Statistical Computing - CWB - 2019

Sharan Maiya (S1608480)

```r
# Sources, libraries, seed
source("CWB2019code.R")
library(tidyverse);library(xtable);library(pander);library(ggplot2)
set.seed(10)
```

Question 1

Task 1

The function `negloglike` shown below takes as input the values N, θ, y_1 and y_2 and outputs $l(N, \theta)$.

```r
negloglike <- function(param, Y) {
  if (param[1] < max(Y)) { # If N >= max(y1, y2) then return +Infinity
    return(+Inf)
  } else { # Otherwise we calculate the negated log-likelihood
    return(sum(
      lgamma(Y + 1),
      lgamma(param[1] - Y + 1),
      -2 * lgamma(param[1] + 1),
      2 * param[1] * log(1 + exp(param[2])),
      -param[2] * sum(Y)
    ))
  }
}
```

Task 2

We seek to use the `optim` function with `negloglike` to find a maximum likelihood estimate of N and θ (and in turn ϕ). Since `optim` is a numerical optimiser it is only guaranteed to find a local minima. It therefore makes sense to try `optim` at different sensible starting values to try and find the best MLE we can in a grid search. The following parameter values were tried as starting points:

- N: We know we must have $N > max(y_1, y_2)$ so we try both $N = max(y_1, y_2)+1$ and $N = 2max(y_1, y_2)$.
- θ: Since this is derived from the actual probability ϕ we choose sensible values of ϕ and convert them to a value of θ using the `logit` function provided. Since ϕ is a probability is makes sense to try starting at the values of 0.01, 0.5 and 0.99.

```r
Y <- c(256, 237) # Given data
bestopt <- list(value = +Inf) # Initialise our optimisation
```

```r
# Perform the grid search
for (N_start in list(max(Y) + 1, 2 * max(Y))) {
  for (theta_start in lapply(list(0.01, 0.5, 0.99), logit)) {
    # Use optim with the current starting values
    opt <- optim(par = c(N_start, theta_start), fn = negloglike, Y = Y)
    if (opt$value < bestopt$value) { # Update if we found a better minima
      bestopt <- opt
    }
  }
}

# Record MLEs of N and theta
N_hat <- bestopt$par[1]
theta_hat <- bestopt$par[2]
# Obtain MLE of phi
phi_hat <- ilogit(theta_hat)
```

Table 1: MLEs for $\hat{N}$, $\hat{\theta}$ and $\hat{\phi}$

$\hat{N}$	$\hat{\theta}$	$\hat{\phi}$
388.131	0.554	0.635

In Table 1 we see the maximum likelihood estimates for N, θ and consequently ϕ. This means that in order to maximise the likelihood $p(y|N, \phi)$ we would require there to be around 388 people buried at the site, with a probability 0.64 of finding a femur.

Task 3

We now want to take our values for $\hat{N}$ and $\hat{\theta}$ from Table 1 and use `optimHess` to determine a Hessian $\mathbf{H}$. The inverse $\mathbf{H}^{-1}$ will be a joint covariance matrix we can use to compute a 95% confidence interval for N. Here we use Normal approximation.

Table 2: 95% confidence interval for N

lower	upper
-50.584	826.847

Table 2 shows a 95% confidence interval for N: we are 95% certain the true value of N lies in this range. It is clear that this interval is not very helpful; the lower bound of -50.28 is well below the bound we had already deduced ($N > max(y_1, y_2)$). In fact, values of N below zero are simply nonsensical as we cannot have a negative number of burials. Our upper bound of 826.85 is also considerably high: consider that even if every bone found belonged to a separate person the excavation would still only have found ~60% of the total number of burials should the true N be near this figure. This seems very unlikely.

Question 2

Task 1

We have that the negated log-likelihood $l(N, \theta)$ is given by:

$l(N, \theta) = \log \Gamma(y_1 + 1) + \log \Gamma(y_2 + 1) + \log \Gamma(N - y_1 + 1) + \log \Gamma(N - y_2 + 1) - 2 \log \Gamma(N + 1) + 2N \log (1 + e^{\theta}) - (y_1 + y_2)\theta.$

We begin with the first partial derivatives:

$$\frac{\partial l(N, \theta)}{\partial N} = \Psi(N - y_1 + 1) + \Psi(N - y_2 + 1) - 2\Psi(N + 1) + 2\log(1 + e^{\theta}),$$

and

$$\frac{\partial l(N, \theta)}{\partial \theta} = \frac{2Ne^{\theta}}{1 + e^{\theta}} - (y_1 + y_2).$$

Now we derive expressions for the second order partial derivatives:

$$\begin{aligned}
\frac{\partial^2 l(N, \theta)}{\partial N^2} &= \Psi'(N - y_1 + 1) + \Psi'(N - y_2 + 1) - 2\Psi'(N + 1), \\
\frac{\partial^2 l(N, \theta)}{\partial \theta^2} &= \frac{2Ne^{\theta}}{(1 + e^{\theta})^2} \text{ and} \\
\frac{\partial^2 l(N, \theta)}{\partial N \partial \theta} &= \frac{2e^{\theta}}{1 + e^{\theta}}.
\end{aligned}$$

Task 2

The function `myhessian` will construct a 2x2 Hessian Matrix for $l(N, \theta)$ using the expressions derived for its second order partial derivatives above. The Hessian matrix will be given by:

$$\begin{bmatrix} \frac{\partial^2 l(N,\theta)}{\partial N^2} & \frac{\partial^2 l(N,\theta)}{\partial N \partial \theta} \\ \frac{\partial^2 l(N,\theta)}{\partial \theta \partial N} & \frac{\partial^2 l(N,\theta)}{\partial \theta^2} \end{bmatrix}$$

Below is the implementation of `myhessian`:

```r
myhessian <- function(param, Y) {
  # Extract parameters
  N <- param[1]
  theta <- param[2]

  # Compute second order partial derivatives
  thetatwo <- 2 * N * exp(theta) / (1 + exp(theta))^2
  theta_n <- 2 * exp(theta) / (1 + exp(theta))
  ntwo <- psigamma(N - Y[1] + 1, 1) + psigamma(N - Y[2] + 1, 1) - 2 * psigamma(N + 1, 1)

  # Return Hessian
  return(matrix(c(ntwo, theta_n, theta_n, thetatwo), nrow = 2, ncol = 2))
}
```

Let us now use our MLEs $\hat{N}$ and $\hat{\theta}$ to compare the output of `myhessian` and `optimHess`.

```r
# Find hessian using myhessian
myhess <- myhessian(bestopt$par, Y=Y)
```

The Hessian matrix $\mathbf{H}$ determined by `optimHess` is:

$$\begin{bmatrix} 0.008988309 & 1.270193 \\ 1.270192513 & 179.898086 \end{bmatrix}$$

The Hessian matrix $\mathbf{H}'$ determied by `myhessian` is:

$$\begin{bmatrix} 0.008988314 & 1.270193 \\ 1.270192559 & 179.898109 \end{bmatrix}$$

The matrix of relative differences between $\mathbf{H}$ and $\mathbf{H}'$ is:

$$\begin{bmatrix} 5.500008 \times 10^{-9} & 4.574974 \times 10^{-8} \\ 4.574974 \times 10^{-8} & 2.351907 \times 10^{-5} \end{bmatrix}$$

We see from the matrix of relative differences that our two computed Hessian matrices are almost identical. Indeed the largest diffrent between two computed values is 2.351907×10^{-5} in the value of $\frac{\partial^2 l(N,\theta)}{\partial \theta^2}$ and even this is extremely close to zero. The reason the two matrices are not exact is likely due to the fact that in `myhessian` we calculated each value directly from its expression whereas in `optimHess` these are estimated numerically.

Task 3

In Lecture 5 we learned that 2nd order differences for $f''(\theta)$ using $f(\theta - h)$, $f(\theta + h)$ and $f(\theta)$ give the bound $\lesssim \frac{\epsilon_0(4L_0+2|\theta|L_1)}{h^2} + \frac{h^2 L_4}{12}$. We want to compare the two Hessian evaluations of $\frac{\partial^2 l(N,\theta)}{\partial N^2}$ with this bound so let us first calculate it at the mode $(\hat{N},\hat{\theta})$.

```
L0 <- negloglike(opt$par, Y) # L0 calculated using the negated log likelihood

L1 <- digamma(N_hat-Y[1]+1)
    + digamma(N_hat-Y[2]+1)
    - 2*digamma(N_hat+1)
    + 2*log(1+exp(theta_hat))

# L4 calculated using code given in question
L4 <- abs(sum(psigamma(N_hat-Y+1,3)) - 2*psigamma(N_hat+1,3))

# Machine epsilon (upper bound on relative error due to rounding in FPA)
e <- .Machine$double.eps
h <- 0.0001 # Step size (given in question)

# Calculating the bound using the formula above this code block
bound <- e*(4*L0+2*abs(theta_hat)*L1) / h^2
         + (L4*h^2) / 12
```

The value of the bound calculated is 7.7696094×10^{-7}. What this number represents is an upper bound on the approximation error due to the numerical method used in `optimHess` when compared to the direct `myhessian`. Recall from above that the relative difference between the two calculated values for $\frac{\partial^2 l(N,\theta)}{\partial N^2}$ was 5.500008×10^{-9}. This is much smaller than our upper bound which is as expected. It indicates that the numerical method used in `optimHess` is quite powerful.

TODO: Consider effect of machine epsilon on `myhessian` calculated value!

Task 4

We now want to compare the computational costs of `optimHess` and `myhessian` using the `microbenchmark` function. This was run an order of magnitude more times than the default in order to provide more stable results.

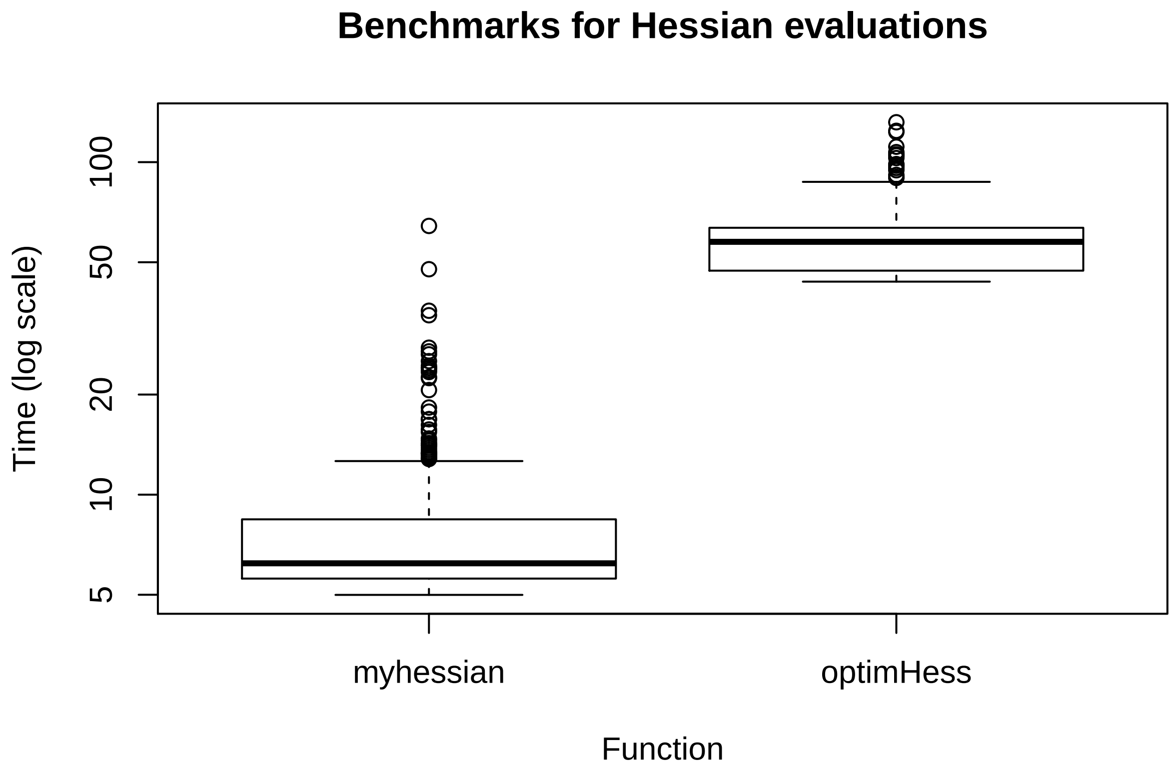
```
bm <- microbenchmark::microbenchmark(myhessian(bestopt$par, Y = Y),
                                     optimHess(bestopt$par, fn = negloglike, Y = Y), times=1000L)
```

Table 3: Summary of benchmarks for both functions (continued below)

expr	min	lq	mean	median	uq	max
myhessian(bestopt$par, Y = Y)	4.996	5.593	7.379	6.217	8.43	64.3
optimHess(bestopt$par, fn = negloglike, Y = Y)	43.72	47.16	57.04	57.59	63.48	131.8

neval
1000
1000

In Table 3 we see a summary of the results of `microbenchmark`. We see that the mean time of running `myhessian` is ~7.3 milliseconds which is significantly lower than the mean time of ~57 milliseconds required by `optimHess`. Indeed, we see that the upper quantile time given for `myhessian` is under 10 milliseconds while the minimum time for `optimHess` is much higher at ~43 milliseconds. `myhessian` did have some outliers to its trend however as we see its maximum time was around 57 milliseconds - this is close to the average time of `optimHess`. A quick look at Table 3 makes it very clear that `myhessian` in this case performs much better than `optimHess` when it comes to the computational cost. It might be easier to interpret the results if we visualise them. This has been done using the boxplot shown over the page.



This boxplot really illustrates that `myhessian` performs a lot better than `optimHess` as we can see the majority of times for `myhessian` are far below the times for `optimHess`. When considering the way the two functions work these results are not surprising. `optimHess` takes as a paramter a function to optimise and carries this out numerically, which for non-trivial functions is sure to take time. `myhessian` on the other hand performs a few simple calculations directly with some input parameters. The algorithm itself is $\mathcal{O}(\Psi') + \mathcal{O}(1)$ which means its runtime is not dependent on how complex the negated log-likelihood function is.

Question 3

Task 1

The function `arch_boot` below takes as input parameters N and θ as well as a positive integer J and produces J parametric bootstrap samples of parameter estimates $\hat{N}$ and $\hat{\theta}$. Recall that in our model the number of left (y_1) and right (y_2) femurs are two independent observations from a $Bin(N, \phi)$ distribution. We will use this to generate our bootstrap samples.

```
arch_boot <- function(param, J) {
  boot_params <- matrix(0, J,2) # Initialise matrix to return

  # Parameters of the binomial distribution
  N <- floor(param[1])
  phi <- ilogit(param[2])

  for (j in 1:J) {
    Y_j <- rbinom(2, N, phi) # Bootstrap sample

    # Estimate parameters
    boot_params[j, ] <- optim(
      par = c(2*max(Y_j), 0),
```

```
      fn = negloglike,
      Y = Y_j)$par
  }
  return(boot_params)
}
```

Task 2

We now use `arch_boot` to estimate the bias ($\mathbb{E}(\hat{\Theta} - \Theta_{true})$)and standard deviations ($\sqrt{Var(\hat{\Theta} - \Theta_{true})}$) of the estimators for N and θ using 10000 parametric bootstrap samples. Here we use the Bootstrap Principle which states that the errors of the bootstrapped estimates have the same distribution as the errors of our parameter estimates ($\hat{N}$,$\hat{\theta}$). In particular, we have that:

$$\mathbb{E}(\hat{N} - N_{true}) = \mathbb{E}(\hat{N}^{(j)} - \hat{N}),$$
$$\mathbb{E}(\hat{\theta} - \theta_{true}) = \mathbb{E}(\hat{\theta}^{(j)} - \hat{\theta})$$

and

$$\sqrt{Var(\hat{N} - N_{true})} = \sqrt{Var(\hat{N}^{(j)} - \hat{N})},$$
$$\sqrt{Var(\hat{\theta} - \theta_{true})} = \sqrt{Var(\hat{\theta}^{(j)} - \hat{\theta})}.$$

This allows us to estimate the bias and standard deviation as follows:

```
# Generate 10000 parametric boostrap estimates
estimates <- arch_boot(bestopt$par, 10000)

errors <- sweep(estimates, 2, bestopt$par)
bias <- colMeans(errors)
std_dev_error <- apply(errors, 2, sd)
bstd_df <- data.frame(bias, std_dev_error)
```

Table 5: Estimated bias and standard deviation of estimators

	Bias	Standard Deviation
$\hat{N}$	93.42	1123
$\hat{\theta}$	2.396	3.831

The bias of $\hat{N}$ is 93.423. This is the expected error of $\hat{N}$ from the true value of N. This is a large bias (recall our estimate $\hat{N}$ was 388.131) and shows that even after our parametric boostrap sampling we are still not very confident of our estimate. This is further evident by the huge standard deviation of this error (1123.017) which is even larger in magnitude to our 95% confidence interval for $\hat{N}$. This shows that while we can estimate N using bootstrap sampling we still have a huge margin of error and are unlikely to be near the true value of N.

TODO: Plot distribution of errors

Task 3

We now construct bootstrap confidence intervals for both N and ϕ.

```r
# Copy of boostrap estimates to change scale
temp <- estimates

# log scale for N
temp[,1] <- log(temp[,1])
log_N_CI <- log(N_hat) - quantile(temp[,1] - log(N_hat),
                                  probs = c(0.975, 0.025))
# theta scale for phi
theta_CI <- theta_hat - quantile(temp[,2] - theta_hat,
                                 probs = c(0.975, 0.025))

# Convert CIs to N and phi
N_CI_2 <- exp(log_N_CI)
phi_CI <- ilogit(theta_CI)
```

Table 6: 95% bootstrap confidence intervals for N and ϕ

	N	ϕ
lower	53.09	4.405e-07
upper	630.3	0.9696

Our 95% bootstrap confidence interval for N is (53.087,630.318). This is smaller on both ends when compared to our confidence interval from before: (-50.584,826.847). This indicates that parametric bootstrap sampling is more reliable than simply estimating our parameters once; this of course makes sense. We also no longer have the lower end of our confidence interval in impossible values below zero. However our lower bound still does not respect the bound we deduced earlier ($N > max(y_1, y_2) = 256$) from the given data we had which indicates that we still have not found a good way of estimating N. It should be noted that our upper bound is now much lower than it was before - a further indication that we are slightly more confident of our results this time. If we impose the lower bound we deduced we can say we are confident the true value of N lies in the interval (256,630.318). In comparison to before this is a much smaller interval (although its range of 374.318 is still not small enough to be really helpful) and helps us get a rough idea of where the true value of N lies.

Question 4

Task 1

Task 2

Task 3

Task 4

Appendix