



# Statistical Computing - CWB - 2019

Sharan Maiya (S1608480)

```
# 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  $N$ ,  $\theta$ ,  $y_1$  and  $y_2$  and outputs  $l(N, \theta)$ .

```
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  $\theta$  (and in turn  $\phi$ ). 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  $N = 2\max(y_1, y_2)$ .
- $\theta$ : Since this is derived from the actual probability  $\phi$  we choose sensible values of  $\phi$  and convert them to a value of  $\theta$  using the `logit` function provided. Since  $\phi$  is a probability it makes sense to try starting at the values of 0.01, 0.5 and 0.99.

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

```

# 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  $\hat{\phi}$

$\hat{N}$	$\hat{\theta}$	$\hat{\phi}$
388.13	0.55	0.64

In Table 1 we see the maximum likelihood estimates for  $N$ ,  $\theta$  and consequently  $\phi$ . 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  $N$

2.5%	97.5%
-50.58	826.85

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`:

```
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`.

```
# 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  $\mathbf{H}'$  determined 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 difference between two computed values is  $2.351907 \times 10^{-5}$  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  $\lesssim \frac{\epsilon_0(4L_0+2|\theta|L_1)}{h^2} + \frac{h^2L_4}{12}$ . 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  $7.7696094 \times 10^{-7}$ . 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  $5.500008 \times 10^{-9}$ . 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)
```

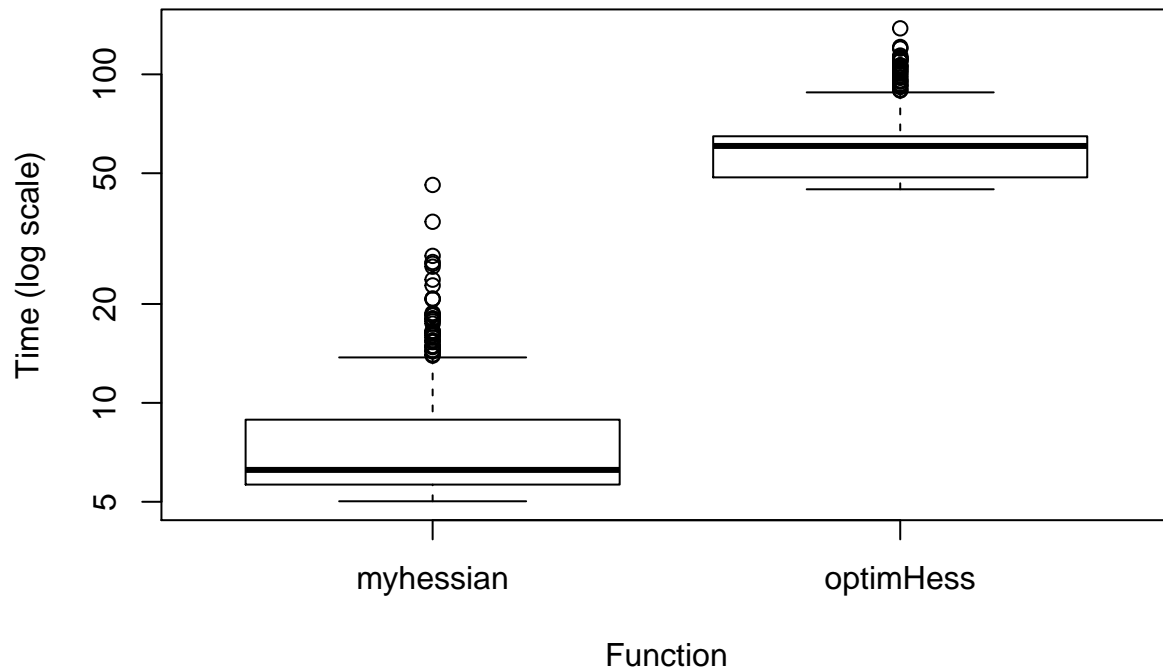
Table 3: Summary of benchmarks for both functions (continued below)

expr	min	lq	mean	median	uq	max
myhessian(bestopt\$par, Y = Y)	5.018	5.64	7.606	6.251	8.89	46.06
optimHess(bestopt\$par, fn = negloglike, Y = Y)	44.68	48.58	59.93	60.53	64.79	138

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.

## Benchmarks for Hessian evaluations



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 parameter 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  $\theta$  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  $\text{Bin}(N, \phi)$  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

Task 3

Question 4

Task 1

Task 2

Task 3

Task 4

Appendix