(X)
```



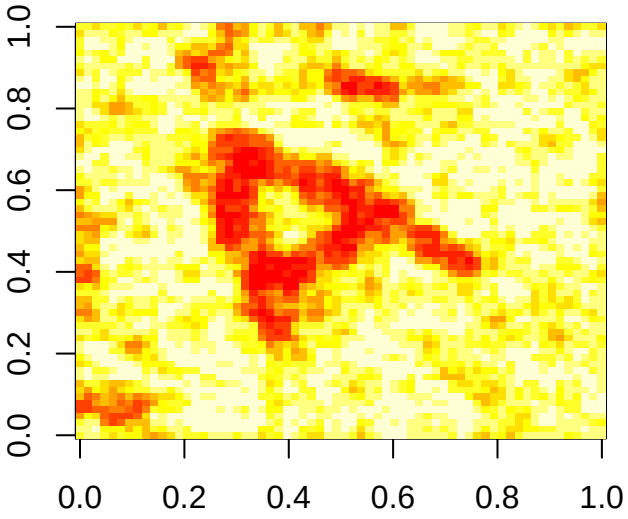
Modify your previous M-H functions to accept a matrix X of data as an argument, and to include the change in the likelihood in your acceptance ratio α. Have the function return the estimated posterior mean of the chain (i.e. the average position of each pixel over the iterations).

```
## Perform one M-H step Y - current state of chain X - data r,c - row/column
## of bit to try to flip theta,p - parameters
mh_step2 <- function(Y, X, r, c, theta, p) {

    ## work out indicies of adjacent rows and columns
    r_adj = r + c(-1, 1)  # one row above and below
    c_adj = c + c(-1, 1)  # columns

    ## if on the boundary then adjust indices appropriately
    if (r == 1)
        r_adj = r_adj[-1] else if (r == nrow(Y))
        r_adj = r_adj[-2]
    if (c == 1)
        c_adj = c_adj[-1] else if (c == ncol(Y))
        c_adj = c_adj[-2]
```

```
    ## change in log-likelihood
    n_agree = sum(Y[r, c] == c(Y[r_adj, c], Y[r, c_adj]))
    n_nbrs = length(r_adj) + length(c_adj)
    ch_loglik = theta * (n_nbrs - 2 * n_agree)

    ## change in log-prior
    if (X[r, c] == Y[r, c]) {
        ch_lprior = -log(p/(1 - p))
    } else {
        ch_lprior = log(p/(1 - p))
    }

    ## log-acceptance ratio
    log_alpha <- ch_lprior + ch_loglik

    ## if U < acceptance ratio then flip entry otherwise keep as is
    if (log(runif(1)) < log_alpha) {
        Y[r, c] = 1 - Y[r, c]  # flip this bit
    }

    return(Y)
}

## perform N Metropolis-Hastings steps and return the mean for each pixel X -
## data N - number of steps to make theta,p - parameters
mh_denoise <- function(X, N = 1e+05, theta = 0.8, p = 2/3) {
    ## matrix to record results
    n <- nrow(X)
    Y <- matrix(rbinom(n^2, 1, 0.5), n, n)
    out <- matrix(0, n, n)

    for (i in seq_len(N)) {
        ## generate random row and column
        r <- sample(n, 1)
        c <- sample(n, 1)

        ## perform M-H step on (r,c)
        Y <- mh_step2(Y, X, r, c, theta = theta, p = p)

        ## add to total
        out <- out + Y
    }

    return(out/N)
}
```

Run the chain for a million iterations, setting $p = \frac{2}{3}$ and $\theta = 0.8$, and plot the results.

```
smoothed <- mh_denoise(X, N = 1e+06)
image(smoothed)
```


0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0