# STAT 840 A2

## Question 2

Consider such a variance components model $Y \sim N(X\beta,\ \sigma^2(I_n + \tau ZZ^T))$ with $Y = (Y_1, ..., Y_n)^T,\ \beta \in R^p,\ Z \in R^{n\times m},\ m < n$ with rank $m$, where the ordered eigenvalues of $ZZ^T$ is $0 < \lambda_1 \le ... \le \lambda_m$. We may prove that the parameter space of $\tau$ is $(-1/\lambda_m, +\infty)$ by using the fact that the covariance matrix $\sigma^2(I_n + \tau ZZ^T)$ must be positive definite (PD).

Let $Z = SDV^T$ be the singular value decomposition (SVD) of $Z$, where $S \in R^{n\times m}$ with orthonormal column vectors, $V, D = diag(\delta_1, ..., \delta_m) \in R^{m\times m}$, $V$ is orthonormal.

Then,

$$ZZ^T = SDV^TVD^TS^T = SD^2S^T$$

$$\tau ZZ^T = \tau SD^2S^T = S(\tau D^2)S^T$$

Then, we complete the columns of $S$ to form an orthonormal basis for $R^n$, say $S' = [S, M],\ M \in R^{n\times(n-m)}$ such that $S'S'^T = S'^TS' = I_n$, i.e., $S'^{-1} = S'^T$. The feasibility for such a completion is guaranteed by Gram-Schmidt process.

We may define a new diagonal matrix $D' = \tau\begin{pmatrix} D^2 & \\ & O_{n-m}\end{pmatrix} = \begin{pmatrix}\tau D^2 & \\ & O_{n-m}\end{pmatrix} \in R^{n\times n}$, then we have:

$$S'D'S'^T = \begin{pmatrix} S & M\end{pmatrix}\begin{pmatrix}\tau D^2 & \\ & O_{(n-m)\times(n-m)}\end{pmatrix}\begin{pmatrix}S^T \\ M^T\end{pmatrix} =$$
$$\begin{pmatrix} S(\tau D^2) + MO & SO + MO_{(n-m)\times(n-m)}\end{pmatrix}\begin{pmatrix}S^T \\ M^T\end{pmatrix} = \begin{pmatrix} S(\tau D^2) & O\end{pmatrix}\begin{pmatrix}S^T \\ M^T\end{pmatrix} = S(\tau D^2)S^T = \tau ZZ^T$$

Therefore,

$$I_n + \tau ZZ^T = I_n + S'D'S'^T = S'I_nS'^T + S'D'S'^T = S'(I_n + D')S'^T =$$
$$S'\begin{pmatrix}1+\tau\lambda_1 & & & \\ & \ddots & & \\ & & 1+\tau\lambda_m & \\ & & & I_{n-m}\end{pmatrix}S'^{-1}$$

Here, we get the eigendecomposition of the covariance matrix $I_n + \tau ZZ^T$. To make it PD,

$$1 + \tau\lambda_m \ge ... \ge 1 + \tau\lambda_m > 0$$

When $\tau \ge 0$, the above inequalities trivially hold. When $\tau < 0$, we must have such conditions met:

$$1 + \tau\lambda_i > 0,\ \forall i = 1, ..., m$$

$\tau\lambda_i > -1, \ \forall i = 1, ..., m$

Since $\lambda_i > 0, \forall i = 1, ..., m$,

$\tau > -1/\lambda_i, \ \forall \ i = 1, ..., m$

$\tau > \max_{i=1,...,m} -1/\lambda_i = -1/\lambda_m$

Therefore, the parameter space of $\tau$ is $(-1/\lambda_m, +\infty)$.

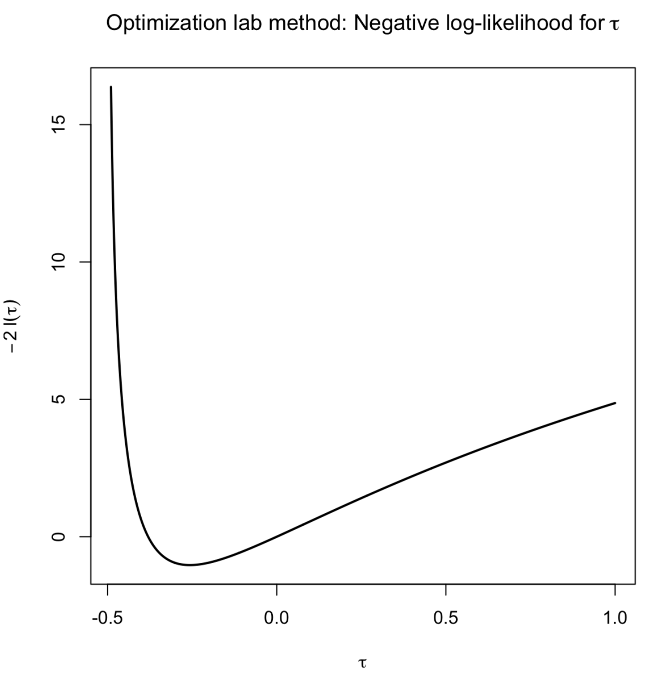
In this question, we implement a descent gradient and Newton's method in R to get the MLE of $\tau$ given the dataset simulated in the optimization lab.

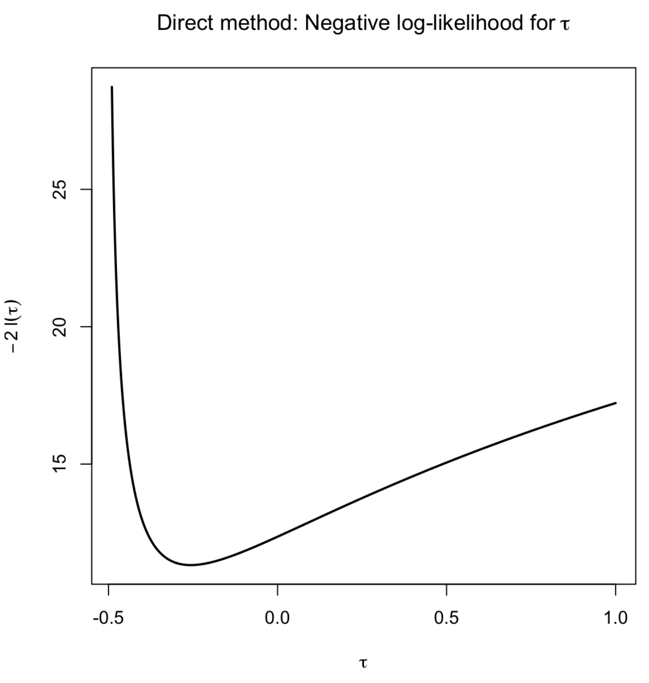
First, we have to implement the direct computation method for getting the negative log-likelihood function value given $\tau$ using the forumula

$$-2l(\tau;\ q) =$$
$$-2\log\left[\frac{\Gamma(\frac{n-p}{2})}{(2\pi)^{(n-p)/2}}|I_{n-p} + \tau U^T ZZ^T U|^{-1/2}\left[q^T (I_{n-p} + \tau U^T ZZ^T U)^{-1} q\right]^{-(n-p)/2}\right]$$

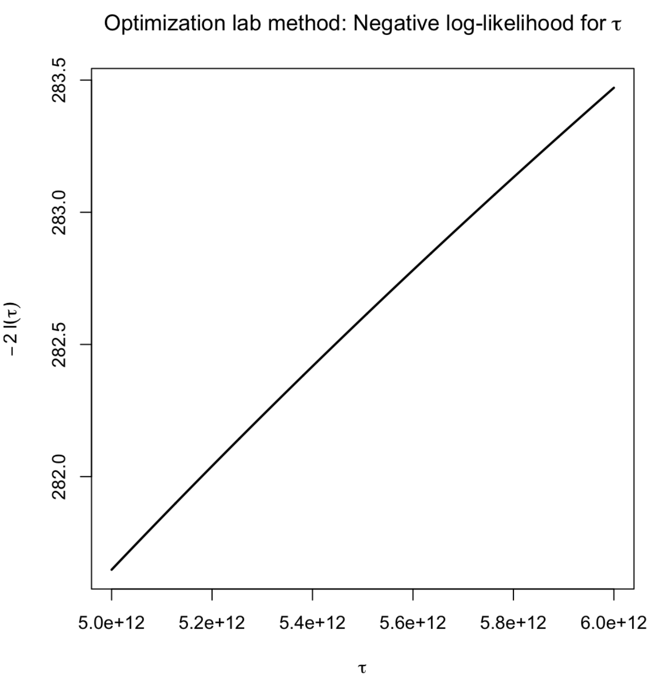
Then, we directly take the gradient and Hessian of the function using numDeriv::grad and numDeriv::hessian in R. Both my methods are defined as functions that take the statistic $q$ as an input and return the MLE of $\tau$. Other inputs include initial value, tolerance and the maximum number of iterations as the stopping criteria. Once the distance between the gradient and 0 is less than the tolerance or the number of iterations reaches the set limit, the algorithm stops. The gradient descent algorithm also has the step size as an input. When updating $\tau$ in the iteration, whenever it is out of the parameter space described in part (a), a reflection operation $\tau \leftarrow -2/\lambda_m - \tau$ is applied to make it satisfy the restriction again.

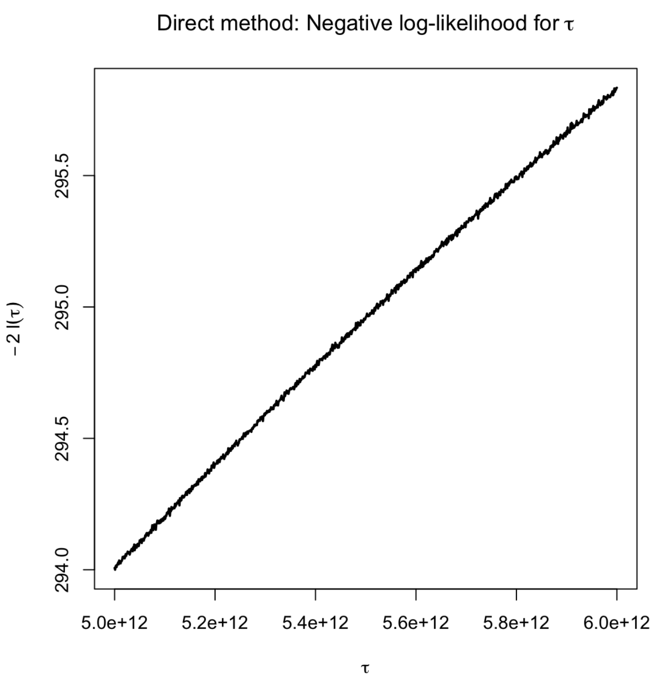
After implementing the direct method in R, I generate a plot of the negative log-likelihood function (NLL) to compare it with the one using the optimization-lab method:



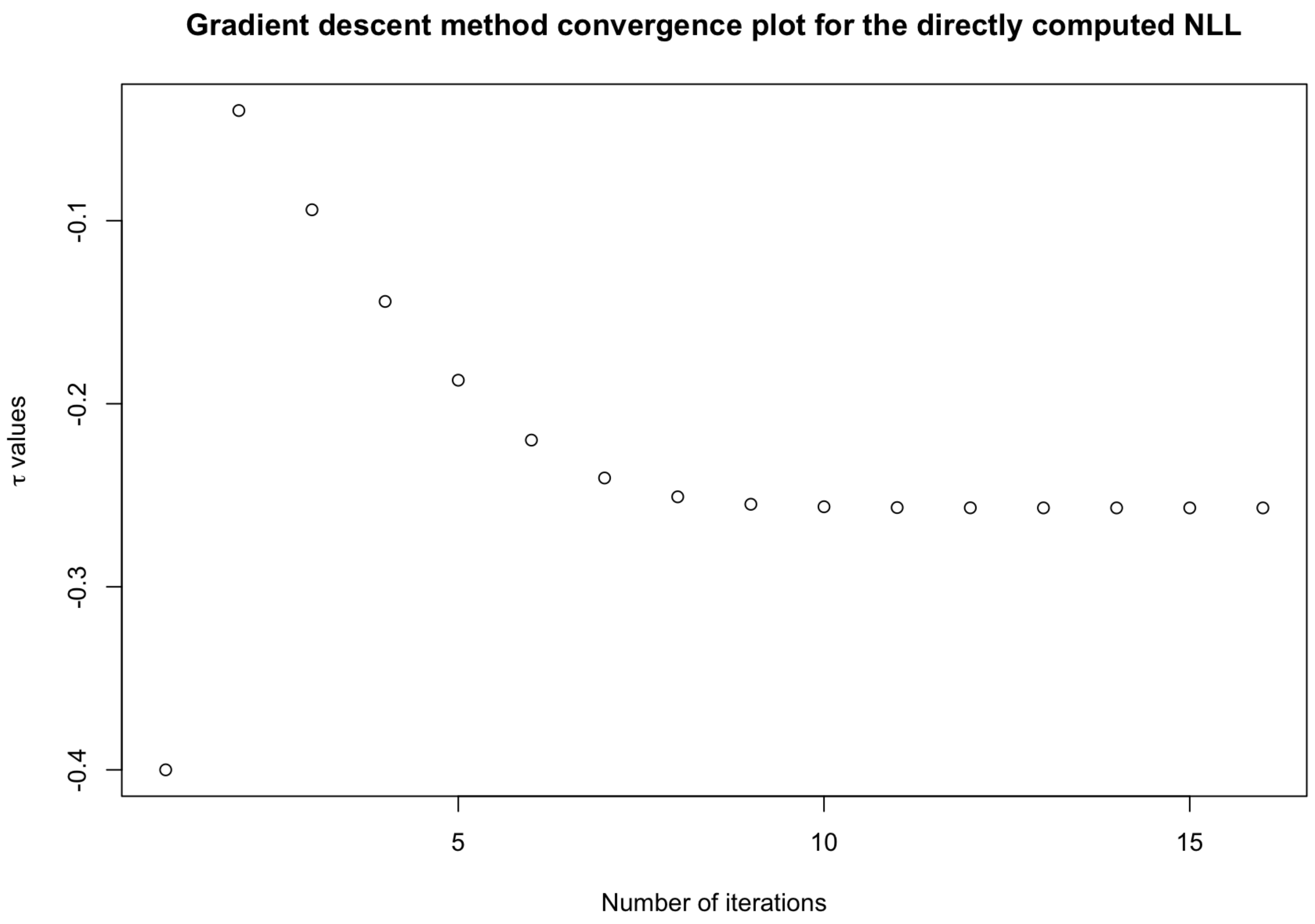


Their shapes are almost identical through inspection. Nonetheless, the direct computing method has very unstable fluctuation behavior when $\tau$ becomes larger (e.g., larger than 5e12):



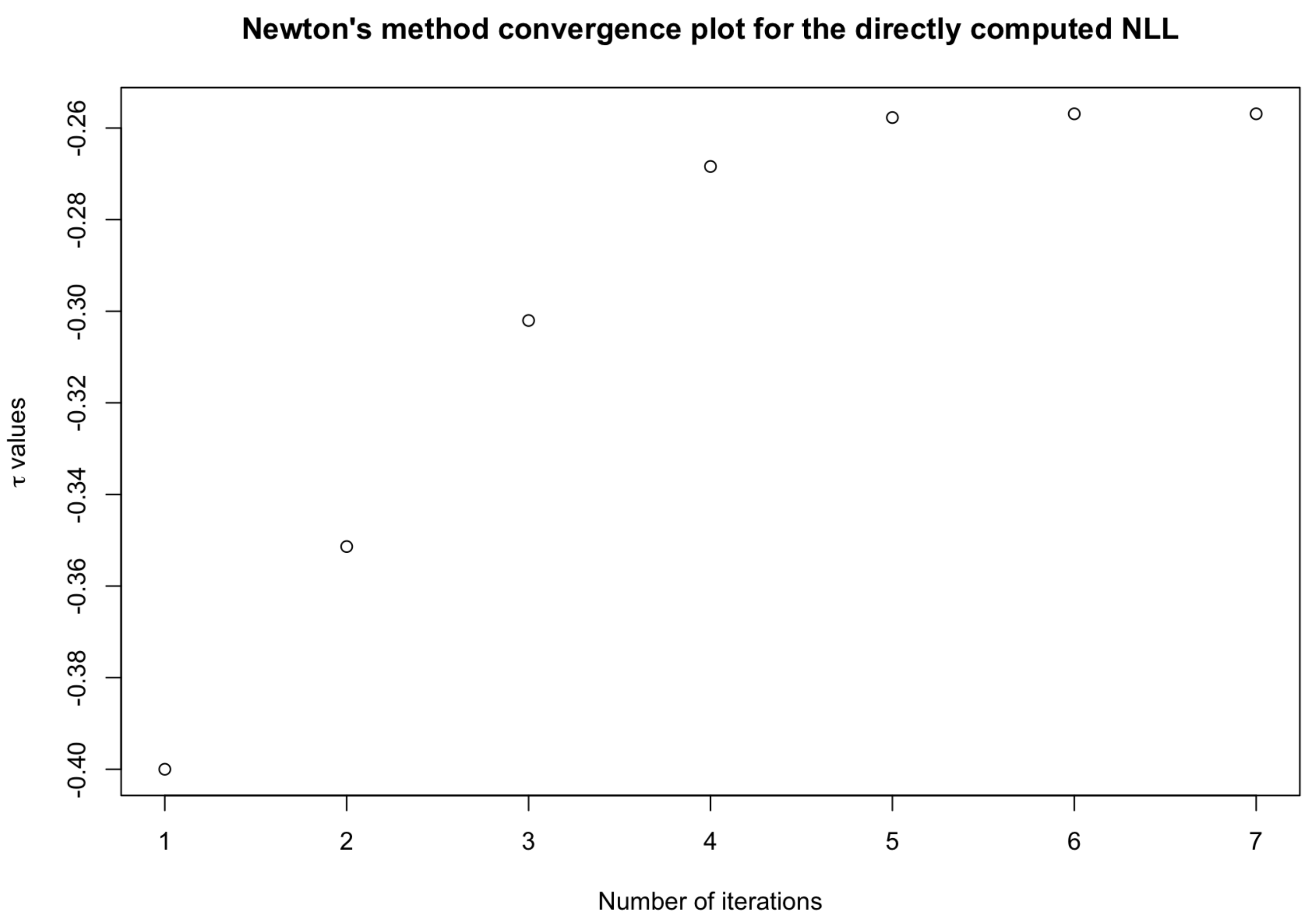


By running the gradient descent algorithm with the initial value and step size set to -0.4 and 0.01, it converges within around 10 iterations. The MLE calculated is about -0.2568479, which is consistent with the inspection and the lab result. The convergence plot is here:



We can see that it first elevates to a high value, then gradually falls down to the MLE.

Then, if we run Newton's method, it converges to the MLE within less iterations:



Newton's method takes about 6 iterations to converge to the MLE, which is better than the 10 iterations for gradient descent. It also converges to -0.2568479, which is exactly the same as the output of the gradient descent method.

Note that the QR decomposition of $X$ will manipulate the columns of $X$ iteratively such that each latter column is orthogonal to all the previous ones, which forms an orthogonal basis for the column space of $X$, which is a $p$ -dimensional subspace of $R^n$ since $n > p$.

Now, we apply the version of QR decomposition that completes the basis of the whole $R^n$ space by adding extra linearly independent columns:

$X = S'R' = \begin{pmatrix} S\ U \end{pmatrix} \begin{pmatrix} R \\ 0 \end{pmatrix}$, where $S \in R^{n\times p},\ R \in R^{p\times p}$, which is the original version of the QR decomposition, and $U \in R^{n\times n-p}$ is the completed part of the basis of $R^n$. As a whole, the matrix $S' = \begin{pmatrix} S\ U \end{pmatrix}$ is orthonormal.

Since $S' = \begin{pmatrix} S\ U \end{pmatrix}$ is orthonormal, we have

$$S'^T X = S'^T S' R' = R'$$

$$\begin{pmatrix} S^T \\ U^T \end{pmatrix} X = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} S^T X \\ U^T X \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix}$$

$U^T X = 0$

Hence, we know that $U$ is composed by the $n-p$ completed basis vectors as its columns.

When using QR decomposition, $S'$ is not directly formed, so we may only use the last $(n-p)$ columns of it (i.e., the columns of $U$) when required for computing $U^T Z$.

Also, since $Z$ is sparse with band $n/m - 1$, the storage of it only costs $O(n)$.

We can further verify that the computational cost for $U^T Z$ is $O(n(n-p))$:

First, since $U^T Z = U^T[z_1, ..., z_m] = [U^T z_1, ..., U^T z_m]$, we only need to add up $n/m$ columns of $U^T$ (i.e., rows of $U$) for each column of $Z$, which can even easily be done in-place to avoid storing another matrix. The dimension of each row of $U$ is $R^{(n-p)}$, so each operation for $U^T z_i, i = 1, 2, ..., m$ is $O((n-p) \times (n/m))$, which means that the overall cost is $O((n-p) \times (n/m) \times m) = O(n(n-p))$.

Therefore, to implement such an algorithm in R, we first get the QR decomposition of $X$ using qr(X). Then, we input the QR decomposition object and each column of matrix $Z$ into qr.qty() function and take the last $n-p$ rows to get $U^T Z$ (p.s. Actually qr.qty accepts a matrix Z, which is easier. Here I didn't use the form, because I want to make my words clearer).

In this problem, we derive an alternative more convenient formula for computing the likelihood of $\tau$.

First, define the inner matrix $A(\tau) = I_{n-p} + \tau U^T Z Z^T U$. Let $U^T Z = SDV^T$ be the singular value decomposition (SVD) of $U^T Z$, where $S \in R^{(n-p)\times m}$ with orthonormal column vectors, $V, D = diag(\delta_1, ..., \delta_m) \in R^{m\times m}$, $V$ is orthonormal.

By plugging in the SVD into the matrix expression, we get:

$$A(\tau) = I_{n-p} + \tau(SDV^T)(SDV^T)^T = I_{n-p} + \tau SDV^T V D^T S^T = I_{n-p} + \tau SDI_m D^T S^T = I_{n-p} + S(\tau D^2)S^T$$

Then, we complete $S' = \begin{pmatrix} S & B \end{pmatrix}$, $B \in R^{(n-p)\times(n-p-m)}$ as an orthornormal basis of the whole $R^{n-p}$. i.e., $S'^T S' = S' S'^T = SS^T + BB^T = I_{n-p}$.

We may also define a new diagonal matrix $D' = \begin{pmatrix} \tau D^2 & \\ & O_{n-p-m} \end{pmatrix}$, then

$$S' D' S'^T = \begin{pmatrix} S & B \end{pmatrix} \begin{pmatrix} \tau D^2 & \\ & O_{n-p-m} \end{pmatrix} \begin{pmatrix} S^T \\ B^T \end{pmatrix} = \begin{pmatrix} S(\tau D^2) & O \end{pmatrix} \begin{pmatrix} S^T \\ B^T \end{pmatrix} = S(\tau D^2)S^T$$

Hence,

$$A(\tau) = I_{n-p} + S' D' S'^T = S' S'^T + S' D' S'^T = S' I_{n-p} S'^T + S' D' S'^T = S'(I_{n-p} + D')S'^T = S' \begin{pmatrix} 1+\tau\delta_1^2 & & & \\ & \ddots & & \\ & & 1+\tau\delta_m^2 & \\ & & & I_{n-p-m} \end{pmatrix} S'^T = S' \begin{pmatrix} 1+\tau\lambda_1 & & & \\ & \ddots & & \\ & & 1+\tau\lambda_m & \\ & & & I_{n-p-m} \end{pmatrix} S'^T$$

Note that since $S'^{-1} = S'^T$, the above form is just an eigendecomposition of $A(\tau)$. The determinant of the matrix can be obtained by multiplying all eigenvalues of the matrix, which is

$$|A(\tau)| = \prod_{j=1}^{m} (1 + \tau\lambda_j)\,.$$

The inverse can also be easily obtained:

$$A(\tau)^{-1} = S' \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & & \\ & \ddots & & \\ & & \frac{1}{1+\tau\lambda_m} & \\ & & & I_{n-p-m} \end{pmatrix} S'^T =$$

$$\begin{pmatrix} S & B \end{pmatrix} \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & & \\ & \ddots & & \\ & & \frac{1}{1+\tau\lambda_m} & \\ & & & I_{n-p-m} \end{pmatrix} \begin{pmatrix} S^T \\ B^T \end{pmatrix}$$

For convenience, we denote

$$\Sigma = \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{1+\tau\lambda_m} \end{pmatrix}$$, then

$$A(\tau)^{-1} = \begin{pmatrix} S & B \end{pmatrix} \begin{pmatrix} \Sigma & \\ & I_{n-p-m} \end{pmatrix} \begin{pmatrix} S^T \\ B^T \end{pmatrix} = \begin{pmatrix} S\Sigma & B \end{pmatrix} \begin{pmatrix} S^T \\ B^T \end{pmatrix} = S\Sigma S^T + BB^T = S\Sigma S^T + I_{n-p} - SS^T = I_{n-p} - (SS^T - S\Sigma S^T) = I_{n-p} - S(I_m - \Sigma)S^T$$

The quadratic term in the likelihood function thus:

$$q^T A(\tau)^{-1} q = q^T (I_{n-p} - S(I_m - \Sigma)S^T) q = q^T q - q^T S(I_m - \Sigma)S^T q = \|q\|_2^2 - q^T S \begin{pmatrix} 1 - \frac{1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & 1 - \frac{1}{1+\tau\lambda_m} \end{pmatrix} S^T q = 1 - q^T S \begin{pmatrix} \frac{\tau\lambda_1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{\tau\lambda_m}{1+\tau\lambda_m} \end{pmatrix} S^T q = 1 - \tau \left[ q^T S \begin{pmatrix} \frac{\delta_1^2}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{\delta_m^2}{1+\tau\lambda_m} \end{pmatrix} S^T q \right] = 1 - \tau \left[ q^T S D^T \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{1+\tau\lambda_m} \end{pmatrix} D S^T q \right]$$

Define $\bar{q} = DS^T q$, we have

$$q^T A(\tau)^{-1} q = 1 - \tau \left[ \bar{q}^T \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{1+\tau\lambda_m} \end{pmatrix} \bar{q} \right] = 1 - \tau \left[ \begin{pmatrix} \bar{q}_1 & \ldots \bar{q}_m \end{pmatrix} \begin{pmatrix} \frac{1}{1+\tau\lambda_1} & & \\ & \ddots & \\ & & \frac{1}{1+\tau\lambda_m} \end{pmatrix} \begin{pmatrix} \bar{q}_1 \\ \vdots \\ \bar{q}_m \end{pmatrix} \right] = 1 - \tau \sum_{j=1}^{m} \frac{\bar{q}_j^2}{1 + \tau\lambda_j}$$

Finally, we combine the derived identities together to redefine a much more convenient negative log-likelihood function formula:

$$-2l'(\tau;q) =$$
$$-2\log\left(\frac{\Gamma(\frac{n-p}{2})}{(2\pi)^{(n-p)/2}}|I_{n-p}+\tau U^TZZ^TU|^{-1/2}\left[q^T(I_{n-p}+\tau U^TZZ^TU)^{-1}q\right]^{-(n-p)/2}\right) =$$
$$-2\log\left(\frac{\Gamma(\frac{n-p}{2})}{(2\pi)^{(n-p)/2}}|A(\tau)|^{-1/2}\left[q^TA(\tau)^{-1}q\right]^{-(n-p)/2}\right) =$$
$$-2\log\Gamma\left(\frac{n-p}{2}\right)+2\log(2\pi)^{(n-p)/2}+\log|A(\tau)|+(n-p)\log\left[q^TA(\tau)^{-1}q\right]$$

After dropping all the terms that are irrelevant to $\tau$ (and thus minimizing the function with respect to $\tau$ ), we have:

$$-2l(\tau;q) = \log|A(\tau)|+(n-p)\log\left[q^TA(\tau)^{-1}q\right] = \log\prod_{j=1}^{m}(1+\tau\lambda_j)+(n-p)\log\left(1-\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}\right) = \sum_{j=1}^{m}\log(1+\tau\lambda_j)+(n-p)\log\left(1-\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}\right)$$

In the last question, we derived a more convenient form for calculating the negative log-likelihood function:

$$N(\tau) = -2l(\tau;q) = \sum_{j=1}^{m}\log(1+\tau\lambda_j)+(n-p)\log\left(1-\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}\right)$$

Now, to compute the MLE using gradient descent and Newton's method more efficiently, we should first compute the first and second derivatives by hand and hardcode it into R:

$$\frac{\partial N}{\partial\tau} = \sum_{j=1}^{m}\frac{\lambda_j}{1+\tau\lambda_j}+(n-p)\frac{1}{1-\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}}\left(-\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}+\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2\lambda_j}{(1+\tau\lambda_j)^2}\right) =$$
$$\sum_{j=1}^{m}\frac{\lambda_j}{1+\tau\lambda_j}+(n-p)\frac{\sum_{j=1}^{m}\left[\frac{\bar{q}_j^2}{1+\tau\lambda_j}\left(\frac{\tau\lambda_j}{1+\tau\lambda_j}-1\right)\right]}{1-\tau\sum_{j=1}^{m}\frac{\bar{q}_j^2}{1+\tau\lambda_j}}$$

For computing and representation convenience, we define $T_1(j) = \frac{\lambda_j}{1+\tau\lambda_j}$, $T_2(j) = \frac{\bar{q}_j^2}{1+\tau\lambda_j}$, so the above gradient is re-represented by:

$$\frac{\partial N}{\partial\tau} = \sum_{j=1}^{m}T_1(j)+(n-p)\frac{\sum_{j=1}^{m}\left[T_2(j)(\tau T_1(j)-1)\right]}{1-\tau\sum_{j=1}^{m}T_2(j)}$$

To compute the second derivative, the second fraction term is tedious, so we should pre-derive some derivatives to get the term derivative beforehand:

$$T_1'(j) = -\frac{\lambda_j^2}{(1+\tau\lambda_j)^2}$$
$$T_2'(j) = -\frac{\bar{q}_j^2\lambda_j}{(1+\tau\lambda_j)^2}$$

Define $u(j) = T_2(j)(\tau T_1(j)-1)$, then

$$u'(j) = T_2'(j)(\tau T_1(j) - 1) + T_2(j)(T_1(j) + \tau T_1'(j))$$

Then, define the derivative of the fraction term as:

$$F' = \frac{\left[\sum_{j=1}^{m} u'(j)\right]\left(1 - \tau \sum_{j=1}^{m} T_2(j)\right) + \left[\sum_{j=1}^{m} u(j)\right]\left[\tau \sum_{j=1}^{m} T_2'(j) + \sum_{j=1}^{m} T_2(j)\right]}{\left(1 - \tau \sum_{j=1}^{m} T_2(j)\right)^2}$$

Finally, we obtain the second derivative here:

$$\frac{\partial^2 N}{\partial \tau^2} = \sum_{j=1}^{m} T_1'(j) + (n-p)F'$$

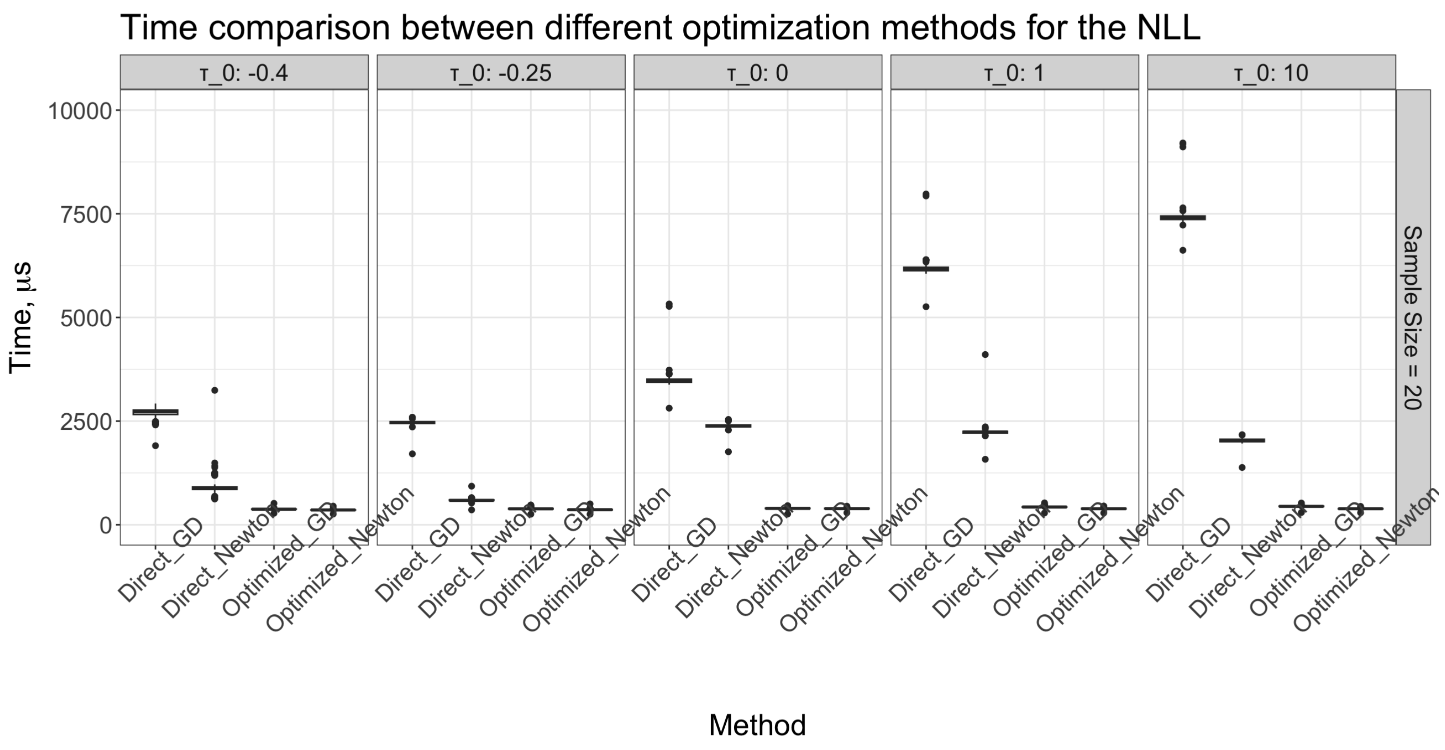
, where $T_1'(j) = -\frac{\lambda_j^2}{(1+\tau\lambda_j)^2}$, $F'$ and other identities in it is as shown above.

To hardcode the derivatives into R, we first pre-compute vectors $T_1,\ T_2,\ T_1',\ T_2'$ and their sums. Then, we should pre-compute $u',\ F'$ accordingly. Based on those quantities, we finally combine them together to get the derivatives using the above expression.

The solutions above have already been implemented and tested by comparing with the results from numDeriv::grad and numDerive::hessian to guarantee correctness.

Then, we can test the time comparison comparison between the 4 methods: Gradient descent (GD) optimized NLL obtained by direct computing (Direct_GD), Newton's method optimized NLL obtained by direct computing (Direct_Newton), GD optimized NLL obtained by the efficient computing method (Optimized_DG), and Newton's method optimized NLL obtained by the efficient computing method (Optimized_Newton).

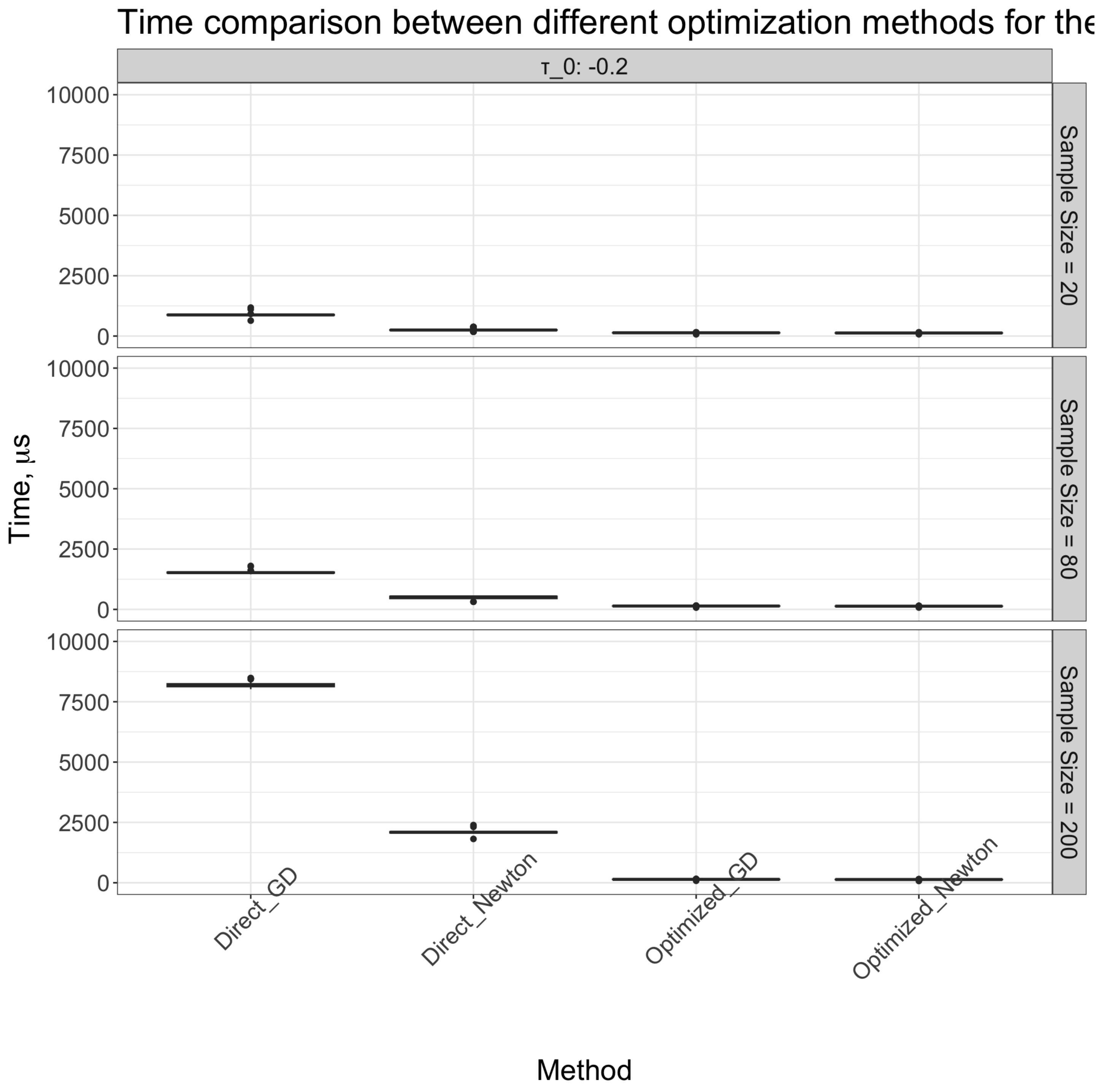
To be clearer, we only compare the sensitivity of the 4 methods to one of $\tau$ and sample size each time:



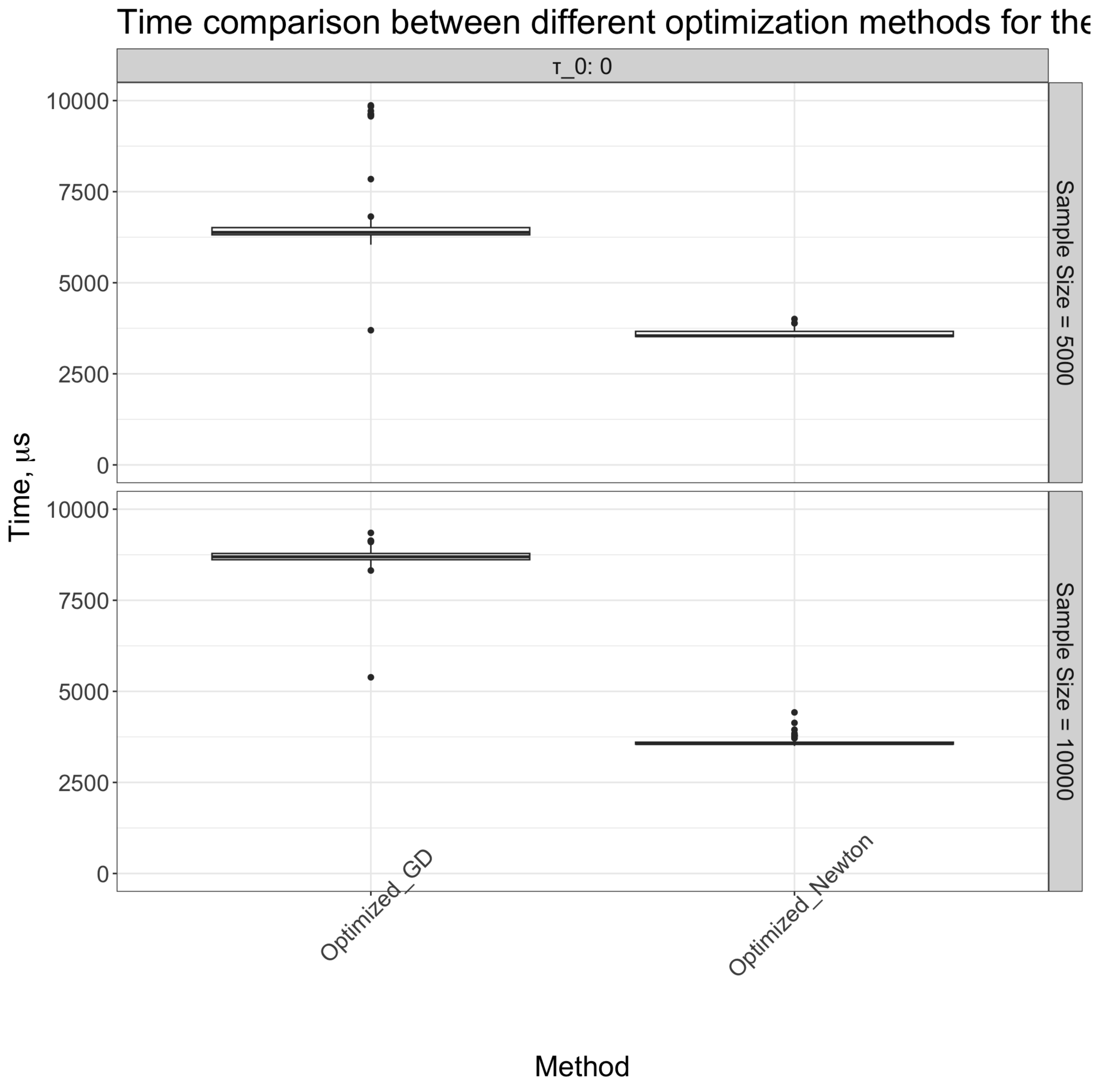
The experiment time for each algorithm has been adjusted appropriately to make the visualization easy. Also, the sample size holds constant across different methods, so that we only see the effects of the initial value of $\tau$ here. According to the comparison plot, the methods with the efficient NLL computing all outperform the ones with direct computing. GD and Newton's methods are not sensitivity for the efficiently computed NLL. This is because the analytical implementation of the gradients and hessians make the algorithms too fast to be differentiated. To expand their differences, we should test much larger $\tau_0$ , which doesn't seem meaningful, since in reality, we never select an initial value that is extremely far from the MLE. Nonetheless, the expectation is that there will be 2 cases: 1) When the initial value is too large for the floating system to handle, then the numerical methods will collapse with some very weird results, or 2) Newton's method will be much faster than GD, since according to the shape of NLL plot at a very large $\tau$ value, the curve is almost linear, which means that the second derivative should be very small, yielding a very large step size to help Newton's method rapidly reach the meaningful region.

The in-group difference for the algorithms using directly computed NLL is interesting. For any initial value, GD is slower than Newton. When the initial value is farther from the MLE, GD becomes slower. When the initial value is farther, at first, Newton's method becomes slower, but if you keep deviating the initial value from the MLE, Newton's method becomes faster. This is because Newton's method leverages the second derivative information. When the gradient of the NLL is flat, it takes a larger step, which helps it get rid of the slow areas, whereas GD holds the step size as constant.

Then, we compare the algorithms with different sample sizes:



I fixed the $\tau$ as -0.2, and changed the sample sizes to 20, 80, and 200. Still, the sample sizes are still too trivial for the algorithms with efficient NLL computing. But we can see that GD is more sensitive than Newton's method by comparing the algorithms with direct NLL computing. To exploit the limit of the latter 2 algorithms with the efficient NLL computing, we run them on a much larger sample size:



This time, we set the sample sizes to be 5000 and 10000, and finally we can differentiate between GD and Newton's method on the efficiently computed NLL. GD is slower than Newton's method, and it's more sensitive w.r.t. sample size than Newton's method.

## Question 3

Here, we obtain the probability density function of a zero-inflated Binomial model of random variable:

$$Y \sim \begin{cases} 0 & \text{with probability } \phi \\ Binom(N, p) & \text{with probability } (1-\phi) \end{cases} \tag{1}$$

First, we calculate the probability $P(Y=0)$, which is the case where either a village eliminates ring worms (i.e., the distribution is just 0), or the village happens to have no infections (i.e., the distribution itself is a binomial model, but the realization happens to be 0 infection). Hence, following the probability of the union of the two events,

$P(Y_i=0)=\phi+(1-\phi)(1-p)^{N_i}$

Then, for any village where the infection count is positive, the ring warms there can't be eliminated. The infection count then must be realized from the binomial model, so in that case,

$P(Y_i=y)=(1-\phi)\binom{y}{N_i}p^y(1-p)^{N_i-y}$

Overall, the probability density function is:

$$P(Y=y)=\begin{cases}\phi+(1-\phi)(1-p)^{N_i} & \text{with probability } \phi \\ (1-\phi)\binom{y}{N_i}p^y(1-p)^{N_i-y} & \text{with probability } (1-\phi)\end{cases} \tag{2}$$

To estimate $p, \phi$ given the observed data in Loaloa, I use an MLE method. We should derive the negative log-likelihood function of the parameters given the data, then use some optimization method to find the global minimizer of the function.

Assuming that the observations are i.i.d., then the likelihood function is:

$l'(p,\phi;y)=\prod_{i\in I_{y^+}}\left[(1-\phi)\binom{y_i}{N_i}p^{y_i}(1-p)^{N_i-y_i}\right]\prod_{i\in I_{y=0}}\left[\phi+(1-\phi)(1-p)^{N_i}\right]=(1-\phi)^{|I_{y^+}|}\prod_{i\in I_{y^+}}\binom{y_i}{N_i}p^{\sum_{i\in I_{y^+}}y_i}(1-p)^{\sum_{i\in I_{y^+}}(N_i-y_i)}\prod_{i\in I_{y=0}}\left[\phi+(1-\phi)(1-p)^{N_i}\right]$

, where $I_{y^+}=\{i\in N|y_i>0\},\ I_{y=0}=\{i\in N|y_i=0\}$.

The log-likelihood function is:

$l(p,\phi;y)=\log\left[(1-\phi)^{|I_{y^+}|}\prod_{i\in I_{y^+}}\binom{y_i}{N_i}p^{\sum_{i\in I_{y^+}}y_i}(1-p)^{\sum_{i\in I_{y^+}}(N_i-y_i)}\prod_{i\in I_{y=0}}\left[\phi+(1-\phi)(1-p)^{N_i}\right]\right]=|I_{y^+}|\log(1-\phi)+\sum_{i\in I_{y^+}}\log\binom{y_i}{N_i}+\log p\sum_{i\in I_{y^+}}y_i+\log(1-p)\sum_{i\in I_{y^+}}(N_i-y_i)+\sum_{i\in I_{y=0}}\log\left[\phi+(1-\phi)(1-p)^{N_i}\right]$

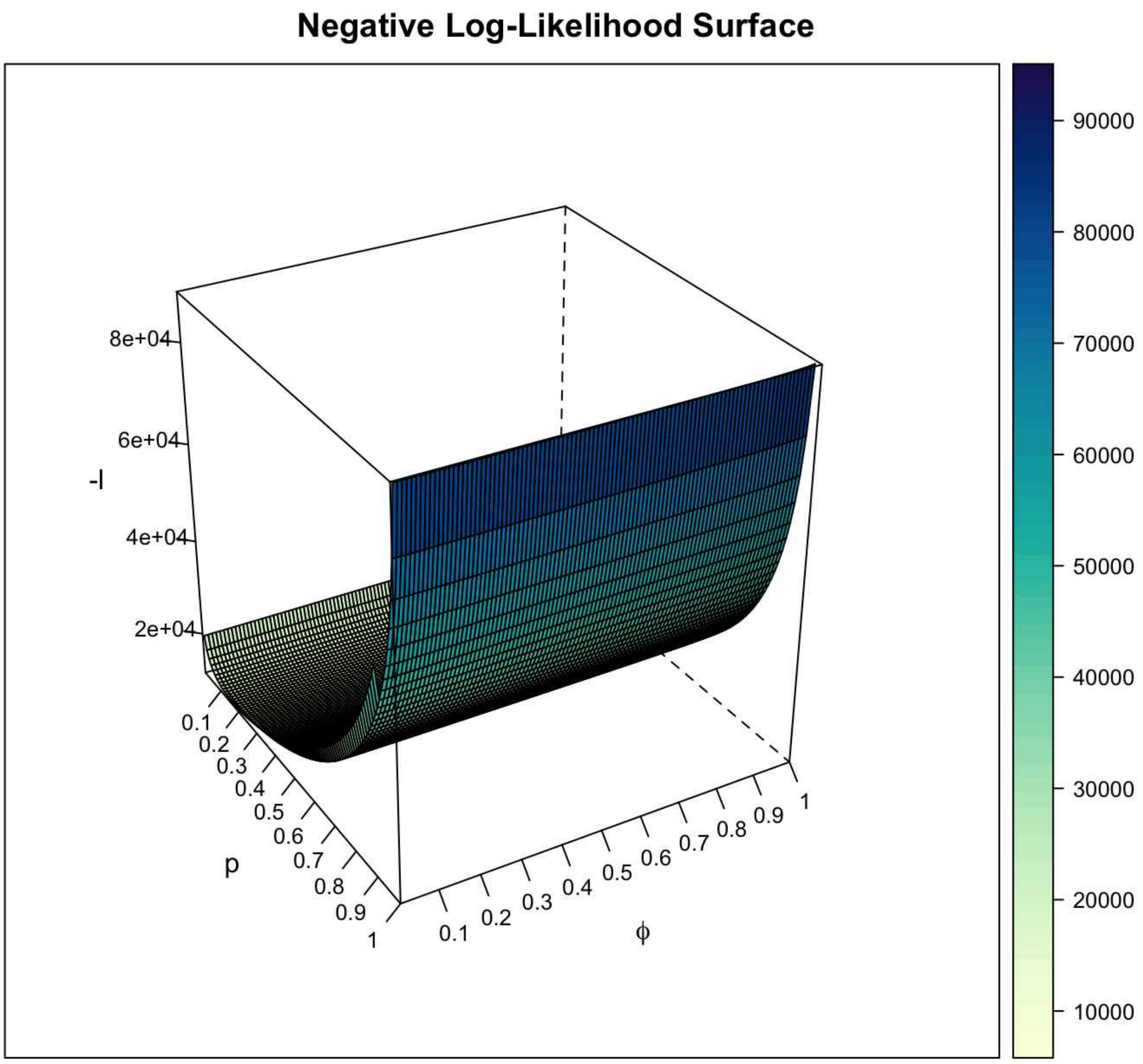
The negative log-likelihood function (NLL) is thus:

$-l(p,\phi;y)=-|I_{y^+}|\log(1-\phi)-\sum_{i\in I_{y^+}}\log\binom{y_i}{N_i}-\log p\sum_{i\in I_{y^+}}y_i-\log(1-p)\sum_{i\in I_{y^+}}(N_i-y_i)-\sum_{i\in I_{y=0}}\log\left[\phi+(1-\phi)(1-p)^{N_i}\right]$

By removing the irrelevant terms in the NLL, we have

$$-l(p,\phi;y) = -|I_{y^+}|\log(1-\phi) - \log p \sum_{i\in I_{y^+}} y_i - \log(1-p)\sum_{i\in I_{y^+}}(N_i - y_i) - \sum_{i\in I_{y=0}} \log\left[\phi + (1-\phi)(1-p)^{N_i}\right]$$

After implementing the NLL in R with the Loaloa dataset, we may visualize it for the interval $(p,\phi) \in [0,1]\times[0,1]$:

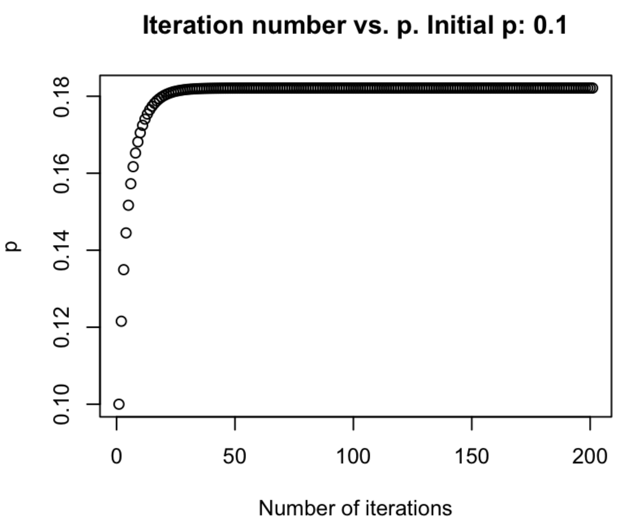


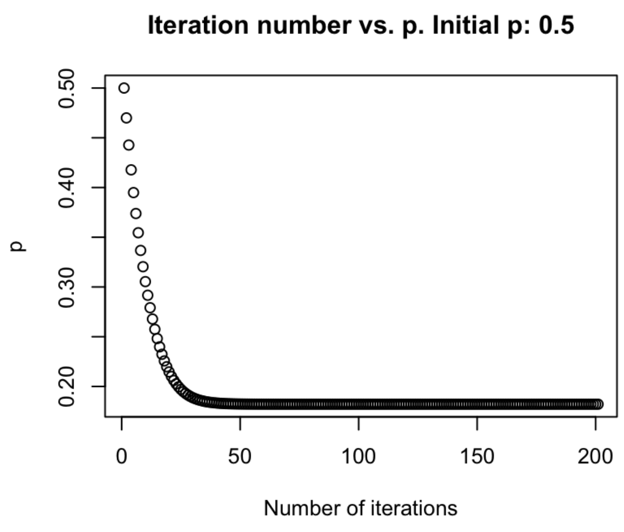
The functions seem approximately convex and the global minimizer should lie in the column around $p = 0.2$, where the $\phi$ value doesn't seem obvious here.

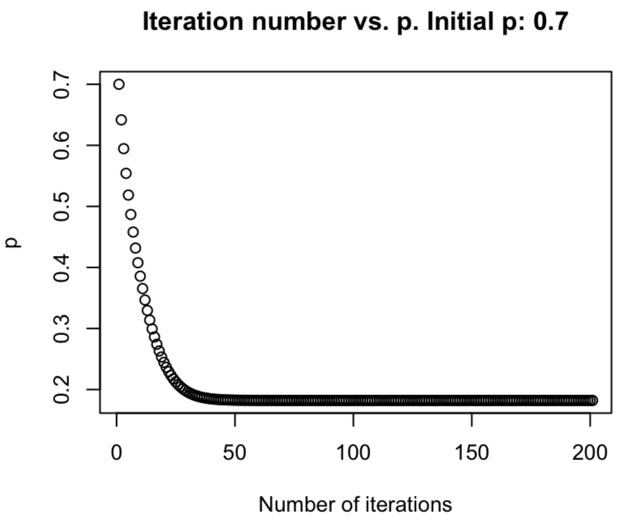
Now, we may apply numerical optimization techniques to solve for the multivariable MLE. First, I implemented a gradient descent method as what we did in question 2. The derivatives are obtained numerically in R. By setting initial values $p = 0.2, \phi = 0.2$, and step size $0.000001$, the MLE is returned within 3 seconds. The result shows that the MLE is approximately $\hat{p} = 0.1821376, \hat{\phi} = 0.1066304$, which is consistent with our observation.

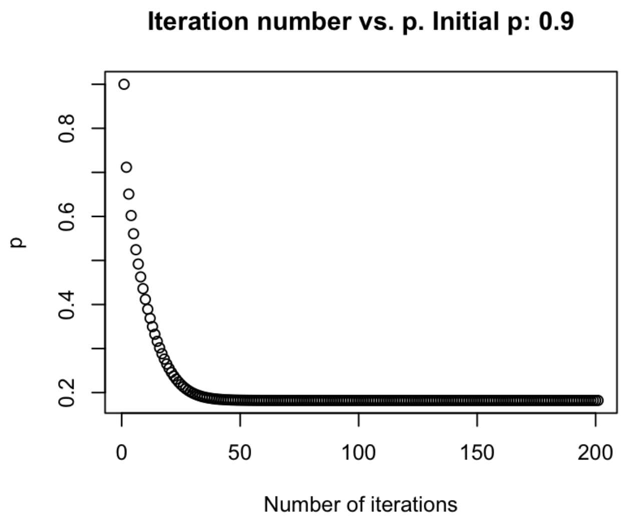
To test the robustness of the method and its sensitivity to the initial values and step size, we generate some plots for convergence of the MLE under different conditions.

First, we explore gradient descent algorithm's performance on the task. We first set the step size to be $0.000001$ and set the maximum number of iterations as 200. When we fix the initial value of $\phi$ to be $0.5$ and change the initial value of $p$ , the convergence plot is:



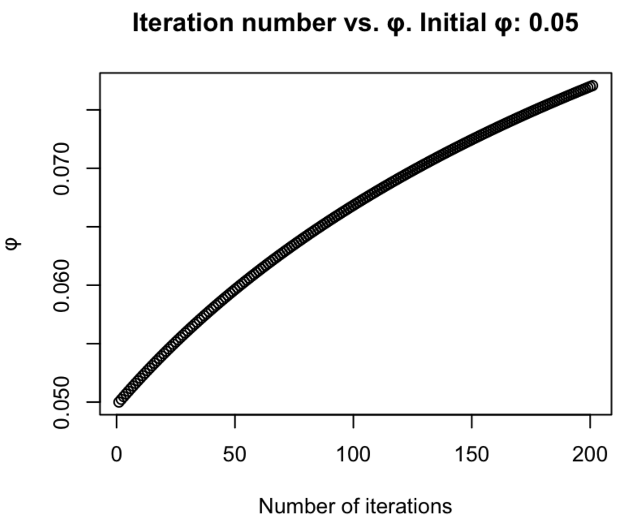


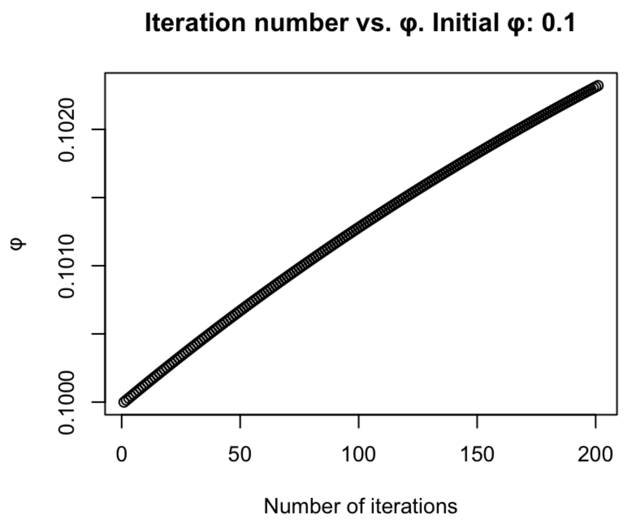


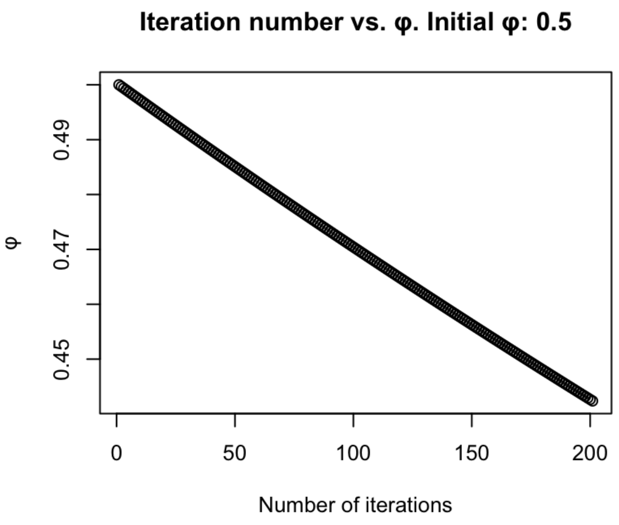


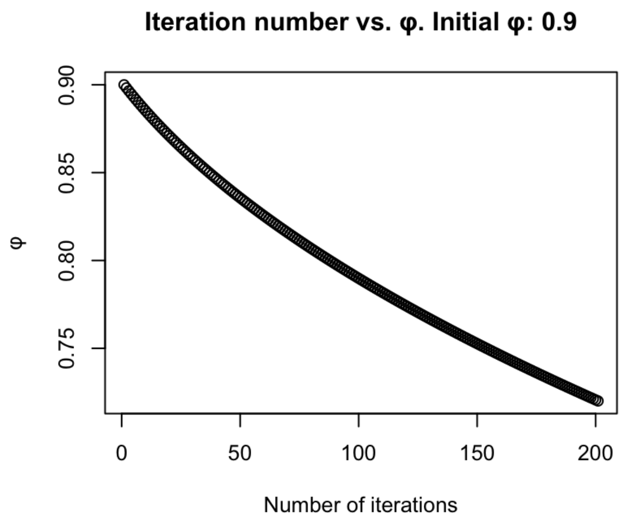
We can observe that when the initial $p$ is farther than the MLE, it takes slightly more iterations to converge. But it still converges to MLE within 40 iterations no matter how far it is from the MLE.

Then, we hold other settings of gradient descent unchanged and the initial value of $p$ fixed as $0.5$ , and plot the convergence plot for $\phi$ with different initial values:



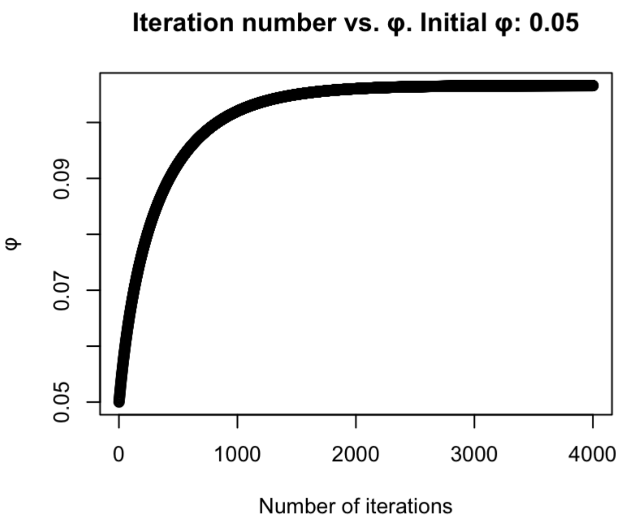


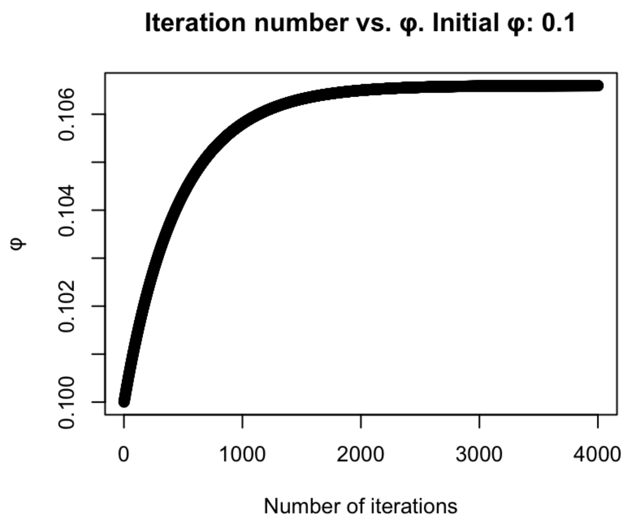


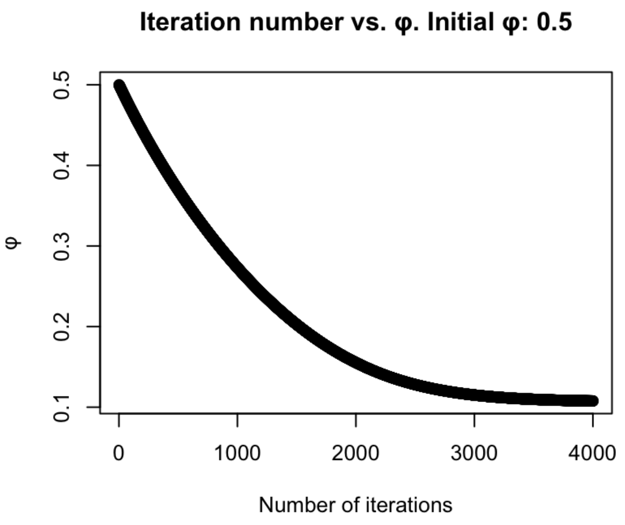


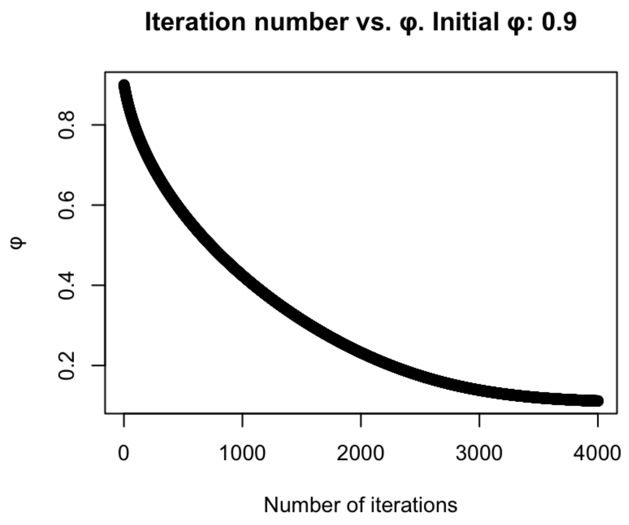
The plots show that the convergence of $\phi$ takes much more iterations than p. This is expected since the rate of change of the likelihood surface is very flat with respect to $\phi$ , which makes the magnitude of parameter update small. Hence, we may either expand the step size or set it to allow more iteration.

This time, we increase the maximum number of iterations to 4000 and regenerate the convergence plot of $\phi$ :









The parameter with any initial value eventually converges to the MLE within 4000 iterations. But the ones that are closer to the MLE converge significantly faster than the more distant ones. If the initial value is within 0.1 distance with the MLE, they can converge in less than 3000 iterations. Another phenomenon is that the convergence rate decays as the parameter is close to the MLE. This is because the magnitude of the slope/gradient of the NLL decays very close to 0 when it reaches the MLE.

Anyways, the method is robust and produces very repeatable results. It's also efficient enough for such a small sample size and model setting. The results are reasonable intuitively, since the sample proportion of zero-infected villages is 0.1193182, which is fairly close but slightly larger than $\hat{\phi} = 0.1115790$ that we obtain, which is expected. The sample proportion of infected people out of all examined people (merging all infected villages) is about 0.1821377. The number is almost equivalent to our $\hat{p} = 0.1821376$, which is expected and should represent the general infection rate of the disease (no matter which village you are from).

In summary, we first obtain the NLL

$$-l(p, \phi; y) = -|I_{y^+}| \log(1-\phi) - \sum_{i \in I_{y^+}} \log \binom{y_i}{N_i} - \log p \sum_{i \in I_{y^+}} y_i - \log(1-p) \sum_{i \in I_{y^+}} (N_i - y_i) - \sum_{i \in I_{y=0}} \log\left[\phi + (1-\phi)(1-p)^{N_i}\right]$$

as always and implement it in R. Then, we implement a gradient descent method through numerical gradients. The MLE final converges to a reasonable value: $\hat{p} = 0.1821376, \hat{\phi} = 0.1066304$ with 4000 iterations, with the step size setting to $0.000001$. The MLE of p converges much faster than $\hat{\phi}$, which only takes less than 40 iterations. The initial value setting doesn't influence the convergence speed of p significantly. The MLE of $\hat{\phi}$ is much slower than p, which takes about 4000 iterations for such a fine step size. The convergence is sensitive to the initial value. Converging with an initial value that is farthest to the MLE takes about 1000 more iterations than the one with an initial value that is within 0.1 distance with the MLE.

In this question, we try to relieve the complexity of the model by replacing the Binomial model with a Bernoulli model, so we now only model whether a village is infected or not if the ring worms are not eliminated. The number of infections is no longer our concern.

The random variable for representing whether an infection occurs is:

$$X \sim \begin{cases} 0 & \text{with probability } \phi \\ Bernoulli(p) & \text{with probability } (1-\phi) \end{cases} \tag{3}$$

Hence, the probability function now becomes:

$$P(X=x) = \begin{cases} \phi \ + \ (1-\phi)(1-p) & x = 0 \\ (1-\phi)p & x > 0 \end{cases} \tag{4}$$

Still, the reasoning for such a probability function is from a case analysis: The occurrence of infections fails is the union of 1) the worms are eliminated, or 2) the worms are never eliminated but no one gets infected; Otherwise, the worm is not eliminated and there are infectious occurrences.

Now, we may obtain the NLL for the parameters $p, \phi$:

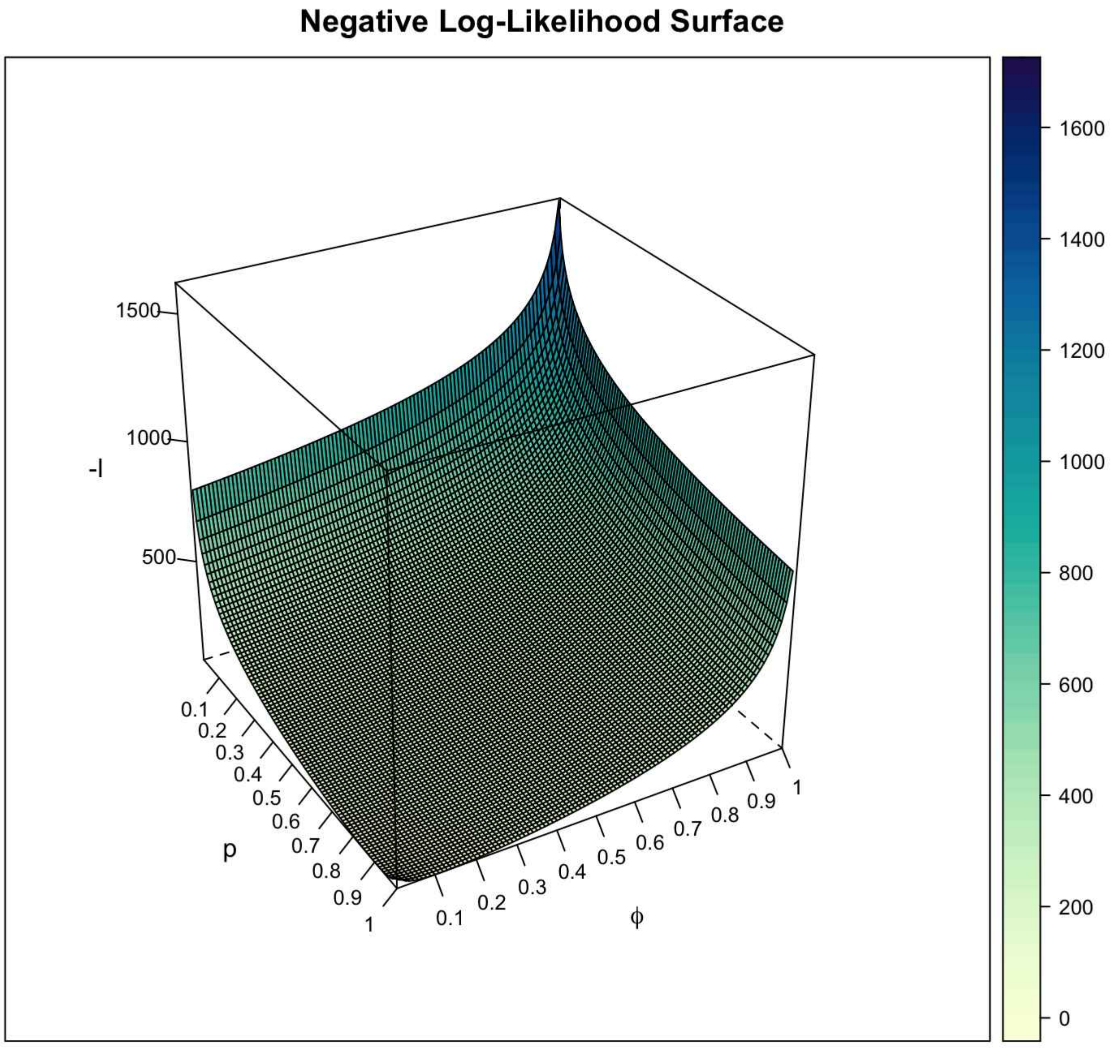
$$L(p,\phi;y) = \prod_{i \in I_{y=0}} [\phi \ + \ (1-\phi)(1-p)] \prod_{i \in I_{y^+}} (1-\phi)p$$

$$l(p,\phi) = \log L = \sum_{i \in I_{y=0}} \log{(\phi + (1-\phi)(1-p))} + \sum_{i \in I_{y=0}} \log{(1-\phi)p} =$$

$$|I_{y=0}| \log{(\phi + (1-\phi)(1-p))} + |I_{y^+}| \log{(1-\phi)p}$$

$$-l(p,\phi) = \log L = -|I_{y=0}| \log{(\phi + (1-\phi)(1-p))} - |I_{y^+}| \log{(1-\phi)p}$$

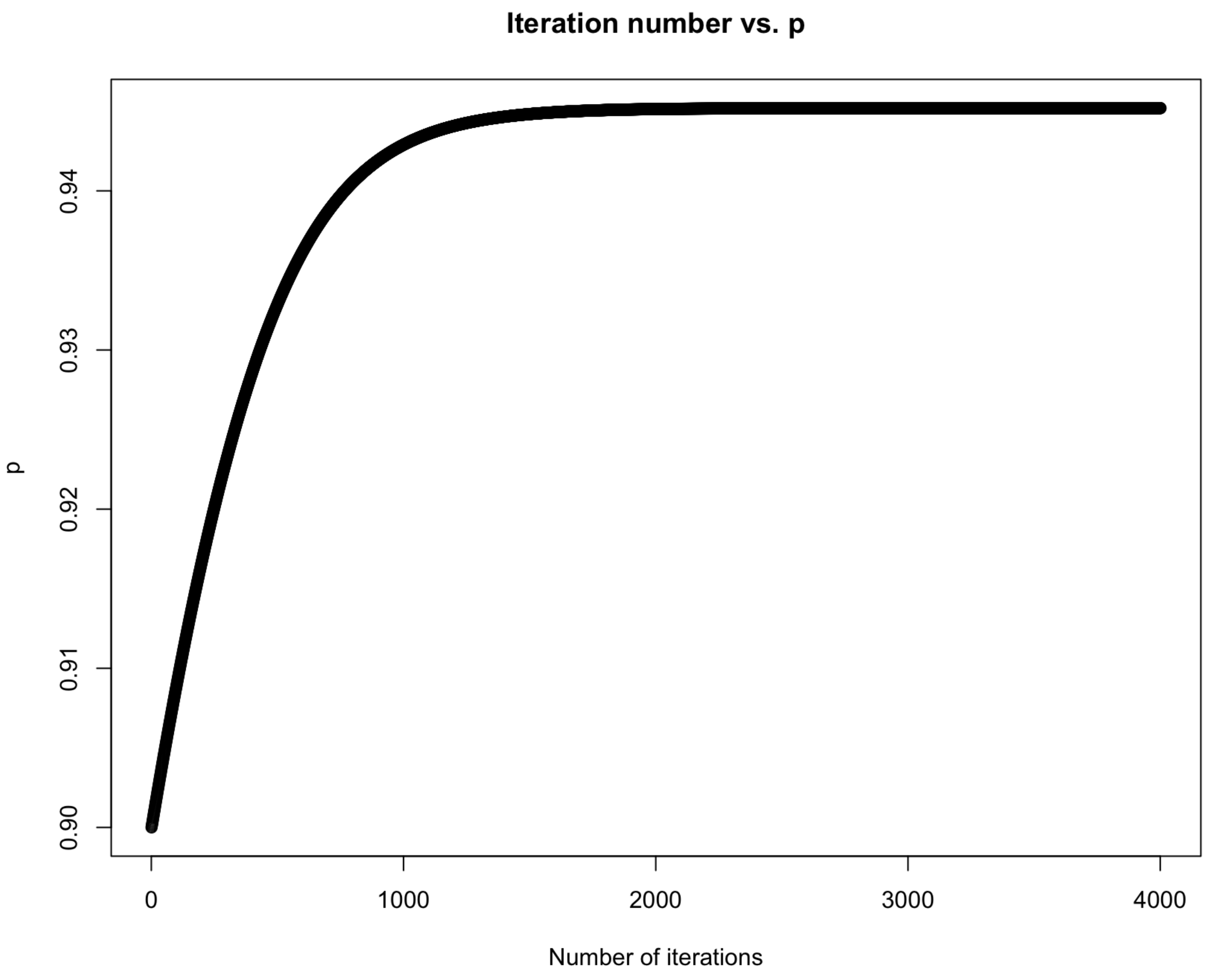
Then, we implement the NLL in R and generate a 3D plot to coarsely study its shape:

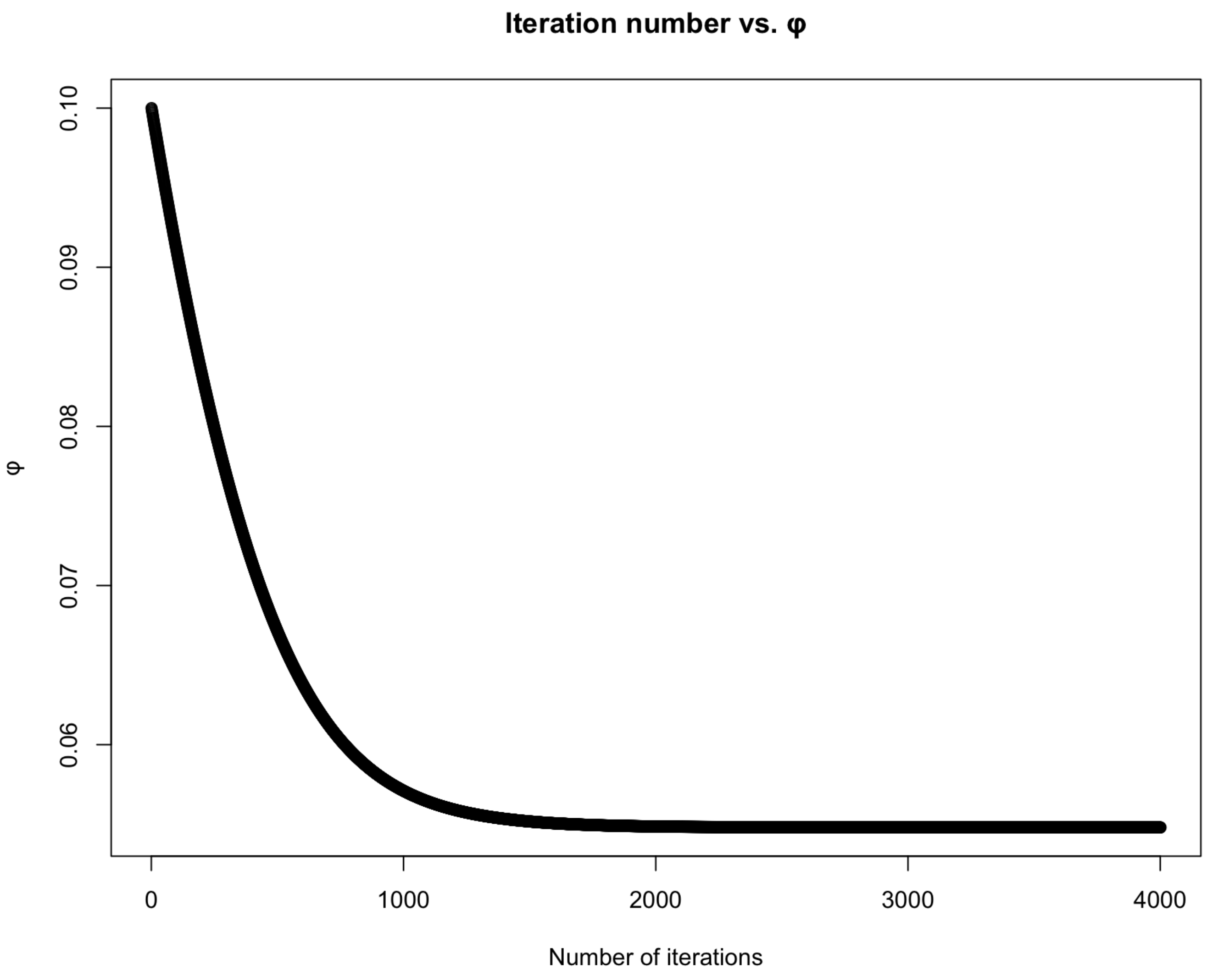


The shape seems approximately convex, and should attain a minimum (i.e., the MLE) at around $\hat{p} = 0.95, \hat{\phi} = 0.05$.

We again solve it using gradient descent with numeric gradient. The algorithm converges to around $\hat{p} = 0.94519889, \hat{\phi} = 0.05480111$ within 4000 iterations as the MLE with an initial value $(p_0, \phi_0) = (0.9, 0.1)$ and step size 0.000001.

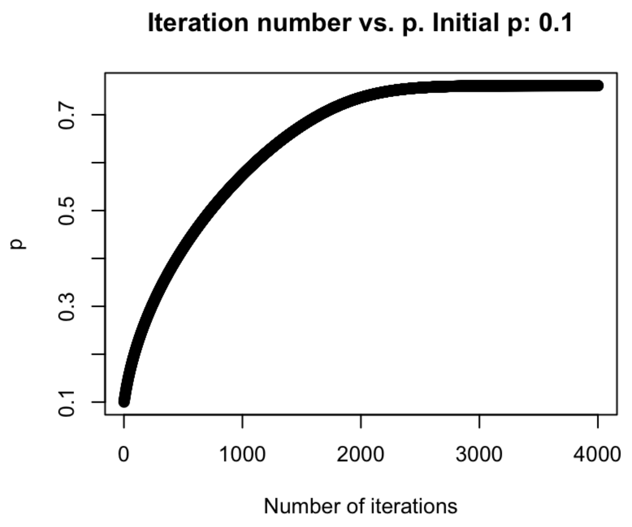
The convergence plots are:

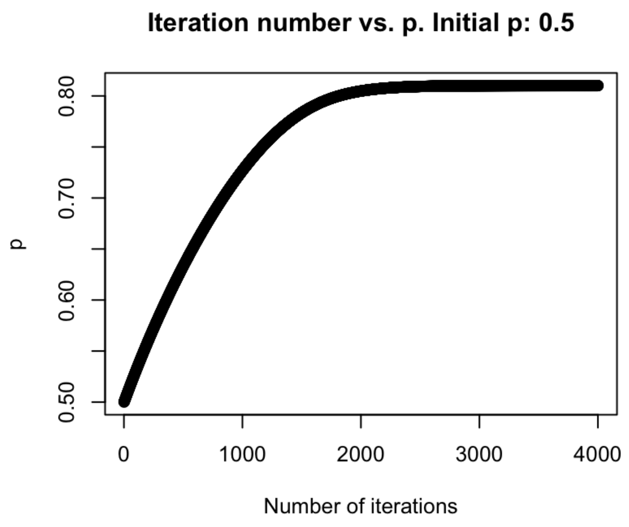

Iteration number vs. p
p
0.90
0.91
0.92
0.93
0.94
0
1000
2000
3000
4000
Number of iterations

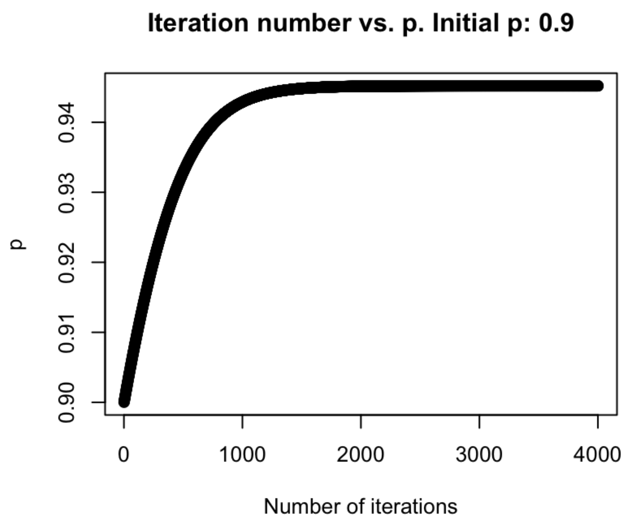


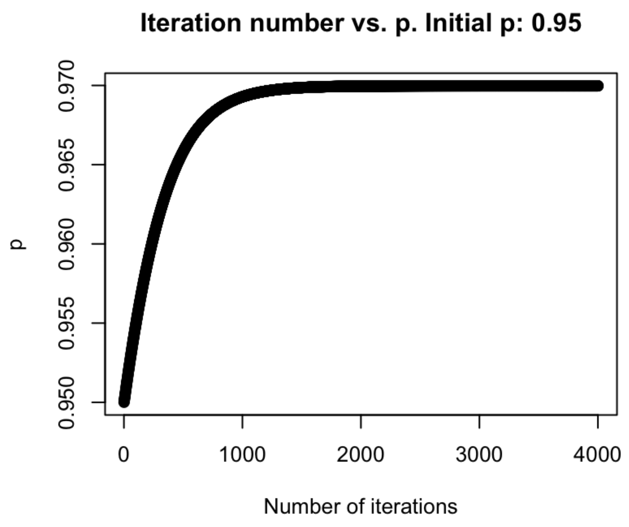
This time, the two parameters have very similar convergence speed.

We then test the sensitivity of their convergence w.r.t. the initial values. For p, we hold the initial value of $\phi_0 = 0.1$ as constant and keep everything else unchanged. Then, by trying different initial values for p, we have such a convergence plot matrix for comparison:





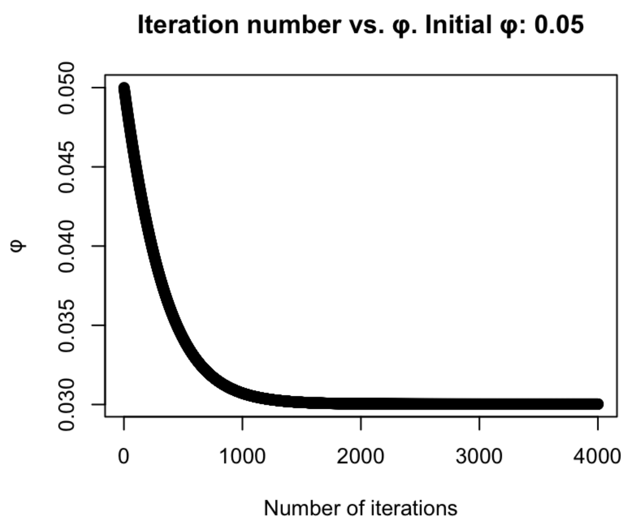


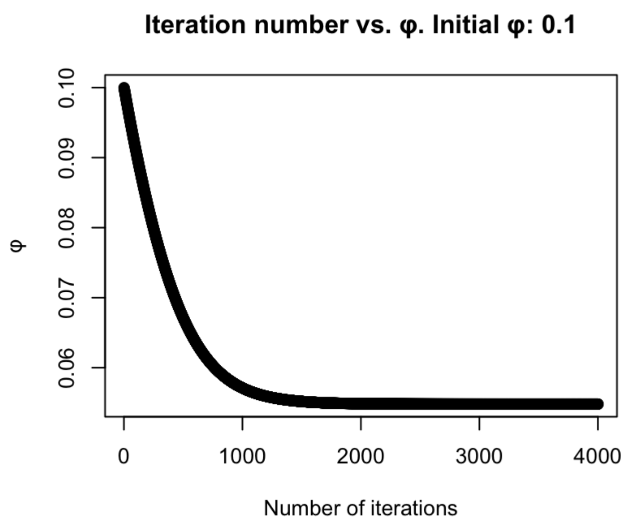


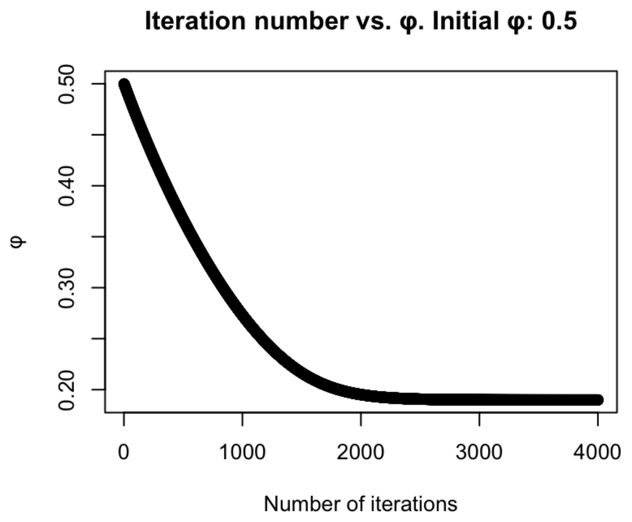
The convergence of p is not robust. Even though they all converge, they converge to different values. The MLE they converge to are: (0.7608729, -0.1741558), (0.8100985, -0.1028250), (0.94519889, 0.05480111), (0.96997284, 0.07894225) respectively in order. The first 2 are invalid, since a negative $\phi$ value is outside the parameter space. It means that the NLL is not convex, or at least not strictly convex. That being said, it's either that the NLL has multiple local minimizers, so the algorithm would be trapped in the nearest local minimizer (i.e., NLL is nonconvex), or the NLL has multiple global minimizers. If we test their NLL values, they are all 66.85098, which means that the NLL might have multiple global minimizers (not necessarily though, since we don't "prove by exmaples"). The MLE failed to be identified with the given data here. They are very sensitive to the initial value. Very close initial values can yield very different MLEs.

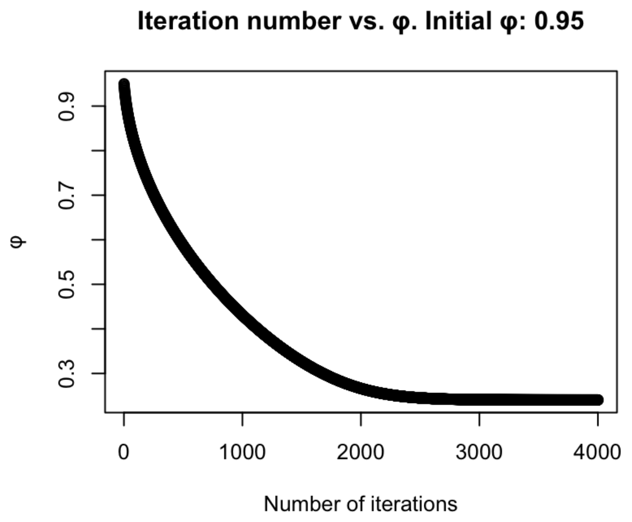
For the initial value that is closer to the MLE, it saves about 1000 iterations compared with the one that is farthest to the MLE (about 2000 iterations vs. 3000 iterations). But no matter which initial value you pick, it converges to the same MLE.

For $\phi$ , we hold the initial value of $p_0 = 0.9$ as constant and keep everything else unchanged. Then, by trying different initial values for $\phi$ , we have such a convergence plot matrix for comparison:









Still, the executions for different inital values don't converge to the same MLE and are sensitive to the initial value. The convergences are: (0.92105775, 0.03002716), (0.94519889, 0.05480111), (1.1028250, 0.1899015), (1.1763684, 0.2405589) in order respectively. The latter two are even invalid since their values of p are larger than 1, which is outside the parameter space. Regardless of the invalid MLEs, the other 2 have the same NLL values, which means they are not successfully identified by the observation data here.

They all take less than 3000 iterations to converge, which is faster than the convergence of $\hat{p}$ . The convergence is faster when the initial value is close to the MLE. For the closest one, it saves around 1000 iterations.

The MLE results are not reliable in many aspects. First, similar initial values converge to very different MLEs, and the MLEs share the same likelihood. Then, back to the context of the modeling, $\phi$ should be around 0.1 as we discussed in question b, which represents the proportion of villages that eliminate ring worms. Nonetheless, none of our results here show a $\hat{\phi}$ around 0.1.

The lack of reliable results is expected, since we can hardly distinguish which zero-infection village is from $\phi$ or p, for any given data. To be more specific, what we have no source of information to infer if a village has no infection is because worms are eliminated or it just happens to have no infections. A binomial model can estimate the part of "zero-infection villages realized by the binomial model" by using other positive-infection village statistics. Then, by subtracting the part, $\phi$ is distinguished. But now, the mixed use of two Bernoulli models make the parameters non-identifiable, so the same dataset can give different MLEs (i.e., different parameters share the same likelihood given any observed data). Therefore, the MLE obtained is not reliable, not because of the algorithm itself, but because of the characteristics of the model setting.