Week 2: Homework Assignment

Patrick Kelly

Saturday, January 24, 2015

Code Your Own Optimizer in R

This project helps understanding how non-linear optimization of log-likelihood function by the Newton-Raphson method works

Use the formulas for the steps of Newton-Raphson method from the lecture notes to code a simple version of optimizer.

Assignment description

1. Create a test function that needs to be optimized.

For this project we use one-dimensional optimization, i.e. optimization with respect to only one variable.

Let the declaration of the function be `my.Function<-function(my.X)`, where `my.X` is a scalar parameter with respect to which the optimization is done.

The function should cross the x-axis at least in one point.

For example, you can use

```
my.Function<-function(my.X) {
  my.X^2*3-my.X*5-5
}
```

2. Write your optimizer.

Let the declaration of the optimizer function be `my.Optimizer<-function(Start.Value,Function.To.Optimize,Epsilon)` where `Start.Value` is the initial guess for the optimizer, `Function.To.Optimize` is the name of your test function that needs to be optimized, `Epsilon` is the stopping criterion, a small number, for example, 0.001.

The function `my.Optimizer` should repeat iterations of the Newton-Raphson algorithm while |x(i+1) - x(i)|>= Epsilon, where x(i+1) is the approximation obtained during the recent iteration and x(i) is the approximation obtained during the previous operation. You can use any of the loops in R, for example, `while(cond) expr` where `cond` is the condition of moving to the next iteration (|x(i+1) - x(i)|>=??) and `expr` is the sequence of the commands that need to be performed at each iteration.

```r
my.Optimizer<-function(Start.Value,Function.To.Optimize,Epsilon){

  #initialize check.done so that it does not satisfy while condition below
  check.done <- 1;

  #initialize 'old.value' parameter
  old.value <- Start.Value;

  #initialize the counter that determines the number of times the while loop executes
  iter <- 0

  #set an 'h' value for the derivative estimation
  h <- .00000001

  while(check.done > Epsilon){

    new.value <- old.value - (Function.To.Optimize(old.value)/
                             ((Function.To.Optimize(old.value+h)-Function.To.Optimize(old.value))/(h)))

    check.done <- abs(new.value - old.value)

    old.value <- new.value

    iter <- iter+1

  }

  return(old.value)

}
```

3. Test the optimizer

Use your optimizer with the test function. For example, use `my.Optimizer(-5,my.Function,.001)`. Make sure you calculate the answer manually to check the answer.

You can also test the optimizer by running `uniroot()`. For example, `uniroot(my.Function,lower=-5,upper=+1)`

The root returned by your optimizer should be the same as the output `$root` of the object returned by `uniroot()`.

```
#first obtain the output from the optimizer that I created
my.Optimizer(-5,my.Function,.0001)
```

```
## [1] -0.7032574
```

```
#now check this against the output out uniroot()
uniroot(my.Function,lower=-5,upper=1)
```

```
## $root
## [1] -0.703257
##
## $f.root
## [1] -3.368345e-06
##
## $iter
## [1] 9
##
## $estim.prec
## [1] 6.103516e-05
```

As you can observe from the output comparisons above, my optimization function `uniroot()` produce the same results!

Try also to run `optim()`. Explain the difference between the two functions: `uniroot()` and `optim()`.

```
optim(-5,
      fn = my.Function,
      method="L-BFGS-B",
      hessian=TRUE,
      lower=c(-Inf,0))
```

```
## $par
## [1] 0.8333333
```

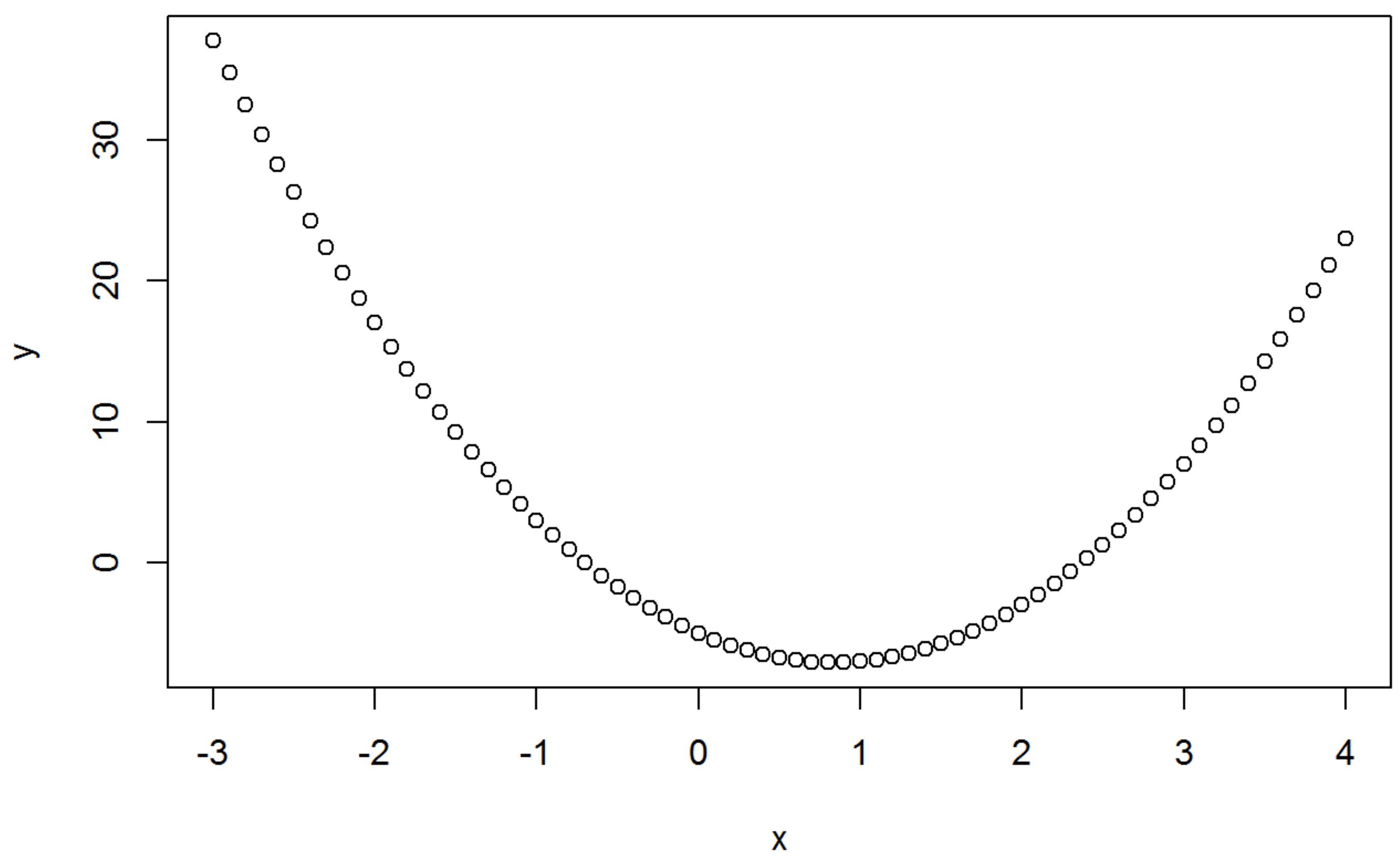
```
##
## $value
## [1] -7.083333
##
## $counts
## function gradient
##       10       10
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
##
## $hessian
##      [,1]
## [1,]    6
```

How can we reconcile their outputs?

As we see above, the `optim()` output is different than the output produced by `my.Optimizer()` and `uniroot()`.

If you observe the plot of the original function (below) you notice that, as expected, my `my.Optimizer()` and `uniroot()` identify a root of the function, while `optim()` identifies the minimum of the function (0.8333).

```
#show a plot of the function
x <- seq(from=-3,to=4,by=.1)
y <- my.Function(x)
plot(x,y)
```



However, our different prodcedures above are related in that if you are optimizing (finding the minimum or maximum) a function then you can either use `optim()` to accomplish this OR take the deravative of the function and use either `my.Optimizer` or `uniroot()` to find the root of the derivation which is equal to the max / min of the original function.

Similarly, if you wanted to find the roots of a function you can either use `my.Optimizer` or `uniroot()` to accomplish this or take intergral of the function and use `optim()` to find min / max points of the intergral which is equal to the roots of the orginal function.

These findings align with our lecture and learnins in class.