Homework 4 [45 points]

Questions related to the homework can be posted on the Piazza forum site, but do not post answers to homework problems. We will try to respond promptly to posted questions. However, please do not send post questions after 9pm the day before the assignment is due.

You may discuss the homework problems and computing issues with other students in the class. However, you must write up your homework solution on your own. In particular, do not share your code or homework files with other students. If you collaborate with other students list their names at the beginning of your writeup.

This homework will build upon the material presented in lectures, and will be focused on root finding and confidence intervals. All programming should be done in `R`. If there are any difficulties starting the homework, please contact the TA.

The homework is split into two parts:

Part 1: Analysis of Root Finding Methods

Part 2: Construction of Confidence Intervals via Root Finding

For Part One, text in blue denotes what needs to be done.

Files to submit on CMS

Do check to see that you have submitted all these files. Each part will be graded as a whole. The writeups should either be TeXed up or done using R Markdown.

1. Part One [25 points]
    - `halley1.r`
    - `FindDeriv.r`
    - `writeup_one.html` or `writeup_one.pdf`
2. Part Two [20 points]
    - `LR_interval.r`
    - `Exact_interval.r`
    - `writeup_two.html` or `writeup_two.pdf`

Part One: The Analysis Of Root Finding Methods (25 points)

Given a function $f(x)$, root finding methods are used to find a value (or values) x such that:

$$f(x) = 0$$

In high school, we learned how to find the roots of a polynomial of degree 2 by completing the square[1]. However, what if we wanted a way to find the roots for a more generic $f(x)$? We might be stuck, as Galois Theory tells us there's no general formulae for polynomials of degree 5 or more. If $f(x)$ includes trigonometric functions, logarithms, exponentials, and other mathematical functions, finding a general formula might be impossible.

Root finding methods like the Newton Raphson method (covered in lecture) come into play here. They (usually) give us an approximate solution[2] x where $f(x) = 0$. However, we can't just apply these root finding methods blindly "out of the box" since there are some questions (not exhaustive) to take into consideration.

- Does my root finding method work when $f(x)$ is not differentiable? Not continuous?
- How fast does it take my root finding method to converge to a root?
- Does my initial starting value affect whether my root finding method converges to a root? The particular root I want? Diverges?
- How do I know that there aren't any more roots?

In this part, we will look at the *construction* of root finding functions, and try to answer some of these above questions.

Newton Raphson Algorithm

In the textbook (Jones et al), the Newton Raphson code is given. We reproduce it here for convenience. We *strongly suggest* playing about with this code and looking at example inputs / outputs to see what this function does, before continuing on.

[1] Note: The quadratic equation is derived from completing the square.

[2] Exact solutions in textbook cases.

```
1.   newtonraphson <- function(ftn, x0, tol = 1e-9, max.iter = 100) {
2.
3.     # Newton_Raphson algorithm for solving ftn(x)[1] == 0
4.     # we assume that ftn is a function of a single variable that returns
5.     # the function value and the first derivative as a vector of length 2
6.     #
7.     # x0 is the initial guess at the root
8.     # the algorithm terminates when the function value is within distance
9.     # tol of 0, or the number of iterations exceeds max.iter
10.
11.     # initialise
12.     x <- x0
13.     fx <- ftn(x)
14.     iter <-  0
15.     # continue iterating until stopping conditions are met
16.
17.     while ((abs(fx[1]) > tol) && (iter < max.iter)) {
18.       x <- x - fx[1]/fx[2]
19.       fx <- ftn(x)
20.       iter <- iter + 1
21.       cat("At iteration", iter, "value of x is:", x, "\n")
22.     }
23.     # output depends on success of algorithm
24.
25.     if (abs(fx[1]) > tol) {
26.       cat("Algorithm failed to converge\n")
27.       return(NULL)
28.     } else {
29.       cat("Algorithm converged\n")
30.       return(x)
31.     }
32.   }
```

We are going to use the Newton Raphson Algorithm as a baseline when looking at another root finding method: *Halley's method*.

Modify lines 29-30 of the function `newtonraphson` to give the output x together with the number of iterations needed for the algorithm to converge. You may add additional lines of code. The output can be in any format you want. Call this function `newtonraphson1`. Put this code in the writeup.

Halley's Method

This excerpt is taken from the textbook. In addition to the first derivative, Halley's method requires second derivative information as well. Thus, if x_n is our current solution then:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n) - (f(x_n)f''(x_n)/2f'(x_n))}$$

Write a function `halley1` that takes in *four* inputs:

```
halley1<-function(ftn, x0, tol, max.iter){

  # ftn - a function of a single variable that returns
  # the function value, the first derivative, and the
  # second derivative as a vector of length 3.

  # x0 is the initial guess at the root

  # the algorithm terminates when the function value is within distance
  # tol of 0, or the number of iterations exceeds max.iter


  return(c(root_x, num_iter))
}
```

The output is a vector of length 2. If the algorithm does not converge, then `root_x` should be `NA`, and `num_iter` = -1. Else, `root_x` should be the approximate root x returned by the algorithm, and `num_iter` the number of iterations it takes for convergence.

Submit `halley1.r` to CMS. Put a copy of this function in the writeup as well.

Analytically Computing The Derivatives

In both `newtonraphson1` and `halley1`, we had an input `ftn` which was a function of a single variable which returns the function value, the first derivative, and the second derivative (for `halley1`).

However, this means we would have to write a new function for every $f(x)$ we have. For example, if we wanted to find the root of $f(x) = x^2$ using `halley1`, we would have to write:

```
ftnxsquare<-function(x){
  fx = x^2
  dfx = 2*x
  ddfx = 2
  return(c(fx,dfx,ddfx))
}
```

and then run:

```
results = halley1(ftnxsquare, 5, 1e-6, 100)
```

Let's see if we can simplify this process by resorting to calculus. Recall that we have the definition of:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Using this idea, write a function `FindDeriv` of the following form:

```
FindDeriv<-function(ftn, x){

  # a ftn of a single variable f(x), which takes in x
  # and returns f(x)

  # Put a value of epsilon here which would work with most
  # functions

  return(c(fx,dfx,ddfx))
}
```

The function should return a vector of `ftn`'s output at x, the first derivative at x, and second derivative at x.

Since taking the derivatives require an epsilon, write the function `FindDeriv` such that it incorporates ϵ which outputs the correct derivatives.

Hint: Suppose you have a function:

```
ftnxsquare<-function(x){
  return(x^2)
}
```

You should get the right derivatives (or something extremely close to them) with the following inputs.

```
FindDeriv(ftnxsquare,1)
FindDeriv(ftnxsquare,10000)
```

Submit `FindDeriv.r` to CMS, and put a copy of the function in the writeup. Explain how you have computed ϵ in your code.

The Comparison

Write `newtonraphson2` and `halley2` where the respective derivatives are analytically computed. Put a copy of both functions in the writeup.

We will now compare how these *four* methods do in root finding.

For *each* question, report how fast each method takes to converge (number of iterations), report the root it converges to, report the true root, *for all of the given starting values of* x_0.

You are encouraged to present the answers in a format which allows easy comparison.

1. The square root of 26 with $x_0 = 1, 3, 6$
2. $\cos x - x$ with $x_0 = 1, 3, 6$
3. $2^x - 3^x + 1$ with $x_0 = -2, 0, 2$
4. $\log x - \exp\{x\}$ with $x_0 = 2, 4, 6$
5. $x^3 - 7x^2 + 14x - 8$ with $x_0 = 1.1, 1.2, \ldots, 1.9$
6. $\frac{\sin x \cos x}{\tan x + \sec x}$ with $x_0 = 1, 2, 3$

Based on the answers you get, which method would you prefer to use in general cases of root finding? Are there any methods you would prefer to use in specialized cases? Explain your answers.

Part Two: Construction Of Confidence Intervals By Root Finding (20 points)

1. Let $X_1, X_2, \ldots, X_n$ be a random sample from a Poisson distribution with mean μ. **Create a function to determine the endpoints of an approximate confidence interval for μ by inverting the likelihood-ratio test.** You will need to determine the endpoints numerically, e.g. using the Newton-Raphson method discussed in class. (Consider calling the function Score.interval from HW3 to get starting values for the endpoints.) The function should return warnings if all the data values are zero, if there are negative values, or if some of the values are not integers. Submit your function in the form:

```
LR_interval(data,alpha,tol=1e-9,max.iter=100)
{
  R code
  return(list(lower = lower, upper = upper))
}
```

   where "data" is the vector containing the sample values, and "alpha" is the desired confidence level. Upload `LR_interval.r` on CMS.

2. The file "horsekicks.csv" has three columns representing the number of deaths by horse kick in the Prussian Army, the year (1875 to 1894), and the Corps. There are numerous web sites that discuss this data; for example,

   `http://mindyourdecisions.com/blog/2013/06/21/`
   `what-do-deaths-from-horse-kicks-have-to-do-with-statistics/`

   and

   `http://blog.minitab.com/blog/quality-data-analysis-and-statistics/`
   `no-horsing-around-with-the-poisson-distribution-troops`

   Use the tapply function to determine the total number of deaths by horse kick per year.

   Put this code in the writeup.

3. Use your LR_interval function to determine an approximate 95% confidence interval for, μ, the expected number of deaths by horse kick per year assuming the counts (y in the data set) are a random sample from the Poisson distribution.

4. Note that the distribution of the total count, X, (over all n years) is Poisson with mean $n\mu$. Consider the hypothesis $H : \mu = \mu_0$, and define an "exact" α-level test of

H_0 to be one that rejects μ_0 if

$$P(X < X_{obs}; \mu_0) + \frac{1}{2}P(X = X_{obs}; \mu_0) < \frac{\alpha}{2}$$

or

$$P(X > X_{obs}; \mu_0) + \frac{1}{2}P(X = X_{obs}; \mu_0) < \frac{\alpha}{2}$$

significance level

Create a function to find a $(1-\alpha)\%$ confidence interval by inverting the "exact" test. That is, find the interval of values of μ_0 that are not rejected. (HINT: You can use the newtonraphson2 function from part 1 along with the built-in R functions ppois and dpois.) Submit your function in the form

```
Exact_interval(data,alpha,tol=1e-9,max.iter=100)
{
 R code
 return(list(lower = lower, upper = upper))
}
```

Upload `Exact_interval.r` to CMS. Put a copy of this in the writeup.

5. Tabulate the LR, Score and Exact confidence intervals.