R Programming: Worksheet 7

By the end of today you should feel comfortable working with numerical optimization methods in R:

optim(), nlm(), uniroot()

1. Maximum Likelihood

Suppose we have non-negative integer valued data \(Y_1, \ldots, Y_n\), and an observed covariate \(x_1, \ldots, x_n\). In a Poisson valued generalized linear model, we assume that

\[ Y_i \sim \text{Poisson}(\lambda_i) \]

independently, where

\[ \log \lambda_i = \beta_0 + \beta_1 x_i. \]

(a) Write down the log-likelihood for \(\beta = (\beta_0, \beta_1)^T\) given these data. \(Y_i \sim \text{Poisson}(e^{\beta_0 + \beta_1 x_i})\), so

\[ l(\beta_0, \beta_1) = \sum_{i=1}^{n} \left\{ y_i(\beta_0 + \beta_1 x_i) - e^{\beta_0 + \beta_1 x_i} \right\} \]

(b) Write a function of three numeric vector arguments: beta, y, x, which returns minus the log-likelihood for the above model evaluated at \(\beta = (\beta_0, \beta_1)\). [Don’t use dpois() for this.] We can ignore the constant term, so:

\[
\text{minusLogLik} <- \text{function}(\text{beta, y, x}) - \text{sum}(y * (\text{beta}[1] + \text{beta}[2] * x) - \text{exp}(\text{beta}[1] + \text{beta}[2] * x))
\]

You can [optionally] check this using dpois() with the option log=TRUE, but you’ll need to add the constant term I ignored:

\[
\text{> degenerate case with beta1 = 0}
\text{> y <- rpois(100, lambda = exp(2))}
\text{> sum(dpois(y, lambda = exp(2), log = TRUE) + lfactorial(y))}
\]

\[
\#
\#
## [1] 605.1
\text{> minusLogLik(c(2, 0), y, 0)}
\#
## [1] -605.1
\]

(c) Let \(n = 100\). Generate a covariate vector \(x\) as independent standard normal random variables. Use this to generate data from the above model with \(\beta_0 = 1, \beta_1 = 1/2\).

\[
\text{> n <- 100}
\text{> x <- rnorm(n)}
\text{> y <- rpois(n, lambda = exp(1 + x/2))}
\]
The R function `optim()` performs generic minimization of functions. Its arguments are `par`, a vector of starting parameters (so in this case some starting value for $\beta$), and `fn`, a function with first argument to be minimized over.

(d) Use the function `optim()` to find the MLE for your dataset.

```R
> optim(c(1, 0), minusLogLik, y = y, x = x)
```

```
$par
[1] 1.0255 0.4337

$value
[1] -59.61

$counts
function gradient
59 NA

$convergence
[1] 0

$message
NULL
```

(e) Check your answer by running the command

```R
> glm(y ~ x, family = poisson)
```

```
Call: glm(formula = y ~ x, family = poisson)

Coefficients:
(Intercept) x
1.026 0.434

Degrees of Freedom: 99 Total (i.e. Null); 98 Residual
Null Deviance: 160
Residual Deviance: 106 AIC: 380
```

[If you haven’t seen GLMs before, you will do soon.]

(f) Try doing the same thing as in (d) but using the function `nlm()` (which is similar to `optim()`). `nlm()` has similar syntax, but doesn’t have as many options, and the first two arguments are the other way around.

```R
> nlm(minusLogLik, c(1, 0), y = y, x = x)
```

(g) * Give the output of `optim()` from (d) the class `optimum`. Write a print method for objects of this class which neatly displays (i) the optimal parameters, (ii) the value of the function at the optimum, and (iii) a suitable explanation of the error code (see ?optim).
2. Estimating Equations

Consider a time series model

\[ X_t = \phi X_{t-1} + \phi^2 X_{t-2} + \epsilon_t, \quad t = 2, \ldots, T \]

where \( X_0 = X_1 = 0, |\phi| < \frac{1}{2} \), and \( \epsilon_t \) are independent, identically distributed random variables with mean 0 and finite variance.

(a) Write a function with arguments \( T \) and \( \phi \), which generates a time series of the form above; have the errors be \( t_5 \)-distributed. This one is hard to do without using a loop:

```r
> genTime = function(T, phi) {
+   out = numeric(T + 1)
+   for (i in 2:T) out[i + 1] = phi * out[i] + phi^2 * out[i - 1] + rt(1, df = 5)
+   out
+ }
```

(b) Generate some data with your function, using \( \phi = 0.4 \) and \( T = 100 \).

```r
> y = genTime(100, 0.4)
```

(c) Let

\[
g(\eta, X) = \frac{1}{T} \sum_{t=2}^{T} X_{t-1} (X_t - \eta X_{t-1} - \eta^2 X_{t-2})
\]

Prove that

\[
E X_t X_{t-1} = \phi E X_{t-1}^2 + \phi^2 E X_{t-1} X_{t-2}
\]

and deduce that \( E g(\phi, X) = 0 \). Just use the definition and the fact that \( E \epsilon_t = 0 \) and is independent of \( X_{t-1} \).

If we don’t know the distribution of the \( \epsilon_t \)s (let’s pretend we don’t), we can’t write down a likelihood for \( \phi \). However, we can find a root of the equation \( g \): that is choose \( \hat{\phi}_T \) such that \( g(\hat{\phi}_T, X) = 0 \). This is called the method of estimating equations. Under reasonable conditions on the choice of \( g \) we find that \( \hat{\phi}_T \to \phi \) as \( T \) grows.

(d) Write an R version of the function \( g \), with first argument \( phi \), and second \( y \).

```r
> g <- function(phi, y) {
+   n = length(y)
+   mean(y[-(1:2)] * y[-c(1, n)] - phi * y[-c(1, n)]^2 -
+     phi^2 * y[-c(1, n)] * y[-c(n - 1, n)])
+ }
```

(e) Now solve the estimating equation: that is, find \( \hat{\phi} \) such that \( g(\hat{\phi}) = 0 \). Use the function `uniroot()` . We just have to pass it our function, together with a sensible interval to search in. Since we know that \( \phi < \frac{1}{2} \), we might try between 0 and 0.5 to start with.
> out <- uniroot(g, interval = c(0, 0.5), y = y)
> out$root
## [1] 0.3641

Answers will vary slightly, but should be approximately 0.4, which is the true value.

(f) Observe that \( \hat{\phi} \) is just the solution to a quadratic equation, and write a function to solve it exactly. [But note that it would be easy to construct an example without such an exact solution.] Using the quadratic formula in the form \( a\phi^2 + b\phi + c = 0 \), gives:

> quadSolve = function(y) {
+   n = length(y)
+   coef_a = mean(y[-c(1, n)] * y[-c(n - 1, n)])
+   coef_b = mean(y[-c(1, n)]^2)
+   coef_c = -mean(y[-(1:2)] * y[-c(1, n)])
+   phi = (-coef_b + sqrt(coef_b^2 - 4 * coef_a * coef_c))/(2 *
+     coef_a)
+   phi
+ }

It is clear from the data generating mechanism that the positive root is the appropriate one.

(g) Generate a single large data set (sample size \( n = 10^4 \)) and use the function from the previous part to find solutions to the estimating equation using the first 100, 300, 1000, 3000, and \( 10^4 \) observations.
Repeat this a large number of times (say 100), and comment on the accuracy of the estimates (of course the estimates improve, but how quickly?)

> set.seed(242)
> sq <- c(100, 300, 1000, 3000, 10000) # sample sizes
> out <- replicate(100, {
+   Y <- genTime(10000, 0.4)
+   sapply(sq, function(i) quadSolve(Y[seq(i)]))
+ })
> apply(out, 1, sd) * sqrt(sq) # similar values
## [1] 0.6323 0.6270 0.6546 0.6387 0.6692

Like most parametric statistical estimators, the standard error improves in proportion to \( \sqrt{n} \). [Note that you can do this much quicker if you change \texttt{genTime()} to generate multiple time series as the columns of a matrix, and use \texttt{apply()} methods.]
3. *Violation of Modelling Assumptions*

(a) Write a function which takes a single integer `n`, and returns a list with entries `x` and `y`, where `x` is a vector of `n` independent uniform random variables on `[-1, 1]`,

\[ y_i = x_i^2 + \varepsilon_i, \quad i = 1, \ldots, n, \]

and \( \varepsilon_i \sim i.i.d. N(0, 1) \).

```r
> gendata = function(n, df = 3) {
+   x = runif(n, -1, 1)
+   y = x^2 + rnorm(n)
+   return(list(x = x, y = y))
+ }
```

(b) Generate a sample of size 1,000 using the function from (a), and fit a linear model using the command `lm1 = lm(y ~ x)`. Look at the summary of your model output, as well as the diagnostic plots with `plot(lm1)`. What do you notice? *The plot of fitted values against residuals appears to have a trend in it.*

(c) Write a second function which generates `x` as before, but

\[ y_i = x_i + \varepsilon_i, \quad i = 1, \ldots, n, \]

where \( \varepsilon_i \sim i.i.d. \sim t_3 \). (Use the `rt()` function.)

```r
> gendata2 = function(n, df = 3) {
+   x = runif(n, -1, 1)
+   y = x + rt(n, df)
+   return(list(x = x, y = y))
+ }
```

(d) Repeat (b) with your new function. *Mostly looks OK, but QQ-plot seems heavy tailed.*

(e) Write a function with argument `n` which generates a sample using the function from (c), fits a linear model, and then reports a 95% confidence interval for the coefficient of `x` (the slope).

```r
> getCI = function(n, df = 3) {
+   dat = gendata2(n, df = df)
+   lm1 = lm(y ~ x, data = dat)
+   confint(lm1)[2,]
+ }
```

(f) For a sample size `n = 10`, use the function from the previous part to generate `N = 1,000` confidence intervals for different data sets. How many of them contain the ‘true’ value of the slope? *Surprisingly good even for small sample sizes.*

(g) Try increasing the sample size and repeating the previous part. *For moderate sample sizes, we still seem to get decent coverage. This is because the t-distribution based confidence interval is asymptotically valid as long as there is a consistent estimate of the standard deviation, and the sample mean is normally distributed.*