Bios 301: Assignment 2

Danielle Nemetz

October 31, 2013

1 Question 1

20 points

A problem with the Newton-Raphson algorithm is that it needs the derivative f. If the derivative is hard to compute or does not exist, then we can use the *secant method*, which only requires that the function f is continuous.

Like the Newton-Raphson method, the **secant method** is based on a linear approximation to the function f. Suppose that f has a root at a. For this method we assume that we have *two* current guesses, x_0 and x_1, for the value of a. We will think of x_0 as an older guess and we want to replace the pair x_0, x_1 by the pair x_1, x_2, where x_2 is a new guess.

To find a good new guess x2 we first draw the straight line from $(x_0, f(x_0))$ to $(x_1, f(x_1))$, which is called a secant of the curve $y = f(x)$. Like the tangent, the secant is a linear approximation of the behavior of $y = f(x)$, in the region of the points x_0 and x_1. As the new guess we will use the x-coordinate x_2 of the point at which the secant crosses the x-axis.

The general form of the recurrence equation for the secant method is:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Notice that we no longer need to know f but in return we have to provide *two* initial points, x_0 and x_1.

Write a function that implements the secant algorithm. Validate your program by finding the root of the function $f(x) = \cos(x)x$. Compare its performance with that of the either the Newton-Raphson or the Fixed-point method – which is faster, and by how much?

Secant Algorithm:

```
Secant <- function(fn, tol, max_iter = 100, x1, x2) {
    iter <- 0
    while ((abs(x2 - x1) > tol) && (max_iter > iter)) {
        x3 <- x2 - fn(x2) * (x2 - x1)/(fn(x2) - fn(x1))
        x1 <- x2
        x2 <- x3
```

```
        iter <- iter + 1
    }
    if (abs(x2 - x1) > tol || iter >= max_iter) {
        print("Error: Unable to find root")
    } else {
        v <- c(x3, iter)
        return(v)
    }
}
# testing function:
test_fn <- function(x) {
    return(cos(x) - x)
}
Secant(test_fn, tol = 1e-09, x1 = 1, x2 = 3)

## [1] 0.7391 5.0000
```

Newton-Raphson Method:

```
newtonraphson <- function(fun, der, x = 1, tol = 1e-09, max_iter = 100) {
    x_old <- x + 1e+06
    # Keep track of number of interations
    iter <- 0
    # Loop
    while ((abs(x - x_old) > tol) && (iter < max_iter)) {
        newx <- x
        x_old <- x
        x <- x_old - (fun(x_old)/der(x_old))
        iter <- iter + 1
    }
    # Check convergence
    if (abs(x - x_old) > tol) {
        cat("Algorithm failed to converge\n")
        return(NULL)
    } else {
        cat("Algorithm converged\n")
        v <- c(x, iter)
        return(v)
    }
}
newtonraphson(test_func, function(x) -sin(x) - 1)

## Error:  object 'test_func' not found
```

The secant method has fewer constraints than Newton-Raphson Method , but mathematically requires more computation and therefore is less efficient.

2 Question 2

15 points

Import the HAART dataset (`haart.csv`) from the GitHub repository into R, and perform the following manipulations:

```
# Note: In HW repo- with homework 1 & homework 2 files grab data in homework
# via '../data/foo.csv'
```

1. Convert date columns into a usable (for analysis) format.

```
# from git use: data <- read.table('../data/haart.csv', sep=',', head=T)
data <- read.table("~/Bios301/datasets/haart.csv", sep = ",", head = T)
# Converting init.date to date format & assigning it to a new variable,
# new_init.date
new_init.date <- as.Date(data$init.date, format = "%m/%d/%y")
# Verifying class of new variable
class(new_init.date)

## [1] "Date"

# Similar operations for last.visit and date.death:
new_last.visit <- as.Date(data$last.visit, format = "%m/%d/%y")
class(new_last.visit)

## [1] "Date"

new_date.death <- as.Date(data$date.death, format = "%m/%d/%y")
class(new_date.death)

## [1] "Date"

# Adding new variables to dataset
data$new_init.date <- new_init.date
data$new_last.visit <- new_last.visit
data$new_date.death <- new_date.death
```

2. Create an indicator variable (one which takes the values 0 or 1 only) to represent death within 1 year of the initial visit.

```
# Creates an empty vector
data$death.indicator <- NULL
for (i in 1:nrow(data)) {
    if (data$death[i] == 1 & (data$new_date.death[i] - data$new_init.date[i]) <=
```

```
        365) {
        data$death.indicator[i] <- 1
    } else data$death.indicator[i] <- 0
}
head(data)
```

```
##   male age aids cd4baseline logvl weight hemoglobin    init.reg init.date
## 1    1  25    0          NA    NA     NA         NA 3TC,AZT,EFV    7/1/03
## 2    1  49    0         143    NA  58.06         11 3TC,AZT,EFV  11/23/04
## 3    1  42    1         102    NA  48.08          1 3TC,AZT,EFV   4/30/03
## 4    0  33    0         107    NA  46.00         NA 3TC,AZT,NVP   3/25/06
## 5    1  27    0          52     4     NA         NA 3TC,D4T,EFV    9/1/04
## 6    0  34    0         157    NA  54.89         NA 3TC,AZT,NVP   12/2/03
##   last.visit death date.death new_init.date new_last.visit new_date.death
## 1    2/26/07     0       <NA>    2003-07-01     2007-02-26           <NA>
## 2    2/22/08     0       <NA>    2004-11-23     2008-02-22           <NA>
## 3   11/21/05     1    1/11/06    2003-04-30     2005-11-21     2006-01-11
## 4     5/5/06     1     5/7/06    2006-03-25     2006-05-05     2006-05-07
## 5   11/13/07     0       <NA>    2004-09-01     2007-11-13           <NA>
## 6    2/28/08     0       <NA>    2003-12-02     2008-02-28           <NA>
##   death.indicator
## 1               0
## 2               0
## 3               0
## 4               1
## 5               0
## 6               0
```

3. Use the `init.date`, `last visit` and `death.date` to calculate a followup time, which is the difference between the first and either the last visit or a death event (whichever comes first). If these times are longer than 1 year, censor them.

```
data$followup <- NULL
for (i in 1:nrow(data)) {
    if (is.na(data$new_last.visit[i])) {
        data$followup[i] <- (data$new_date.death[i] - data$new_init.date[i])
    } else if (data$new_last.visit[i] < data$new_date.death[i] || is.na(data$new_date.d
        data$followup[i] <- (data$new_last.visit[i] - data$new_init.date[i])
    }
}
for (i in 1:nrow(data)) {
    if (data$followup[i] > 365) {
        data$followup[i] = NA
    }
```

```
}
```

4. Create another indicator variable representing loss to followup; that is, if their status 1 year after the first visit was unknown.

```
data$loss.followup <- 0
for (i in 1:nrow(data)) {
    if (!is.na(data$followup[i])) {
        data$loss.followup[i] = 1
    }
}
```

5. Recall our work in class, which separated the `init.reg` field into a set of indicator variables, one for each unique drug. Create these fields and append them to the database as new columns.

```
init.reg <- as.character(data$init.reg)
data$init.reg_list <- strsplit(init.reg, ",")
all_drugs <- unique(unlist(data$init.reg_list))
reg_drugs <- c()
for (drug in all_drugs) {
    reg_drugs <- cbind(reg_drugs, sapply(data$init.reg_list, function(x) drug %in%
        x))
}
head(reg_drugs)

##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11]
## [1,] TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2,] TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,] TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4,] TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [5,] TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [6,] TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18]
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE

reg_drugs <- data.frame(reg_drugs)
names(reg_drugs) <- all_drugs
data_merged <- cbind(data, reg_drugs)
head(data_merged)
```

```
##   male age aids cd4baseline logvl weight hemoglobin    init.reg init.date
## 1    1  25    0          NA    NA     NA         NA 3TC,AZT,EFV    7/1/03
## 2    1  49    0         143    NA  58.06         11 3TC,AZT,EFV  11/23/04
## 3    1  42    1         102    NA  48.08          1 3TC,AZT,EFV   4/30/03
## 4    0  33    0         107    NA  46.00         NA 3TC,AZT,NVP   3/25/06
## 5    1  27    0          52     4     NA         NA 3TC,D4T,EFV    9/1/04
## 6    0  34    0         157    NA  54.89         NA 3TC,AZT,NVP   12/2/03
##   last.visit death date.death new_init.date new_last.visit new_date.death
## 1    2/26/07     0       <NA>    2003-07-01     2007-02-26           <NA>
## 2    2/22/08     0       <NA>    2004-11-23     2008-02-22           <NA>
## 3   11/21/05     1    1/11/06    2003-04-30     2005-11-21     2006-01-11
## 4     5/5/06     1     5/7/06    2006-03-25     2006-05-05     2006-05-07
## 5   11/13/07     0       <NA>    2004-09-01     2007-11-13           <NA>
## 6    2/28/08     0       <NA>    2003-12-02     2008-02-28           <NA>
##   death.indicator followup loss.followup init.reg_list  3TC   AZT   EFV
## 1               0       NA             0 3TC, AZT, EFV TRUE  TRUE  TRUE
## 2               0       NA             0 3TC, AZT, EFV TRUE  TRUE  TRUE
## 3               0       NA             0 3TC, AZT, EFV TRUE  TRUE  TRUE
## 4               1       41             1 3TC, AZT, NVP TRUE  TRUE FALSE
## 5               0       NA             0 3TC, D4T, EFV TRUE FALSE  TRUE
## 6               0       NA             0 3TC, AZT, NVP TRUE  TRUE FALSE
##     NVP   D4T   ABC   DDI   IDV   LPV   RTV   SQV   FTC   TDF   DDC   NFV
## 1 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 3 FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 4  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 5 FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 6  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##     T20   ATV   FPV
## 1 FALSE FALSE FALSE
## 2 FALSE FALSE FALSE
## 3 FALSE FALSE FALSE
## 4 FALSE FALSE FALSE
## 5 FALSE FALSE FALSE
## 6 FALSE FALSE FALSE
```

6. The dataset **haart2.csv** contains a few additional observations for the same study. Import these and append them to your master dataset (if you were smart about how you coded the previous steps, cleaning the additional observations should be easy!).

```
# from git use: data <- read.table('../data/haart2.csv', sep=',', head=T)
data2 <- read.table("~/Bios301/datasets/haart2.csv", sep = ",", head = T)
# Converting init.date to date format & assigning it to a new variable,
# new_init.date
```

```r
new_init.date <- as.Date(data2$init.date, format = "%m/%d/%y")
# Verifying class of new variable
class(new_init.date)

## [1] "Date"

# Similar operations for last.visit and date.death:
new_last.visit <- as.Date(data2$last.visit, format = "%m/%d/%y")
class(new_last.visit)

## [1] "Date"

new_date.death <- as.Date(data2$date.death, format = "%m/%d/%y")
class(new_date.death)

## [1] "Date"

# Adding new variables to dataset2
data2$new_init.date <- new_init.date
data2$new_last.visit <- new_last.visit
data2$new_date.death <- new_date.death
# Creating death.indicator variable for data2
data2$death.indicator <- NULL
for (i in 1:nrow(data2)) {
    if (data2$death[i] == 1 & (data2$new_date.death[i] - data2$new_init.date[i]) <=
        365) {
        data2$death.indicator[i] <- 1
    } else data2$death.indicator[i] <- 0
}
head(data2)

##   male   age aids cd4baseline logvl weight hemoglobin    init.reg
## 1    0 27.00    0         232    NA     NA         NA 3TC,AZT,NVP
## 2    1 38.72    0         170    NA  84.00         NA 3TC,AZT,NVP
## 3    1 23.00   NA         154 3.996  65.50         14 3TC,DDI,EFV
## 4    0 31.00    0         236    NA  45.81         NA 3TC,D4T,NVP
##   init.date last.visit death date.death new_init.date new_last.visit
## 1   12/1/03     1/5/04     0         NA    2003-12-01     2004-01-05
## 2   9/26/02    3/29/04     0         NA    2002-09-26     2004-03-29
## 3   1/31/07    4/16/07     0         NA    2007-01-31     2007-04-16
## 4   12/3/03   10/11/07     0         NA    2003-12-03     2007-10-11
##   new_date.death death.indicator
## 1           <NA>               0
## 2           <NA>               0
## 3           <NA>               0
## 4           <NA>               0
```

```r
# Creating followup variable for data2
data2$followup <- NULL
for (i in 1:nrow(data2)) {
    if (is.na(data2$new_last.visit[i])) {
        data2$followup[i] <- (data2$new_date.death[i] - data2$new_init.date[i])
    } else if (data2$new_last.visit[i] < data2$new_date.death[i] || is.na(data2$new_dat
        data2$followup[i] <- (data2$new_last.visit[i] - data2$new_init.date[i])
    }
}
for (i in 1:nrow(data2)) {
    if (data2$followup[i] > 365) {
        data2$followup[i] = NA
    }
}
# Creating loss.followup variable for data2
data2$loss.followup <- 0
for (i in 1:nrow(data2)) {
    if (!is.na(data2$followup[i])) {
        data2$loss.followup[i] = 1
    }
}
init.reg <- as.character(data2$init.reg)
data2$init.reg_list <- strsplit(init.reg, ",")
all_drugs <- unique(unlist(data2$init.reg_list))
reg_drugs <- c()
for (drug in all_drugs) {
    reg_drugs <- cbind(reg_drugs, sapply(data2$init.reg_list, function(x) drug %in%
        x))
}
head(reg_drugs)
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
## [1,] TRUE  TRUE  TRUE FALSE FALSE FALSE
## [2,] TRUE  TRUE  TRUE FALSE FALSE FALSE
## [3,] TRUE FALSE FALSE  TRUE  TRUE FALSE
## [4,] TRUE FALSE  TRUE FALSE FALSE  TRUE
```

```r
reg_drugs <- data2.frame(reg_drugs)
```

```
## Error:  could not find function "data2.frame"
```

```r
names(reg_drugs) <- all_drugs
data2_merged <- cbind(data2, reg_drugs)
head(data2_merged)
```

```
##   male   age aids cd4baseline logvl weight hemoglobin    init.reg
## 1    0 27.00    0         232    NA     NA         NA 3TC,AZT,NVP
## 2    1 38.72    0         170    NA  84.00         NA 3TC,AZT,NVP
## 3    1 23.00   NA         154 3.996  65.50         14 3TC,DDI,EFV
## 4    0 31.00    0         236    NA  45.81         NA 3TC,D4T,NVP
##   init.date last.visit death date.death new_init.date new_last.visit
## 1   12/1/03     1/5/04     0         NA    2003-12-01     2004-01-05
## 2   9/26/02    3/29/04     0         NA    2002-09-26     2004-03-29
## 3   1/31/07    4/16/07     0         NA    2007-01-31     2007-04-16
## 4   12/3/03   10/11/07     0         NA    2003-12-03     2007-10-11
##   new_date.death death.indicator followup loss.followup init.reg_list    1
## 1           <NA>               0       35             1 3TC, AZT, NVP TRUE
## 2           <NA>               0       NA             0 3TC, AZT, NVP TRUE
## 3           <NA>               0       75             1 3TC, DDI, EFV TRUE
## 4           <NA>               0       NA             0 3TC, D4T, NVP TRUE
##       2     3     4     5     6
## 1  TRUE  TRUE FALSE FALSE FALSE
## 2  TRUE  TRUE FALSE FALSE FALSE
## 3 FALSE FALSE  TRUE  TRUE FALSE
## 4 FALSE  TRUE FALSE FALSE  TRUE
```

3 Question 3

15 points

The game of craps is played as follows. First, you roll two six-sided dice; let x be the sum of the dice on the first roll. If x is 7 or 11 you win, otherwise you keep rolling until either you get x again, in which case you also win, or until you get a 7 or 11, in which case you lose.

Write a program to simulate a game of craps. You can use the following snippet of code to simulate the roll of two (fair) dice:

```
x <- sum(ceiling(6*runif(2)))
```

The instructor should be able to easily import and run your program (function), and obtain output that clearly shows how the game progressed.

```
# rolling one die, selecting number 1-6, n times, with replacement
# sum of two die
roll <- function() {
    x1 <- sum(ceiling(6 * runif(2)))
    if (x1 == 7 || x1 == 11) {
        cat("Your first roll was a", x1, "\n")
        print("YOU WIN!")
        return()
```

```
    } else {
        i = 0
        while (i < 15) {
            x2 <- sum(ceiling(6 * runif(2)))
            if (x2 == x1) {
                cat("Your second roll was a", x2, "\n")
                cat("It took", i, "rolls to match your second roll", "\n")
                print("YOU WIN!")
                return()
            } else if (x2 == 7 || x2 == 11) {
                print("YOU LOSE!")
                return()
            } else i <- i + 1
        }
    }
}
roll()
```

```
## [1] "YOU LOSE!"
## NULL
```

```
# Remember to use test.rnw to find error
```