Problem 2

(a)

Considering the adaptive rejection sampling (ARS) algorithm, we build a set of p points, $x_1, ..., x_p$ and an upper envelope function that is linear on the log scale withing each interval, i.e. $f(x) = exp(a_i + b_i x)$ for $x \in (x_{i-1}, x_i)$

We need to normalize this exponentiated upper envelop function in order to have a density of interest that we can draw samples from.
The normalization factor N_f is constant,
$N_f = \sum_{i=1}^{p} \int_{x_{i-1}}^{x_i} exp(a_i + b_i x)dx = \sum_{i=1}^{p} \frac{e^{a_i}}{b_i}[e^{b_i x_i} - e^{b_i x_{i-1}}]$.
The probability associated with each interval P_i is,
$P_i = Prob(x \in (x_{i-1}, x_i)) = \frac{1}{N_f} \int_{x_{i-1}}^{x_i} exp(a_i + b_i x)dx = \frac{e^{a_i}}{N_f b_i}[e^{b_i x_i} - e^{b_i x_{i-1}}]$.

(b)

We want to draw x from the ith interval using the inverse CDF method.
The CDF function $F(x')$ is,
$F(x') = Prob(x \in (x_{i-1}, x')) = \frac{1}{N_f} \int_{x_{i-1}}^{x'} exp(a_i + b_i x)dx = \frac{e^{a_i}}{N_f b_i}[e^{b_i x'} - e^{b_i x_{i-1}}]$
We will draw $Z \sim Unif(0,1)$ and set $x = F^{-1}(z)$ for ith interval to get $X \sim F$,
$x = F^{-1}(z) = \frac{1}{b_i}\{log[N_f b_i z + exp(a_i + b_i x_{i-1})] - a_i\}$

(c)

The psuedo-code style software that would implement ARS is listed below.

```
# Problem 5.2: ARS [Reference]
# http://www.amsta.leeds.ac.uk/~wally.gilks/adaptive.rejection/web_page/Welcome.html

# calculate the intercept and slope of the line set by succecutive points
lineCoef <- function(x, y) {
    # x in ascending order
    return(data.frame(intercept = a, slope = b))
}

# calculate the intersection points above density curve given the input
# points on the density curve
intersec <- function(x, y) {
    # x and y are the coordinates of the n points
    # get line info for n-1 intersection points
    ab <- lineCoef(x = x, y = y)
    # extrapolate intersection points outside the curve
    qx[i] <- (a[i + 1] - a[i - 1])/(b[i - 1] - b[i + 1])
    qy[i] <- a[i - 1] + b[i - 1] * qx[i]
    return(cbind(qx, qy))
}

# get the set of points that construct the envelop
envelop <- function(x, y) {
    # x and y are input points on the curve
    Q <- intersec(x, y)  # all the upper points
    T <- PQ[order(PQ)]  # in order
    return(data.frame(T))
}

# function that initializes input to sampling algorithm; The inner and env
# element include the points and lines info for the inner chords and
# envelopes respectively.  The region element includes the line info for
# each region that is needed to be calculated
init <- function(x, y) {
    # order x and y in the increasing order of x
    inner <- list(pts = data.frame(x = x, y = y), line = lineCoef(x, y))
    env <- list(pts = envelop(x, y), line = lineCoef(xy.env[1], xy.env[2]))
    reg <- rbind(ab.inner, ab.env)
    return(list(inner = inner, env = env, region = ab.reg))
}

# function to compute the areas of the envelop
area <- function(pts, lb = NULL, rb = NULL) {
    # init from initialization
    # evaluation regions and points
    reg <- pts$region
    x <- pts$env$pts$x
    # calculate the area under the envelope
    ar[i] <- (exp(a + b * x[i + 1]) - exp(a + b * x[i]))/b
    # normalize area and compute cumulatives
    sum.ar <- sum(ar)  # raw total area
    ar <- ar/sum.ar  # normalized - sum to 1
    cum.ar <- cumsum(ar)  # cummulative
    return(list(ar = ar, cum.ar = cum.ar, sum.ar = sum.ar))
}

# Draw sample from envelope
sample.env <- function(pts, lb = NULL, rb = NULL) {
    # calculate the area under the envelope
    ar <- area(pts, lb = lb, rb = rb)
    reg <- pts$region
    x <- pts$env$pts$x
    # line coefficients for that region
    a <- reg[rn, 1]
    b <- reg[rn, 2]
    # sample a point from the envelope using inversion method
    u <- runif(1)
    # invert cumulative df to get sample point
    sx <- (log(u * b * ar$sum.ar + exp(b * x[rn] + a)) - a)/b
    return(sx)
}

# test whether the sample should be accepted or not and store the
# evaluation information of the target function
test.sample <- function(sx, pts, func) {
    x <- pts$env$pts$x
    u <- runif(1)
    if (u < exp(r[1] - r[2])) {
```

```r
        # simple acceptance step
        accept <- 1
    } else {
        # evaluation and rejection step
        if (u < exp(func(sx) - r[2])) {
            accept <- 1
        }
        if (!(sx %in% x)) {
            # updating step: update pts
            x <- c(x, sx)
            y <- c(y, sy)
            pts <- init(x, y)
        }
    }
    return(list(accept = accept, pts = pts, xeval = x))
}

# main function to implement the ARS algorithm; n: number of samples
# wanted; func: the log density; xinit: initial values for the X's, at
# least 3; lb: left bound; rb: right bound; The output is a list of 4
# elements; sample: the sample values; xeval: the new x's that cause the
# func to be evaluated; pts: the points and line info for the chords and
# envelop;
ars <- function(n, func, xinit, lb = NULL, rb = NULL) {
    # initialize
    s <- c()
    yinit <- func(xinit)
    pts <- init(xinit, yinit)
    # sample
    while (length(s) < n) {
        sx <- sample.env(pts, lb = lb, rb = rb)
        tsx <- test.sample(sx, pts, func)
        if (tsx$accept)
            s <- c(s, sx)
        pts <- tsx$pts
        xeval <- tsx$xeval
    }
    return(list(sample = s, xeval = xeval, pts = pts))
}
```

Problem 3

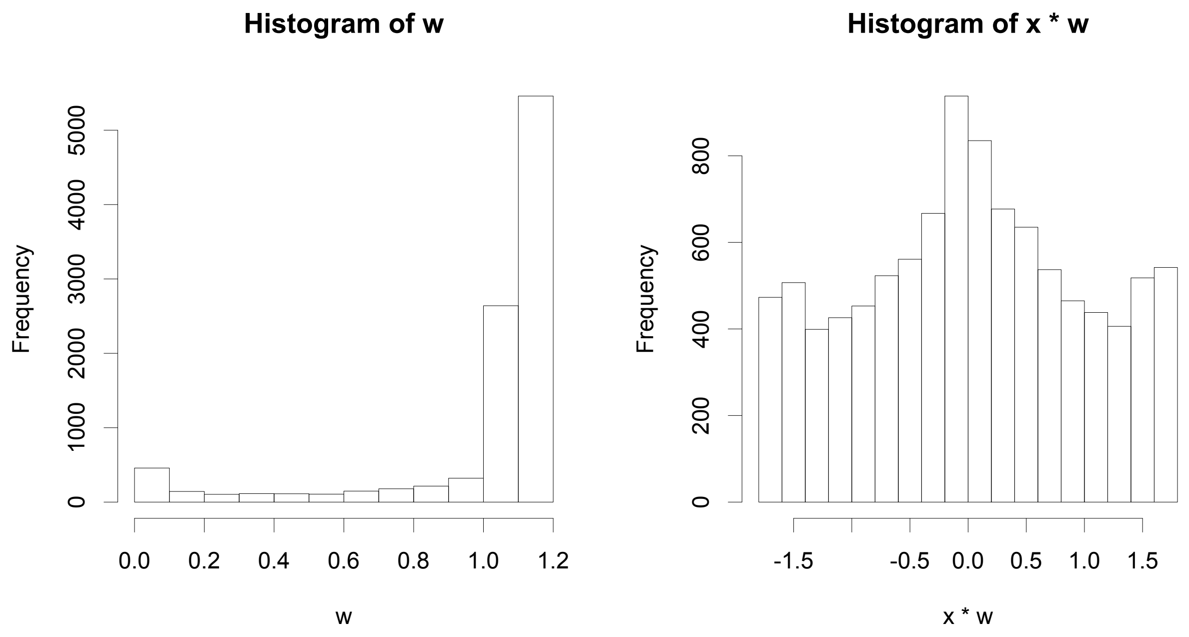
We will explore the need in importance sampling that the sampling density have heavier tails than the density of interest. We will estimate EX and $E(X^2)$ with respect to density f in this problem.

(a)

Suppose $f \sim N(0,1)$ with sampling density $g \sim t(df = 3)$. We sample $m = 10000$ points to extract histograms of estimates and weights in order to get an idea whether $Var(\hat{\mu})$ is large.

As you can see from the results shown below, the variance is not so large since there are not many extreme weights in the samples.

```
## (a) we want to estimate EX and VX for a standrad normal drawn from a t
## with 3 degrees of freedom
m <- 10000  # number of samples for each estimator
set.seed(0)
x <- rt(m, df = 3)  # i.e sample from g being a t with df=3
f <- dnorm(x)  # density of x under f
g <- dt(x, df = 3)  # density of x under g
w <- f/g  # weights
mean(x * w)  #EX

## [1] 0.02107

mean((x * w)^2)  #VX

## [1] 0.9225


par(mfrow = c(1, 2), cex = 10, cex.main = 1.2)
hist(w)
hist(x * w)
```

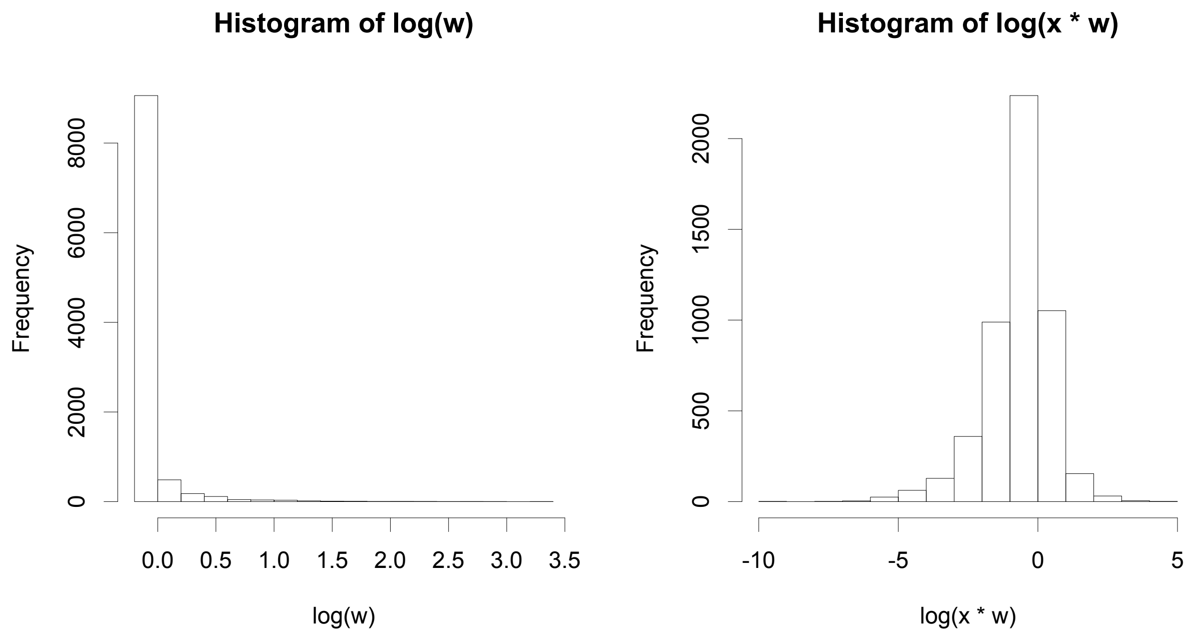


(b)

Suppose $f \sim t(df = 3)$ with sampling density $g \sim N(0, 1)$. We sample $m = 10000$ points to extract histograms of estimates and weights in order to get an idea whether $Var(\hat{\mu})$ is large.

As the sampling density $t(df = 3)$ has less heavier tails than the density of interest $N(0, 1)$, there are many extreme weights that have a very strong influence on the estimation variance, as shown below.

```
## (b) we want to estimate EX and EX^2 for a t with v=3 drawn from a
## standard normal
m <- 10000  # number of samples for each estimator
set.seed(0)
x <- rnorm(m)  # i.e sample from g being a standard normal
f <- dt(x, df = 3)  # density of x under f
g <- dnorm(x)  # density of x under g
w <- f/g  # weights
mean(x * w)  #EX

## [1] 0.004417

mean((x * w)^2)  #VX

## [1] 4.846


par(mfrow = c(1, 2), cex = 10, cex.main = 1.2)
hist(log(w))
hist(log(x * w))

## Warning:  NaNs produced
```



Problem 4

This is a truncated/censored regression problem. In a given i.i.d. sample, stochastically, c of the n original observations will be censored (point y_i will be denoted as NA in the simulated test sample if and only if $y_i > \tau$). Without loss of generalarity, we can separate the indices set into $i = 1, .., c$ with NA and $i = c+1, ..., n$ with y_i. Here, we assume a simple linear regression model, $Y \sim X\beta + \epsilon$ with $y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$, i.e. define $\beta = (\beta_0, \beta_1), \mu_i = (1, x_i)^T\beta$

(a)

EM algorithm is designed to estimate the parameter set, $\theta = (\beta_0, \beta_1, \sigma^2)$. Here, we take the complete data to be the available data plus the actual values of the truncated observations, i.e. $\{Y, Z\} = \{\{y_i\}, \{z_i\}\}$, of which $z_i, i = 1, ..., c$ are values in place of the initially NA censored observations.

and they are functions of $\theta^{(t)}$, constant in terms of the maximization step. So the expected complete log likelihood is

$logN(y_i; \theta) = -log\sqrt{2\pi\sigma^2} - \dfrac{(y_i - \mu_i)^2}{2\sigma^2}; \, \mu_i = \beta_0 + \beta_1 x_i$

$logf(Y, Z; \theta) = \sum_{i=c+1}^{n} logN(y_i; \theta) + \sum_{i=1}^{c} logN(z_i; \theta)$

$E[logN(z_i; \theta)|Y, X, \theta^{(t)}] = E[logN(z_i; \theta)|z_i > \tau, \theta = \theta^{(t)}] = -log\sqrt{2\pi\sigma^2} - \dfrac{1}{2\sigma^2}E[(z_i - \mu_i)^2|z_i > \tau, \theta = \theta^{(t)}]$

Now, let $m_i^{(t)} = E(z_i|z_i > \tau, \theta = \theta^{(t)})$ and $v_i^{(t)} = V(z_i|z_i > \tau, \theta = \theta^{(t)})$,
where $m_i^{(t)} = \mu_i^{(t)} + \sigma^{(t)}\rho(\tau_i^{*(t)})$ and $v_i^{(t)} = (\sigma^2)^{(t)}(1 + \tau_i^{*(t)}\rho(\tau_i^{*(t)}) - \rho^2(\tau_i^{*(t)}))$
of which $\mu_i^{(t)} = \beta_0^{(t)} + \beta_1^{(t)} x_i$, $\tau_i^{*(t)} = (\tau - \mu_i^{(t)})/\sigma^{(t)}$,
$\rho(\tau_i^{*(t)}) = \dfrac{\phi(\tau_i^{*(t)})}{1 - \Phi(\tau_i^{*(t)})}$ and $\phi(\cdot)$ is the standard normal PDF, $\Phi(\cdot)$ is the standard normal CDF.

We have $E[(z_i - \mu_i)^2|z_i > \tau, \theta = \theta^{(t)}] = (m_i^{(t)} - \mu_i)^2 + v_i^{(t)}; So,$

$$Q(\theta; \theta^{(t)}) = E[logf(Y, Z; \theta)|Y, X, \theta^{(t)}] = \sum_{i=c+1}^{n} logN(y_i; \theta) + \sum_{i=1}^{c} logN(m_i^{(t)}; \theta) + \sum_{i=1}^{c} \frac{v_i^{(t)}}{-2\sigma^2}$$

Then, the maximization step would give us updates on θ very similar to standard linear regression;

$$\beta^{(t+1)} = (X^TX)^{-1}X^TW^{(t)}, \text{ where } W^{(t)} = \{Y, M^{(t)}\} = \{\{y_i\}, \{m_i^{(t)}\}\};$$

$$(\sigma^2)^{(t+1)} = \frac{1}{n}[\sum_{i=c+1}^{n} (y_i - \mu_i^{(t)})^2 + \sum_{i=1}^{c}(m_i^{(t)} - \mu_i^{(t)})^2 + \sum_{i=1}^{c} v_i^{(t)}]$$

(b)

The reasonble starting values for the 3 parameters as functions of the observations are:

$\beta_0^{(0)} = \bar{y} - \beta_1^{(0)}\bar{x}$

$\beta_1^{(0)} = [(n - c)\bar{y} + c\tau]/\bar{x}$

$(\sigma^2)^{(0)} = \dfrac{1}{n - c}[\sum_{i=c+1}^{n} (y_i - \beta_0^{(0)} - \beta_1^{(0)} x_i)^2]$

(c)

In order to test the implemeted EM algorithm for parameter estimation, we have simulated data for this purpose. For a sample size of $n = 100$, we make the true parameters such that with complete data, $\hat{\beta}_1/se(\hat{\beta}_1) \approx 3$, i.e. $\sigma^2 = (\beta_1/3)^2 \sum_{i=1}^{n}(x_i - \bar{x})^2$. The R function *EM.censor* takes in x, y, τ and estimate θ

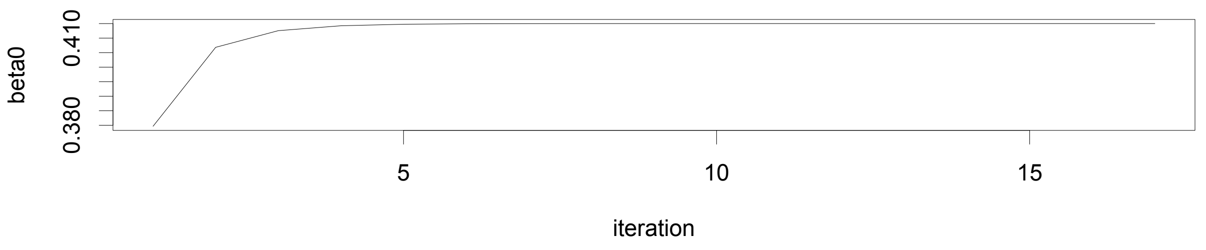
with initializatiion from (b) and $lm()$ for updating β. The criteria for deciding when to stop the optimization is with $diff < tol$, i.e. the difference between the last and current θ is below the fixed tolerance value.

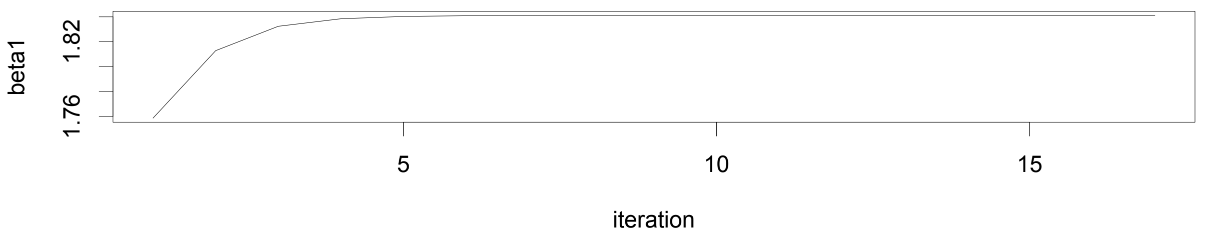
```
#### (c) EM implementation
EM.censor <- function(x, y, tau) {
    obs <- which(!is.na(y))  # observed y indices
    cen <- which(is.na(y))  # censored y indices
    c <- length(cen)
    n <- length(y)  # sample size
    ny <- y  #censored y's to be updated by EM
    # simulation setup
    tol <- 1e-08
    iter <- 0
    lim <- 1000
    beta0.trace <- matrix(NA, lim, 1)
    beta1.trace <- matrix(NA, lim, 1)
    sigma2.trace <- matrix(NA, lim, 1)
    more = TRUE
    # initialization
    beta1 <- ((n - c) * mean(y[obs]) + c * tau)/(n * mean(x))
    beta0 <- mean(y[obs]) - beta1 * mean(x)
    sigma2 <- sum((y[obs] - beta0 - beta1 * x[obs])^2)/(n - c)
    # auxiliary function
    rho <- function(x) {
        return(dnorm(x)/(1 - pnorm(x)))
    }
    # main optimization
    while (more) {
        iter <- iter + 1
        # E-step
        mu <- beta0 + beta1 * x
        tauS <- (tau - mu)/sqrt(sigma2)
        ny[cen] <- mu[cen] + sqrt(sigma2) * rho(tauS[cen])  # E(Z)
        vy <- sigma2 * (1 + tauS[cen] * rho(tauS[cen]) - rho(tauS[cen])^2)  # V(Z)
        # M-step
        fit <- lm(ny ~ x)  # LS fit for beta MLEs
        b0 <- fit$coefficients[1]
        b1 <- fit$coefficients[2]
        sig2 <- (sum(fit$residuals^2) + sum(vy))/n  # MLE of sigma2
        # convergence
        diff.th <- abs(beta0 - b0) + abs(beta1 - b1) + abs(log(sigma2) - log(sig2))
        more <- (diff.th > tol)  # check tol
        # update theta
        beta0 <- b0
        beta1 <- b1
        sigma2 <- sig2
        # record trace
        beta0.trace[iter] <- beta0
        beta1.trace[iter] <- beta1
        sigma2.trace[iter] <- sigma2
    }
    # results
    theta.result <- c(beta0, beta1, sigma2)
    names(theta.result) <- c("beta0", "beta1", "sigma2")
    print(theta)  #global true parameter
    print(iter)
    print(theta.result)
    # plots
    par(mfrow = c(3, 1), mar = c(4, 4, 2, 2), cex = 10, cex.main = 1.2)
    plot(1:iter, beta0.trace[1:iter], xlab = "iteration", ylab = "beta0", "l")
    plot(1:iter, beta1.trace[1:iter], xlab = "iteration", ylab = "beta1", "l")
    plot(1:iter, sigma2.trace[1:iter], xlab = "iteration", ylab = "sigma2",
        "l")
    par(mfrow = c(1, 1), cex = 10, cex.main = 1.2)
    plot(x, y, ylim = c(min(y[obs]) - 1, max(y[obs]) + 2))  # plot data
    points(x[cen], ny[cen], type = "p", pch = 20)
    lines(c(sort(x)[1], sort(x)[n]), c(beta0 + beta1 * sort(x)[1], beta0 + beta1 *
        sort(x)[n]))  # LS line
}
```

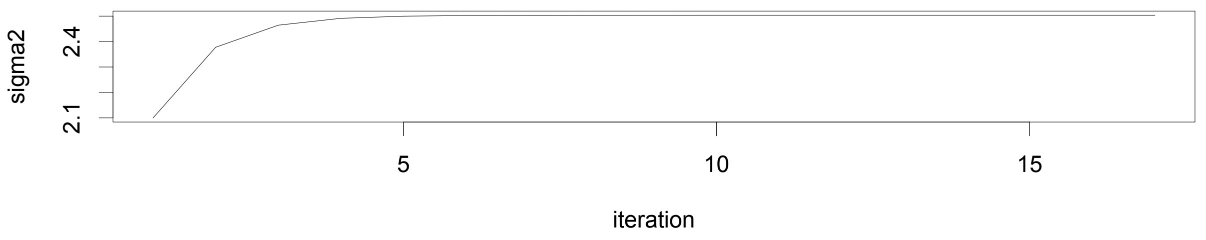
The test case (a) with a modest proportion of exceedances expected is shown below.

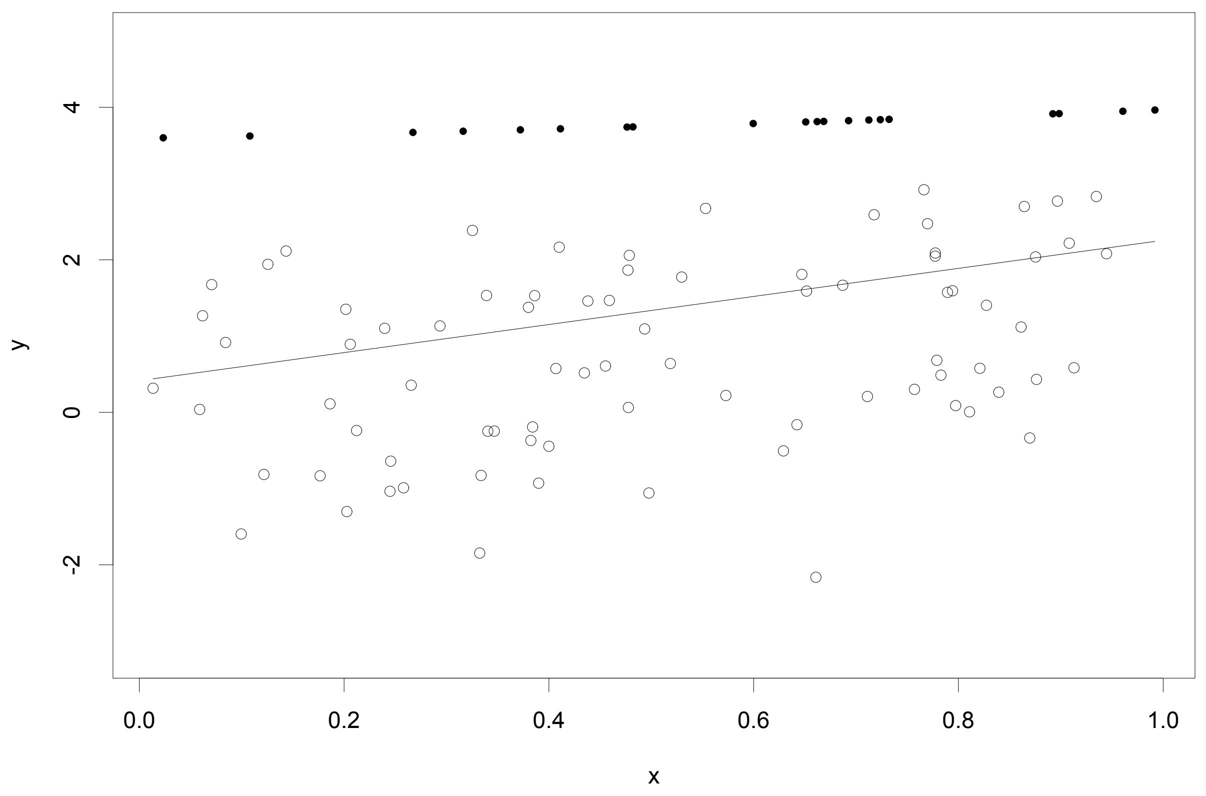
```
#### test data generation
set.seed(0)
n <- 100  # sample size
c <- n * 0.2  # test (a)/(b) exceedance=20%/80%
# true parameter values
theta <- rep(NA, 3)
names(theta) <- c("beta0", "beta1", "sigma2")
theta["beta0"] <- 0.5
theta["beta1"] <- 2
x <- runif(n)
theta["sigma2"] <- (theta["beta1"]/3)^2 * sum((x - mean(x))^2)
# simulated data
e <- rnorm(n, mean = 0, sd = sqrt(theta["sigma2"]))
y <- theta["beta0"] + theta["beta1"] * x + e
sort.y <- sort(y, decreasing = TRUE)
tau <- sort.y[c + 1]  # threshold
y[which(y > tau)] <- as.numeric("NA")
```

```
#### EM result
EM.censor(x, y, tau)

##  beta0  beta1 sigma2
##  0.500  2.000  3.211
## [1] 17
##  beta0  beta1 sigma2
##  0.415  1.841  2.504
```


beta0
0.380
0.410
5
10
15
iteration


beta1
1.76
1.82
5
10
15
iteration

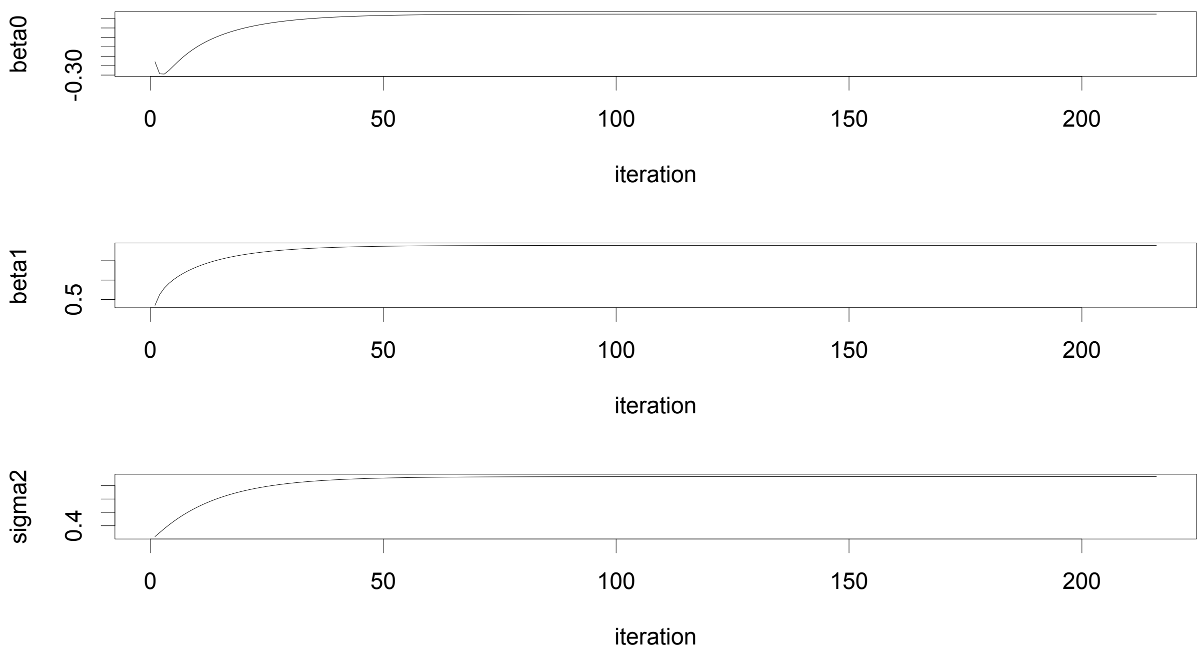

sigma2
2.1
2.4
5
10
15
iteration

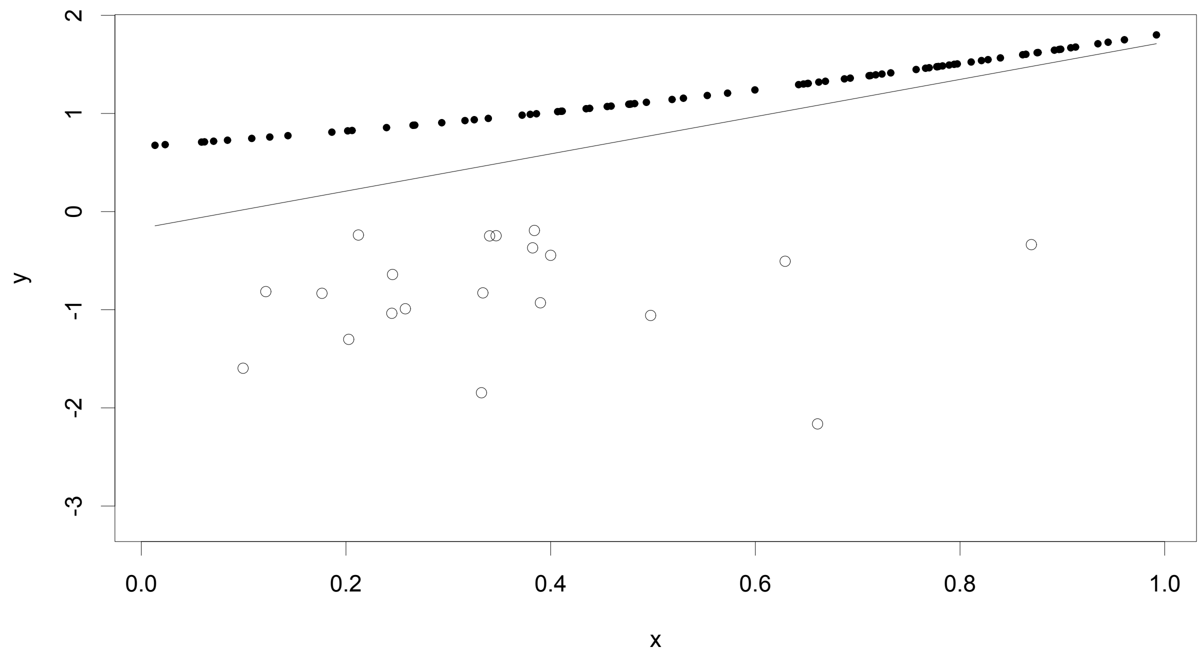

y
-2
0
2
4
0.0
0.2
0.4
0.6
0.8
1.0
x

The test case (b) with a high proportion of exceedances expected is shown below.

```
#### EM result
EM.censor(x, y, tau)
```

```
##  beta0  beta1 sigma2
##  0.500  2.000  3.211
## [1] 216
##   beta0   beta1  sigma2
## -0.1705  1.8967  1.1354
```





(d)

A different approach to this problem will be just directly minimize the negative log likelihood of the data. For the censored observations, we just involve the likelihood terms, $P(Y_i > \tau), Y_i \sim N(\mu_i, \sigma^2)$. We use *optim*() with BFGS option in R to estimate the parameters and their standard errors.

```
#### (d) BFGS implementation
BFGS.censor <- function(x, y, tau) {
    obs <- which(!is.na(y))  # observed y indices
    cen <- which(is.na(y))  # censored y indices
    c <- length(cen)
    n <- length(y)  # sample size
    # initialization
    beta1 <- ((n - c) * mean(y[obs]) + c * tau)/(n * mean(x))
    beta0 <- mean(y[obs]) - beta1 * mean(x)
    sigma2 <- sum((y[obs] - beta0 - beta1 * x[obs])^2)/(n - c)
    init0 <- c(beta0, beta1, sigma2)
    # log liklihood
    pp.lik <- function(par, x, y, tau, obs, cen) {
        b0 <- par[1]
        b1 <- par[2]
        sig2 <- par[3]
        mu <- b0 + b1 * x
        if (sig2 <= 0) {
            sig <- 1e-06
        } else {
            sig <- sqrt(sig2)
        }
        obs.lik <- sum(log(dnorm(y[obs], mean = mu[obs], sd = sig)))
        cen.lik <- sum(log(pnorm(tau, mean = mu[cen], sd = sig, lower.tail = F)))
        return(-(obs.lik + cen.lik))
    }
    # optimize
    opt <- optim(init0, pp.lik, x = x, y = y, tau = tau, obs = obs, cen = cen,
        method = "BFGS", control = list(trace = TRUE, parscale = c(1, 1, 10)),
        hessian = TRUE)
    # results
    print(theta)
    cat(opt$par, "\t theta \n")
    cat(sqrt(diag(solve(opt$hessian))), "\t se(theta) \n")
}
```

The test case (a) with a modest proportion of exceedances expected is shown below. It requires $iter = 10$ for BFGS, while $iter = 17$ for EM.

```
#### BFGS result
BFGS.censor(x, y, tau)
```

```
## initial  value 191.328206
## iter  10 value 171.107012
## final  value 171.049515
## converged
##  beta0  beta1 sigma2
##  0.500  2.000  3.211
## 0.415 1.841 2.504   theta
## 0.3506 0.603 0.4165   se(theta)
```

The test case (b) with a high proportion of exceedances expected is shown below. It requires $iter = 42$ for BFGS, while $iter = 216$ for EM.

```
#### BFGS result
BFGS.censor(x, y, tau)
```

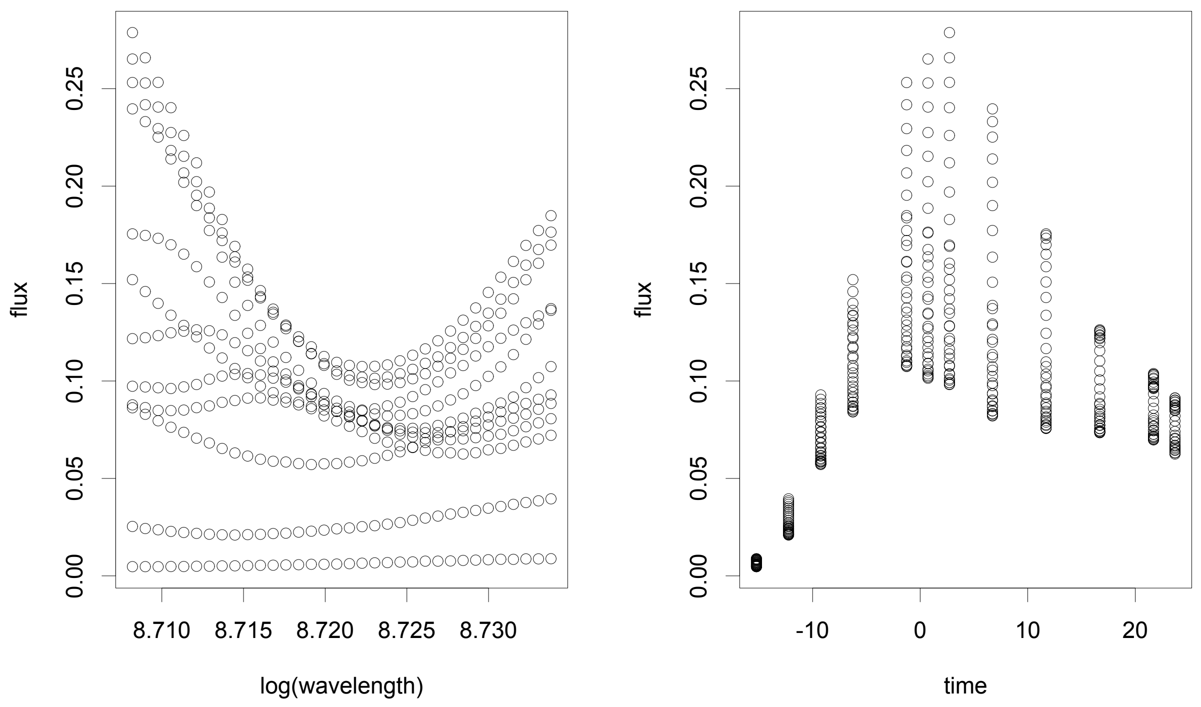
```
## initial  value 187.280367
## iter  10 value 108.318570
## iter  20 value 98.301409
## iter  30 value 80.195175
## iter  40 value 58.781916
## final  value 55.725382
## converged
##  beta0  beta1 sigma2
##  0.500  2.000  3.211
## -0.1705 1.897 1.136   theta
## 0.3105 0.6849 0.4201   se(theta)
```

Problem 5

(a)

The data are measurements of flux from the supernova as a function of (log) wavelength and time. Our goal is to estimate flux (Y_{wt}) as a smooth 2-dimensional function of log wavelength (w) and time t pairs. We assume a normal likelihood as a function of parameters, $\theta = \{\kappa, \lambda, \sigma^2, \rho_w, \rho_t, \tau^2, \alpha\}$, $Y \sim N(\mu_\theta, C_\theta)$.
Here, μ_θ is a vector of value for observation
$E(Y_{wt}) = \mu(t;\theta) = \kappa f(\frac{t}{\lambda})$; and C_θ is a matrix with entries of pairwise covariances of the observations
$Cov(Y_{wt}, Y_{w't'}) = \sigma^2 exp(\frac{-|w-w'|}{\rho_w}) exp(\frac{-|t-t'|}{\rho_t}) + \tau^2 I(t=t') + \alpha v_{wt}^2 I(w=w', t=t')$.
So the plots of data with respect to log wavelength and time are shown below.

```
#### 5. optim
rm(list = ls(all = TRUE))  # remove all objects
source("ps5prob5.R")
data$logw <- log(data$wavelength)  # log wavelength
data$verr <- (data$fluxerror)^2  # square fluxerror
## (a) plot flux data
attach(data)
par(mfrow = c(1, 2), cex = 10, cex.main = 1)
plot(logw, flux, xlab = "log(wavelength)", ylab = "flux")
plot(time, flux, xlab = "time", ylab = "flux")
```



```
detach(data)
```

(b)

A reasonable set of initial values based on the scale of the flux values and the log wavelength and time values are given below inline with the code.

```
## (b) initialize
theta <- rep(NA, 7)
names(theta) <- c("kappa", "lambda", "sigma2", "rho_w", "rho_t", "tau2", "alpha")
theta["lambda"] <- 0.8
theta["kappa"] <- mean(data$flux)/mean(meanpts$value)
Ef <- meanfunc(theta, data$time)
resf <- data$flux - Ef
Covf <- outer(resf, resf)
theta["sigma2"] <- mean(diag(Covf)) * 0.3
theta["rho_w"] <- diff(range(data$logw))
theta["rho_t"] <- diff(range(data$time))
theta["tau2"] <- mean(diag(Covf)) * 0.3
theta["alpha"] <- mean(diag(Covf))/mean(sqrt(data$verr))
print(theta)

##     kappa    lambda    sigma2     rho_w     rho_t      tau2     alpha
##  0.335992  0.800000  0.003243  0.025623 38.916000  0.003243 42.993455
```

(c)

We use *optim*() to optimize the log likelihood to find estimates and standard error for θ. The results below showed that we have tried 2 different methods for finding the global optimum, the default Nelder-Mead and the BFGS. Also, *parscale* is used in the optimization to make the optimization reliable. However, due to the complexity in the likelihood function, we relaxed the convergence a little to reduce the optimization time.

```
## (c) optimize
covfunc <- with(list(verrs = data$verr), function(theta, wavel, times) {
    n <- length(times)
    CV <- matrix(0, n, n)
    scale_w <- wavel/theta["rho_w"]
    scale_t <- times/theta["rho_t"]
    alpv2 <- theta["alpha"] * verrs
    for (i in 1:n) {
        for (j in 1:n) {
            del_w <- abs(scale_w[i] - scale_w[j])
            del_t <- abs(scale_t[i] - scale_t[j])
            cv <- theta["sigma2"] * exp(-del_w) * exp(-del_t)
            if (!del_t) {
                cv <- cv + theta["tau2"]
                if (!del_w) {
                  cv <- cv + alpv2[i]
                }
            }
            CV[i, j] <- cv
        }
    }
    CV
})
```

```
# negative log likelihood
require(mvtnorm)
nn.lik <- with(list(f = data$flux, w = data$logw, t = data$time), function(par) {
    MU <- meanfunc(par, t)
    CV <- covfunc(par, w, t)
    ll <- sum(sum(dmvnorm(f, MU, CV, log = T)))
    -ll
})
```

```
init0 <- theta
opt1 <- optim(par = init0, fn = nn.lik, control = list(trace = TRUE, parscale = init0,
    reltol = 1e-04, maxit = 80), hessian = TRUE)

##   Nelder-Mead direct search function minimizer
## function value for initial parameters = -1610.819434
##   Scaled convergence tolerance is 0.161082
## Stepsize computed as 0.100000
## BUILD              8 -1597.511663 -1623.889530
## EXTENSION         10 -1607.195503 -1647.098990
## LO-REDUCTION      12 -1608.778113 -1647.098990
## LO-REDUCTION      14 -1610.596435 -1647.098990
## LO-REDUCTION      16 -1610.819434 -1647.098990
## EXTENSION         18 -1611.126717 -1666.114092
## LO-REDUCTION      20 -1621.006730 -1666.114092
## EXTENSION         22 -1623.889530 -1690.194185
## LO-REDUCTION      24 -1631.267141 -1690.194185
## LO-REDUCTION      26 -1635.133629 -1690.194185
## EXTENSION         28 -1640.554216 -1729.373748
## LO-REDUCTION      30 -1647.098990 -1729.373748
## EXTENSION         32 -1663.290820 -1761.646704
## LO-REDUCTION      34 -1666.114092 -1761.646704
## LO-REDUCTION      36 -1684.387736 -1761.646704
## LO-REDUCTION      38 -1689.328919 -1761.646704
## LO-REDUCTION      40 -1690.194185 -1761.646704
## HI-REDUCTION      42 -1722.380543 -1761.646704
## LO-REDUCTION      44 -1723.828526 -1761.646704
## HI-REDUCTION      46 -1729.373748 -1761.646704
## LO-REDUCTION      48 -1741.130855 -1761.646704
## LO-REDUCTION      50 -1748.173626 -1761.646704
## LO-REDUCTION      52 -1749.897789 -1761.646704
## LO-REDUCTION      54 -1752.170165 -1761.646704
## EXTENSION         56 -1756.418165 -1769.176145
## LO-REDUCTION      58 -1756.762710 -1769.176145
## LO-REDUCTION      60 -1757.052092 -1769.176145
## EXTENSION         62 -1759.142872 -1778.579957
## LO-REDUCTION      64 -1759.596655 -1778.579957
## LO-REDUCTION      66 -1760.845131 -1778.579957
## LO-REDUCTION      68 -1761.646704 -1778.579957
## EXTENSION         70 -1763.350160 -1792.370206
## LO-REDUCTION      72 -1768.382542 -1792.370206
## LO-REDUCTION      74 -1769.176145 -1792.370206
## LO-REDUCTION      76 -1769.458449 -1792.370206
## EXTENSION         78 -1771.363143 -1814.308692
## LO-REDUCTION      80 -1775.572363 -1814.308692
## Exiting from Nelder Mead minimizer
##     82 function evaluations used

cat("theta \n", opt1$par, "\n")

## theta
##  0.261 1.21 0.001584 0.04837 60.05 0.003646 16.87

cat("se(theta) \n", sqrt(diag(solve(opt1$hessian))), "\n")

## se(theta)
##  0.04703 0.3371 0.0001027 0.007372 11 NaN NaN
```

```
init0 <- theta
opt2 <- optim(par = init0, fn = nn.lik, method = "BFGS", control = list(trace = TRUE,
    parscale = init0, reltol = 1e-04, maxit = 20))

## initial  value -1610.819434
## iter  10 value -1929.347625
## final  value -1941.131307
## converged

cat("theta \n", opt2$par, "\n")

## theta
##  0.1759 0.8808 0.0005201 0.04311 30.64 0.00256 -1.27
```

Problem 6

In the work that won the Netflix Prize done by C. Volinsky etc. at AT&T labs, a variation on the singular value decomposition (SVD) of the matrix factorization methodology is employed.

Given a matrix R that represents m users (individual Netflix members) and n items (movies) with entries of movie ratings r_{ui}, $R = \{r_{ui}\}_{1 \le u \le m, 1 \le i \le n}$, the SVD factorization-based methodology computes the besk rank-f approximation $R^f = P_{m\times f} Q^T_{n\times f}$, where $f \le m, n$.

Due to the fact that most entries of R in this movie recommendation problem are unknown, the SVD here is used to extend the given data by filling in the unknow ratings by estimation $r_{ui} \sim R^f_{ui} = p_u^T \cdot q_i$, where p_u is the u-th row of P corresponding to user u and q_i is the i-th row of Q corresponding to item i.

An EM-based algorithm is employed to iteratively compute the R matrix SVD by doing least square minimization of $\|R - PQ^T\|_F$ when alternating the fixed point between matrix P and Q, i.e.
$Q^T \leftarrow (P^T P)^{-1} P^T R$
$P \leftarrow RQ(Q^T Q)^{-1}$
Shrinkage is integrated to alleviate the overfitting problem and further reduce the estimation error,
$Err(P, Q) \equiv \sum_{(u,i)} (r_{ui} - p_u^T q_i)^2$.

To be more specific, assuming that we have already computed the first $f - 1$ columns of matrices P, Q. The pseudo code for computing the f-th column of matrices P, Q is given below [1]:

```r
# Problem 5.6: EM based SVD variant
f <- f + 1  # iteration
computeNext <- function(r, P, Q) {
    #known ratings, user factors, item factors
    # compute the f-th column of P,Q to fit given ratings columns 1,...,f-1
    # were already computed
    a <- alpha
    e <- eps
    # compute residuals portion not explained by previous factors
    for (u in 1:m) {
        for (i in 1:n) {
            r[u, i] <- r[u, i] - crossprod(P[u, 1:(f - 1)], Q[i, 1:(f - 1)])
            n <- s[u, i]  # support behind r
            r[u, i] <- n * r[u, i]/(n + a * f)  #shrinkage
        }
    }
    # compute the f-th factor for each user and item by solving many least
    # square problems, each with a single unknown
    while (err(P, Q) < 1 - e) {
        for (u in 1:m) {
            P[u, f] <- crossprod(r[u, ], Q[, f])/crossprod(Q[, f])
        }
        for (i in 1:n) {
            Q[i, f] <- crossprod(r[, i], P[, f])/crossprod(P[, f])
        }
    }
    return(list(P = P, Q = Q))
}
```

At the end of the process, we obtain an approximation of all ratings in the form of a matrix product PQ^T, with predictions for unrated movies. The eigenvectors from SVD are user and item factors, while the eigenvalues are confidence indicators for the predictions.

References

[1] R. Bell, Y. Koren, C. Volinsky, "Modeling relationships at multiple scales to improve accuracy of large recommender systems", *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pp. 95-104, Aug. 2007, San Jose, CA.