STA250, Homework 3
Christopher Aden
November 23, 2013

1 Basic Optimization:

Bisection Method

Coding the bisection algorithm is incredibly simple (algorithm description taken from Rex Cheung's lecture notes):

1. Find l and u such that $g(l)g(u) < 0$. By the Intermediate Value Theorem, there exists a root between l and u. Set iteration counter, $t = 0$.
2. If t is equal or greater than max number of iterations, stop and return a null value. Declare non-convergence.
3. Set $c = \frac{l+u}{2}$, compute $g(c)$.
4. If $|g(c)| < \epsilon$ for some small $\epsilon > 0$, convergence is reached–return c.
5. Otherwise, if $g(l)g(c) < 0$, set $u = c$, else set $l = c$.
6. $++t$.
7. Goto 2.

The code takes in an arbitrary function g, as well as boundary points satisfying the condition outlined in step one of the algorithm. These can be found through guess and check or plugging in the boundaries of the support of the function (not always possible, if the function is supported on the whole real line). It also takes a tolerance ϵ, a maximum number of iterations before stopping and declaring non-convergence, and a debug flag that when enabled, prints values useful in debugging the function.

Newton-Raphson

Coding the Newton-Raphson algorithm is a little less simple, but still pretty fast (algorithm description taken from Rex Cheung's lecture notes):

- Pick x_0, set $t = 0$.
- If t is equal to the max number of iterations, stop and return a null value, declaring non-convergence.
- Update $x_{t+1} = x_t - \frac{g(x_t)}{g'(x_t)}$.

- If $|g(x_{t+1})| < \epsilon$, stop. Else, $++t$.
- Goto 2.

The code takes in an arbitrary function g, its derivative g', a starting point x_0, tolerance ϵ, and the maximum number of iterations and debug flag, identical to the bisection method.

Comparison of Bisection and Newton-Raphson: Linkage Problem

We compare the results of the two optimization algorithms by testing them on a problem. Define $\lambda = 1 - 2\theta + \theta^2$, where $\theta \in (0, 1)$. We want to maximize

$$L(\lambda) \propto (2 + \lambda)^{125}(1 - \lambda)^{18+20}\lambda^{34}$$

From calculus, note that this is equivalent to find the root of the derivative of the log-likelihood.

$$log(L)'(\lambda) \propto \frac{\partial}{\partial\lambda}\left(125log(2 + \lambda) + 38log(1 - \lambda) + 34log(\lambda)\right)$$

$$\Rightarrow log(L)'(\lambda) \propto \frac{125}{2 + \lambda} + \frac{38}{1 - \lambda} + \frac{34}{\lambda}$$

We want to find the root of this function (for what value of λ this derivative takes the value zero). To do bisection, we just need to find two points, (l, u), such that $log(L)'(l) \cdot log(L)'(u) < 0$. Two suggested points that worked were $l = .5$ and $u = .8$. Bisection reached convergence (settling on $\lambda = 0.6268215$ after 22 steps).

Newton-Raphson required a second derivative of the log-likelihood. The second derivative is given by

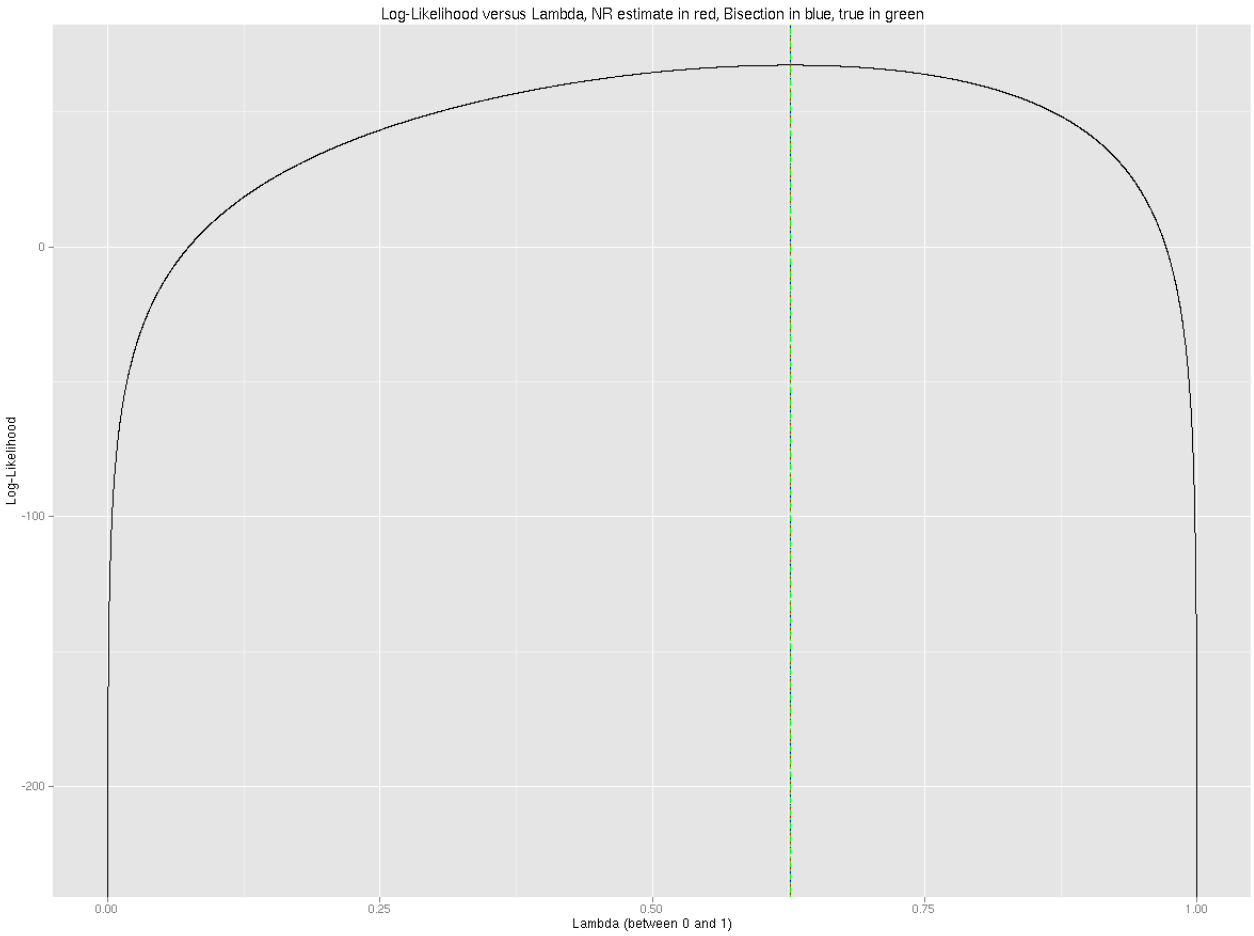
$$log(L)''(\lambda) \propto \frac{-125}{(2 + \lambda)^2} - \frac{38}{(1 - \lambda)^2} - \frac{34}{\lambda^2}.$$

Using the starting value of $x_0 = .8$ and tolerance 10^{-5}, Newton-Raphson also converged (5 iterations, settling on $\lambda = 0.6268215$.

It should be clear that Newton-Raphson drastically outperformed the Bisection method in terms of number of iterations, but the computation involved in each step was more than bisection, so the comparisons is apples-to-oranges. Using the rbenchmark package in R, We run 10,000 replications of the optimization. The results indicated that bisection took more than twice as long as Newton-Raphson, confirming our belief that Newton-Raphson is a faster algorithm, despite the greater computational cost in each iteration. We may have deduced this from the asymptotic convergence rates, but run time checks aren't a bad idea.

	test	replications	elapsed	relative	user.self	sys.self	user.child	sys.child
1	bisection	10000	3.55	2.28	3.58	0.08	0.00	0.00
2	Newton-Raphson	10000	1.56	1.00	1.51	0.10	0.00	0.00

Let's take a look at the likelihood function to see how accurate they were.



Location of true MLE, Newton-Raphson optimizer and Bisection optimizer, plotted on log-likelihood

Both methods managed to hit the true MLE (determined through the woefully inefficient grid search, for completeness) dead on. This demonstrates that finding the root of the derivative is an effective method for finding the extrema of a function iteratively when the extremum cannot be easily determined.

R Source Code

Code to compute the Newton-Raphson and Bisection Algorithms, along with the code to produce the GGPlot and benchmarking.

```r
library(ggplot2); library(rbenchmark)

bisection = function(f, l, u, tol=1E-6, niter=100, debug=FALSE){
  x = numeric(niter)
  for(t in 1:niter){
    x[t] = {l + u} / 2 #Set the value of c.

    if( abs(f(x[t])) < tol ) { #Bisection method reached convergence. Return val.
      print(paste("Converged in ", t, " steps.", sep=''))
      return(list(final=x[t], iterations=t, xvalues=x[1:t], fvalues=f(x[1:t])))
    }

    if ( f(x[t]) * f(l) < 0 ) { #If f(c) * f(l) < 0, u=c. Else, l=c.
      u=x[t]
    } else { l=x[t] }

    if (debug){ #Print pieces of Bisection calculation
      print(paste("(x_t, f(x_t), f(u), f(l), u, l ): (",
            paste(round(c(x[t], f(x[t]), f(u), f(l), u, l), 5), collapse=", ", sep=""), ")", sep=''))}
  }
  return(NULL) #Did not converge!
}

newt.raph = function(f, df, initial=0, tol=1E-6, niter=100, debug=FALSE){
  x = numeric(niter+1)
  x[1] = initial
  for(t in 1:niter){
    x[t+1] = x[t] - f(x[t]) / df(x[t]) #Heart of the NR Algorithm

    if (debug){ #Print pieces of NR calculation.
      print(paste("(x_t, f(x_t), f'(x_t), x_{t+1}): (",
        paste(round(c(x[t], f(x[t]), df(x[t]), x[t+1]), 6), collapse=", ", sep=""), ")", sep=''))}

    if( abs(f(x[t+1])) < tol ) { #NR reached convergence. Return val.
      print(paste("Converged in ", t, " steps.", sep=''))
      return(list(final=x[t+1], iterations=t, xvalues=x[1:(t+1)], fvalues = f(x[1:(t+1)])))
    }
  }
  return(NULL) #Did not converge!
}

#Define log-likelihood and derivatives.
log.l = function(lambda) {125 * log(2+lambda) + 38 * log(1-lambda) + 34 * log(lambda) }
dlog.l = function(lambda) { 125 / (2+lambda) - 38 / (1-lambda) + 34 / lambda}
ddlog.l = function(lambda) {-125 / (2+lambda)^2 -38 / (1-lambda)^2 - 34 / lambda^2}

MLE.bisection = bisection(f=dlog.l, l=.5, u=.8, tol=1E-5, niter=100, debug=FALSE)
system.time(replicate(n=1000, bisection(f=dlog.l, l=.5, u=.8, tol=1E-5, niter=100, debug=FALSE)) )

benchmark(bisection(f=dlog.l, l=.5, u=.8, tol=1E-5, niter=100, debug=FALSE),
          newt.raph(f=dlog.l, df=ddlog.l, initial=.8, tol=1E-5, niter=100, debug=FALSE), replications=1E4)

x = seq(0, 1, by=.0001)
likelihoods = log.l(x) #By grid search, for the truth.
loglike.x = data.frame(likelihoods, x)
ggplot(loglike.x, aes(x=x, y=likelihoods)) + geom_line() +
  geom_vline(xintercept = MLE.bisection$final, color="blue", linetype="longdash") +
  geom_vline(xintercept = MLE.NR$final, color="red", linetype="dashed") +
  geom_vline(xintercept = x[which.max(likelihoods)], color="green", linetype="twodash") +
  ggtitle("Log-Likelihood versus Lambda, NR estimate in red, Bisection in blue, true in green") +
  xlab("Lambda (between 0 and 1)")  + ylab("Log-Likelihood")
```