Homework 2

Santiago Ruiz, Nikita Karetnikov, Felix Ubl

2025-04-22

Exercise 1

In this exercise we need to write a function that implements Newton-Raphson method. Let us first copy the material from the slides about this method:

Let $f : \mathcal{I} \to \mathbb{R}$ be a convex, once continuously differentiable function with at least one root in $\mathcal{I}$. Then the sequence converges to a root.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Good! As we understand, to find the root we would need to implement the cycle in R, where we will use the function and its derivative.

The function is given in the task:

$$f(x) = x^3 - 4x^2 + 18x - 115.$$

Let us implement it:

```r
target_function <- function(x){
  y = x^3 - 4*x^2 + 18*x - 115

  return(y)
}
```

The derivative of the function is:

$$f'(x) = 3x^2 - 8x + 18.$$

Let us implement it:

```r
derivative <- function(x){
  y = 3*x^2 - 8*x + 18

  return(y)
}
```

Now let us implement the cycle.

```r
x_0 = 10 # initial guess

x_n = x_0
x_n_prev = 0
```

```r
epsilon <- .Machine$double.eps
while ( abs(x_n_prev-x_n)>epsilon){
  x_n_prev = x_n
  x_n = x_n_prev -  target_function(x_n_prev)/derivative(x_n_prev)
}

root = x_n
print(root)
```

```
## [1] 5
```

Let us verify that the computed value is indeed a root of the function by evaluating $f(x_n)$ and checking whether it is equal to zero (within numerical precision):

```r
target_function(root) == 0
```

```
## [1] TRUE
```

Looks very good!

Exercise 2

In this exercise, we assume lifetimes of light bulbs follow an exponential distribution with parameter λ. Data arise from two experiments:

1. **Complete data**: failure times $u_1, \dots, u_n$ recorded exactly.
2. **Censored data**: out of m bulbs observed until time t, only the total count r failures by t are recorded; individual failure times v_j are missing.

(i) Log-likelihood

The combined log-likelihood (up to additive constants) is:

$$\ell(\lambda; u, r, t, m) = n \log \lambda - \lambda \sum_{i=1}^{n} u_i + r \log(1 - e^{-\lambda t}) + (m - r)(-\lambda t).$$

(ii) R function for the log-likelihood

```r
loglik <- function(lambda, u, r, t, m) {
  if (lambda <= 0) return(-Inf)
  term1 <- length(u)*log(lambda) - lambda * sum(u)
  term2 <- r * log(1 - exp(-lambda * t)) + (m - r) * (-lambda * t)
  term1 + term2
}
```

(iii) Maximum-likelihood estimation via `optim()`

We generate artificial data and maximize $\ell(\lambda)$:

```r
set.seed(1234)
n <- 100; m <- 20; t <- 3
u <- rexp(n, 2)
v <- rexp(m, 2)
r <- sum(v <= t)
```

```r
# Negative log-likelihood for minimization:
nll <- function(lam) -loglik(lam, u = u, r = r, t = t, m = m)
opt1 <- optim(par = 1, fn = nll,
              method = "Brent", lower = 0.0001, upper = 10)
mle_lambda <- opt1$par
cat("MLE via optim(): lambda =", mle_lambda, "\n")
```

```
## MLE via optim(): lambda = 2.053554
```

Exercise 3

We implement the golden-section search to maximize a univariate function.

(i) Golden-section search function

```r
GoldenSection <- function(f, interval, ..., tol = 1e-6, iter.max = 100) {
  a <- interval[1]; b <- interval[2]
  gr <- (sqrt(5) + 1) / 2
  c <- b - (b - a) / gr
  d <- a + (b - a) / gr
  fc <- f(c, ...); fd <- f(d, ...)
  for (i in seq_len(iter.max)) {
    if (abs(b - a) < tol) break
    if (fc > fd) {
      b <- d; d <- c; fd <- fc
      c <- b - (b - a) / gr; fc <- f(c, ...)
    } else {
      a <- c; c <- d; fc <- fd
      d <- a + (b - a) / gr; fd <- f(d, ...)
    }
  }
  x_opt <- (a + b) / 2
  list(estimate = x_opt, value = f(x_opt, ...))
}
```

(ii) Application to the light-bulb log-likelihood

```r
# Maximize the log-likelihood from Exercise 2:
gs_res <- GoldenSection(
  function(lam) loglik(lam, u = u, r = r, t = t, m = m),
  interval = c(0.001, 10), tol = 1e-8
)
cat("GoldenSection: lambda =", gs_res$estimate,
    " value =", gs_res$value, "\n")
```

```
## GoldenSection: lambda = 2.053554  value = -28.34572
```

```r
# Compare to built-in optimize():
opt_res <- optimize(
  function(lam) loglik(lam, u = u, r = r, t = t, m = m),
  interval = c(0.001, 10), maximum = TRUE
)
```

```r
cat("optimize():    lambda =", opt_res$maximum,
    " value =", opt_res$objective, "\n")
```

```
## optimize():    lambda = 2.053573  value = -28.34572
```

Both methods yield essentially the same MLE for λ. The numerical agreement confirms our golden-section implementation.

Exercise 4

It is assumed that the lifetime of light bulbs follows an exponential distribution with parameter λ. To estimate λ, n light bulbs were tested until they all failed. Their failure times were recorded as $u_1, ..., u_n$. In a separate experiment, m bulbs were tested, but the individual failure times were not recorded. Only the number of bulbs, r, that had failed at time t was recorded. The missing data are the failure times of the bulbs in the second experiment, $v_1, ..., v_m$.

i) Determine the complete-data log-likelihood.

The complete data log-likelihood is defined as follows:

$$x \sim \text{Exp}(\lambda), \quad f(x) = \lambda e^{-\lambda x}$$

$$\text{Likelihood}(x) = \prod_{i=1}^{n+m} \lambda e^{-\lambda x_i}$$

$$\text{Log-Likelihood}(x) = \sum_{i=1}^{n+m} \log(\lambda e^{-\lambda x_i}) = \sum_{i=1}^{n+m} \log(\lambda) - \lambda x_i$$

ii) Determine the conditional means

By Bayes' rule, we have:

$$P(x \mid y) = \frac{P(y, x)}{P(y)}$$

Let f denote the density function of the exponential distribution. Then we have:

$$P(X \leq t) = \frac{f(s)}{P(X \leq t)}$$

The expression in the denominator is the cumulative distribution function (CDF) of the exponential distribution, which is given by:

$$P(X \leq t) = 1 - e^{-\lambda t}$$

The support of this new density is given by $[0, t]$, and the density function is:

$$g(s) = \frac{\lambda e^{-\lambda s}}{1 - e^{-\lambda t}}$$

Now, define the random variable $Y = X + Nt$, where $N = \max\{n \in \mathbb{N} : nt < X\}$. Then we have:

$$P(nt < X) = 1 - P(X \leq nt) = (e^{-\lambda t})^n = P(X \leq t)^n$$

In other words, N is a random variable that counts the number of failures before time t. We can define such a probability in terms of success, meaning in terms of the probability of not having a failure until time t, $P(X > t) = 1 - e^{-\lambda t}$. Thus, N follows a geometric distribution with parameter $p = P(X > t)$.

Since X and Y are independent, we can write the joint distribution as:

$$P(X, Y) = P(X)P(Y)$$

$$P(N = n, Y \leq s) = P(nt < X \leq nt + s) = F(nt + s) - F(nt) = e^{-\lambda nt}(1 - e^{-\lambda s})$$

If we multiply and divide by $1 - e^{-\lambda t}$, we get the PDF of the random variable N times the cumulative distribution function of the random variable Y:

$$P(N = n, Y \leq s) = (e^{-\lambda t})^n (1 - e^{-\lambda t}) \frac{1 - e^{-\lambda s}}{1 - e^{-\lambda t}}$$

Therefore, the CDF of Y is given by:

$$G(s) = \frac{1 - e^{-\lambda s}}{1 - e^{-\lambda t}}$$

Differentiating with respect to s, we get the PDF of Y:

$$g(s) = \frac{\lambda e^{-\lambda s}}{1 - e^{-\lambda t}}$$

Thus, Y is the random variable that counts the average value of the light bulbs that failed before time t.

$$E[Y] = E[X] + tE[N] = \frac{1}{\lambda} + \frac{e^{-\lambda t}}{1 - e^{-\lambda t}}$$

Now, as for $E[X|X > t] = t + \frac{1}{\lambda}$ because the exponential distribution is memoryless.

iii) Determine the E- and M-step of the EM algorithm.

Let us define a variable Z that takes the value of $Z = I(x \leq 1)$ and 0 otherwise. We can compute the expected value of Z as follows:

$$E[X|Z = z] = z\left(\frac{1}{\lambda} + \frac{e^{-\lambda t}}{1 - e^{-\lambda t}}\right) + (1 - z)(t + \frac{1}{\lambda})$$

The EM algorithm consists of two steps: the E-step and the M-step.

The E-step computes the expected value of the complete-data log-likelihood given the observed data and the current estimate of the parameters. The M-step maximizes this expected value with respect to the parameters.

In the E-Step we compute the value of

$$\lambda$$

based on its previous value then we update in the M-step. Namely, in the E-step we compute the expectation below given the current value of

$$\lambda$$

:

$$E[\text{Log-Likelihood}(x)|Z = z, \lambda_k, x_i] = N\log(\lambda_k) - \lambda_k \sum_{i=1}^{n+m} E[x_i|Y = y, \lambda_k]$$

In the M step, we use the values provided by remplacing v_i with its expectation, and maximize to finf the new value of

$$\lambda_{k+1}$$

:

The value λ_{k+1} that maximizes the expected log-likelihood is given by:

$$\lambda_{k+1} = \frac{N}{\sum_{i=1}^{n} u_i + \sum_{j=1}^{m} E[x_i|Z = z, \lambda_k]}$$

iv) Implement the EM algorithm and apply it to artificial data generated with:

```
n <- 100; m <- 20; t <- 3
set.seed(1234)

# True lambda
lambda_true <- 2

# Generate observed and censored data
u <- rexp(n, lambda_true)          # fully observed failure times
v <- rexp(m, lambda_true)          # partially observed (we only know how many fail before t)
r <- sum(v <= t)                   # number of censored bulbs that failed before time t

# Initial estimate using only observed data
# lambda_init <- n / sum(u)
lambda_init <- 35

# Initialize vector for expected values
v_em <- numeric(m)

# Expected value of truncated exponential (0   X   t)
expected_truncated <- function(lambda, t) {
  return((1 / lambda) - (t * exp(-lambda * t)) / (1 - exp(-lambda * t)))
}

# Expected value of exponential given X > t (memoryless property)
expected_censored <- function(lambda, t) {
  return(t + (1 / lambda))
}

# EM Algorithm
lambda <- lambda_init

for (i in 1:10) {
```

```
  # E-step: compute expected values for the m partially observed failure times
  for (j in 1:m) {
    if (j <= r) {
      v_em[j] <- expected_truncated(lambda, t)  # Expected failure time given v_j   t
    } else {
      v_em[j] <- expected_censored(lambda, t)   # Expected failure time given v_j > t
    }
  }

  # M-step: update lambda using complete expected data
  lambda <- (n + m) / (sum(u) + sum(v_em))

  print(lambda)
}
```

```
## [1] 2.429424
## [1] 2.104733
## [1] 2.061269
## [1] 2.054733
## [1] 2.053734
## [1] 2.053581
## [1] 2.053558
## [1] 2.053554
## [1] 2.053554
## [1] 2.053554
```

Exercise 5

i) Specify the log-likelihood for a single observation y_i. Assume for y_i that the first d_1 variables are observed and that the next d_2 variables are missing, i.e.,

$$\ell(\mathbf{y}_i; \mu, \Sigma) = -\frac{1}{2}\left[d \log(2\pi) + \log|\Sigma| + (\mathbf{y}_i - \mu)^\top \Sigma^{-1} (\mathbf{y}_i - \mu)\right]$$

ii) Implement a function loglik which given the data and the parameter values and Σ returns the log-likelihood values.

iii) The parameters need to be vectorized for the general purpose optimizer. Suggest a suitable vectorizing scheme.

```
library(mvtnorm)
```

```
## Warning: package 'mvtnorm' was built under R version 4.2.3
```

```
loglik <- function(y, mu, sigma) {
  d <- ncol(y)
  n <- nrow(y)
  log_likelihood <- numeric(n)

  # Check for valid sigma
  if (any(is.na(sigma)) || det(sigma) <= 0 || any(!is.finite(sigma))) {
    return(NA)
  }
```

```r
  for (i in 1:n) {
    y_i <- y[i, ]
    diff <- y_i - mu
    ll <- tryCatch({
      -0.5 * (d * log(2 * pi) + log(det(sigma)) +
              t(diff) %*% solve(sigma) %*% diff)
    }, error = function(e) NA)
    log_likelihood[i] <- ll
  }
  return(sum(log_likelihood))
}

# Create a wrapper to unpack parameters
neg_loglik_wrapper <- function(params, y, d) {
  mu <- params[1:d]
  L_vals <- params[(d + 1):length(params)]

  # Reconstruct lower triangle matrix
  L <- matrix(0, d, d)
  L[lower.tri(L, diag = TRUE)] <- L_vals

  # sigma = L %*% t(L), to ensure positive-definite
  sigma <- L %*% t(L) + diag(1e-6, d)

  # Return negative log-likelihood (since optim minimizes)
  return(-loglik(y, mu, sigma))
}

# Generate sample data
set.seed(1234)
n <- 100
d <- 3
true_mu <- c(0, 0, 0)
true_sigma <- matrix(c(1, 0.5, 0.3,
                       0.5, 1, 0.2,
                       0.3, 0.2, 1), nrow = d)
y <- rmvnorm(n, mean = true_mu, sigma = true_sigma)

# Initial guesses
init_mu <- rep(3, d)
init_L <- diag(1, d)
init_L_vals <- init_L[lower.tri(init_L, diag = TRUE)]
init_params <- c(init_mu, init_L_vals)

# Optimize
result <- optim(init_params, neg_loglik_wrapper, y = y, d = d, method = "BFGS",
                control = list(maxit = 1000, reltol = 1e-8))

# Extract optimized mu and sigma
opt_mu <- result$par[1:d]
opt_l_vals <- result$par[(d + 1):length(result$par)]
lower_triangle_matrix <- matrix(0, d, d)
lower_triangle_matrix[lower.tri(lower_triangle_matrix, diag = TRUE)] <- opt_l_vals
```

```
opt_sigma <- lower_triangle_matrix %*% t(lower_triangle_matrix)

# Show result
cat("Optimized mu:\n")
```

```
## Optimized mu:
```

```
print(opt_mu)
```

```
## [1] -0.02946748  0.09323005 -0.01456996
```

```
cat("Optimized sigma:\n")
```

```
## Optimized sigma:
```

```
print(opt_sigma)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.8994061 0.6005967 0.2878640
## [2,] 0.6005967 1.0564672 0.4036455
## [3,] 0.2878640 0.4036455 1.1987436
```

iv) Use optim to maximize the log-likelihood on an artificial data set of size 200 drawn from a 3-dimensional multivariate normal distribution with mean zero and variance-covariance matrix

```
library(mvtnorm)

# Generate sample data with missing values
set.seed(1234)
n <- 200
d <- 3
true_mu <- c(0, 0, 0)
true_sigma <- matrix(c(1, 0.5, 0.5,
                       0.5, 1, 0.5,
                       0.5, 0.5, 1), nrow = d)
y <- rmvnorm(n, mean = true_mu, sigma = true_sigma)

# Assign 30% missing values randomly
missing_indices <- matrix(runif(n * d) < 0.3, nrow = n, ncol = d)
y[missing_indices] <- NA

# Log-likelihood function
loglik <- function(y, mu, sigma) {
  d <- ncol(y)
  n <- nrow(y)
  log_likelihood <- numeric(n)

  if (any(!is.finite(mu)) || any(!is.finite(sigma)) || det(sigma) <= 0) {
    return(NA)
  }

  for (i in 1:n) {
    y_i <- y[i, ]
    if (any(is.na(y_i))) next
```

```r
    diff <- y_i - mu
    ll <- tryCatch({
      -0.5 * (d * log(2 * pi) + log(det(sigma)) +
                t(diff) %*% solve(sigma) %*% diff)
    }, error = function(e) NA)
    log_likelihood[i] <- ll
  }
  return(sum(log_likelihood, na.rm = TRUE))
}

# Imputation function
impute_missing <- function(y, mu, sigma) {
  y_imputed <- y
  for (i in 1:nrow(y)) {
    obs_idx <- !is.na(y[i, ])
    miss_idx <- is.na(y[i, ])
    if (any(miss_idx) && any(obs_idx)) {
      mu_obs <- mu[obs_idx]
      mu_miss <- mu[miss_idx]
      sigma_oo <- sigma[obs_idx, obs_idx]
      sigma_mo <- sigma[miss_idx, obs_idx]
      y_obs <- y[i, obs_idx]

      y_miss_hat <- tryCatch({
        mu_miss + sigma_mo %*% solve(sigma_oo) %*% (y_obs - mu_obs)
      }, error = function(e) rep(NA, sum(miss_idx)))

      y_imputed[i, miss_idx] <- y_miss_hat
    }
  }
  return(y_imputed)
}

# Wrapper for optimization
neg_loglik_wrapper <- function(params, y, d) {
  mu <- params[1:d]
  L_vals <- params[(d + 1):length(params)]
  L <- matrix(0, d, d)
  L[lower.tri(L, diag = TRUE)] <- L_vals
  sigma <- L %*% t(L) + diag(1e-6, d)  # regularization for stability
  return(-loglik(y, mu, sigma))
}

# Initialization
init_mu <- colMeans(y, na.rm = TRUE)
init_L <- diag(1, d)
init_L_vals <- init_L[lower.tri(init_L, diag = TRUE)]
init_params <- c(init_mu, init_L_vals)

# Initial values for opt_mu and opt_sigma
opt_mu <- init_mu
opt_sigma <- init_L %*% t(init_L)
```

```r
# EM Algorithm
for (iter in 1:5) {
  cat("\n--- Iteration", iter, "---\n")

  # E-step
  y_imputed <- impute_missing(y, opt_mu, opt_sigma)

  # M-step
  result <- optim(init_params, neg_loglik_wrapper, y = y_imputed, d = d, method = "BFGS",
                  control = list(maxit = 1000, reltol = 1e-8))

  # Update parameters
  opt_mu <- result$par[1:d]
  opt_L_vals <- result$par[(d + 1):length(result$par)]
  L <- matrix(0, d, d)
  L[lower.tri(L, diag = TRUE)] <- opt_L_vals
  opt_sigma <- L %*% t(L)

  init_params <- result$par

  # Output
  cat("mu:\n"); print(opt_mu)
  cat("sigma:\n"); print(opt_sigma)
  cat("log-likelihood:", loglik(y_imputed, opt_mu, opt_sigma), "\n")
}
```

```
##
## --- Iteration 1 ---
## mu:
## [1] -0.008317493 -0.022773774 -0.093718452
## sigma:
##           [,1]      [,2]      [,3]
## [1,] 0.6500835 0.2446865 0.1792482
## [2,] 0.2446865 0.7745268 0.2953061
## [3,] 0.1792482 0.2953061 0.7509705
## log-likelihood: -708.6999
##
## --- Iteration 2 ---
## mu:
## [1]  0.003065366 -0.006070936 -0.083917269
## sigma:
##           [,1]      [,2]      [,3]
## [1,] 0.6928270 0.4619437 0.3253880
## [2,] 0.4619437 0.8402189 0.5047157
## [3,] 0.3253880 0.5047157 0.7947876
## log-likelihood: -665.9273
##
## --- Iteration 3 ---
## mu:
## [1]  0.007791569  0.006893808 -0.072282276
## sigma:
##           [,1]      [,2]      [,3]
## [1,] 0.7583619 0.5946404 0.3922682
## [2,] 0.5946404 0.9241251 0.6075103
```

```
## [3,] 0.3922682 0.6075103 0.8457565
## log-likelihood: -651.6274
##
## --- Iteration 4 ---
## mu:
## [1]  0.005708927  0.013447549 -0.064367021
## sigma:
##           [,1]      [,2]      [,3]
## [1,] 0.7958350 0.6525081 0.3994355
## [2,] 0.6525081 0.9635697 0.6378150
## [3,] 0.3994355 0.6378150 0.8634479
## log-likelihood: -648.0823
##
## --- Iteration 5 ---
## mu:
## [1]  0.002353799  0.016942668 -0.059376083
## sigma:
##           [,1]      [,2]      [,3]
## [1,] 0.8127925 0.6758842 0.3888605
## [2,] 0.6758842 0.9806473 0.6456530
## [3,] 0.3888605 0.6456530 0.8681330
## log-likelihood: -646.8705
```