Homework 2

Louis Bensard

February 12, 2018

Homework 2 – Louis Bensard

Problem J-2.1

(a)

```
# This function computes the gradient of the log-likelihood

grad <- function (x, t) {
#x=(x1,x2,x3,x4), and t os the parameter (teta)

    dl = x[1]/(2+t)-(x[2]+x[3])/(1-t)+x[4]/t

}


# Secant Method for the multinomial
# t_0 = t_(n-1) and t_1 = t_n

secant <- function (maxit, x, t_0, t_1, tolgrad, tolerr, part){
#part is a char so we print the right part of the problem
    mre = abs(t_1-t_0)/max(1,abs(t_1))

    if(part == 'b'){

        #Output for part (c)==b)
        cat("Iteration number n", "   Value of theta_n", "     MRE", "    Grad(theta_n)\n")
        cat(sprintf('%2.0f                    %3.3f             %1.1e      %1.1e\n',
                    0, t_0,mre, grad(x,t_0)))
    }

    if(part == 'c'){

        #Output for part (c)
        beta = (1+sqrt(5))/2 #golden ratio
        t_star = (-1657 + sqrt(3728689))/7680
        cv_ratio = 0 #we arbitrary set the convergence ratio to 0 for n=0
        digits = -log10(abs(t_0 - t_star)/abs(t_star))
        cat("Iteration number n", "   Value of theta_n", "     CV ratio", "    # of digits\n")
        cat(sprintf('%2.0f                    %3.3f                %1.1e      %2.1f\n',
                    0, t_0,cv_ratio, digits))

    }

    if((part !='b') & (part !='c')){
```

```
        return("Precise part for the algorithm to run (a or b)")
    }


    for (it in 1:maxit){

        dl0 = grad(x,t_0)
        dl1 = grad(x,t_1)

        if((abs(dl1) < tolgrad) & (mre < tolerr)){

            break
        }

        mre = abs(t_1-t_0)/max(1,abs(t_1))

        if(part == 'b'){

            # Output for part (b)
            cat(sprintf('%2.0f                    %12.12f     %1.1e      %1.1e\n',
                        it, t_1, mre, grad(x,t_1)))
        }

        if(part == 'c'){

            #Output for part (c)
            cv_ratio = abs(t_1 - t_star)/(abs(t_0 - t_star)^beta)
            digits = -log10(abs(t_1 - t_star)/abs(t_star))
            cat(sprintf('%2.0f                    %12.12f        %1.1e      %2.1f\n',
                      it, t_1,cv_ratio, digits))
        }

        t_new = t_1 - dl1*(t_1-t_0)/(dl1-dl0)
        t_0 = t_1
        t_1 = t_new

      }
}
```

(b)

```
maxit = 20
x = c(1997,907,904,32)
tolgrad = 1e-9 ; tolerr = 1e-6
t_0=0.02; t_1=0.01; secant(maxit, x, t_0, t_1, tolgrad, tolerr, 'b')
```

```
## Iteration number n    Value of theta_n      MRE     Grad(theta_n)
##  0                     0.020              1.0e-02      7.4e+02
##  1                     0.010000000000     1.0e-02      2.4e+03
##  2                     0.024561848011     1.5e-02      4.3e+02
##  3                     0.027823212918     3.3e-03      2.7e+02
##  4                     0.033351047002     5.5e-03      6.8e+01
```

```
##  5                    0.035197558267     1.8e-03      1.3e+01
##  6                    0.035646185180     4.5e-04      7.9e-01
##  7                    0.035674309037     2.8e-05      9.6e-03
##  8                    0.035674655571     3.5e-07      6.9e-06
```

(c)

```
t_0=0.02; t_1=0.01; secant(maxit, x, t_0, t_1, tolgrad, tolerr, 'c')
```

```
## Iteration number n    Value of theta_n      CV ratio     # of digits
##  0                     0.020                 0.0e+00      0.4
##  1                     0.010000000000        2.1e+01      0.1
##  2                     0.024561848011        4.2e+00      0.5
##  3                     0.027823212918        1.1e+01      0.7
##  4                     0.033351047002        5.9e+00      1.2
##  5                     0.035197558267        8.7e+00      1.9
##  6                     0.035646185180        6.7e+00      3.1
##  7                     0.035674309037        7.9e+00      5.0
##  8                     0.035674655571        7.1e+00      8.2
```

(d)

With trials and error, the only value that "stabilizes" the convergence ratio is the famous "Golden Ratio", which is about 1.61803. Since $1 < 1.61803 < 2$ then the secant method is super-linearly convergent.

(e)

```
t_0=0.5; t_1=0.01; secant(maxit, x, t_0, t_1, tolgrad, tolerr, 'c')
```

```
## Iteration number n    Value of theta_n      CV ratio     # of digits
##  0                     0.500                 0.0e+00      -1.1
##  1                     0.010000000000        8.9e-02      0.1
##  2                     0.236113205980        7.5e+01      -0.7
##  3                     0.154232612419        1.6e+00      -0.5
##  4                     -0.091534935810        4.0e+00      -0.6
##  5                     -5.415293039221        1.5e+02      -2.2
##  6                     -57.393356615762        3.7e+00      -3.2
##  7                     -61.758036194809        8.8e-02      -3.2
##  8                     -118.507564138350        1.5e-01      -3.5
##  9                     -179.641949878101        7.9e-02      -3.7
## 10                     -297.549730154520        6.7e-02      -3.9
## 11                     -476.603259666372        4.7e-02      -4.1
## 12                     -773.572321173815        3.6e-02      -4.3
## 13                     -1249.599529131543        2.6e-02      -4.5
## 14                     -2022.598694332578        2.0e-02      -4.8
## 15                     -3271.626847784568        1.5e-02      -5.0
## 16                     -5293.655269176950        1.1e-02      -5.2
## 17                     -8564.712525064278        8.1e-03      -5.4
## 18                     -13857.798623401799        6.0e-03      -5.6
## 19                     -22421.942237832507        4.5e-03      -5.8
## 20                     -36279.172111427077        3.3e-03      -6.0
```

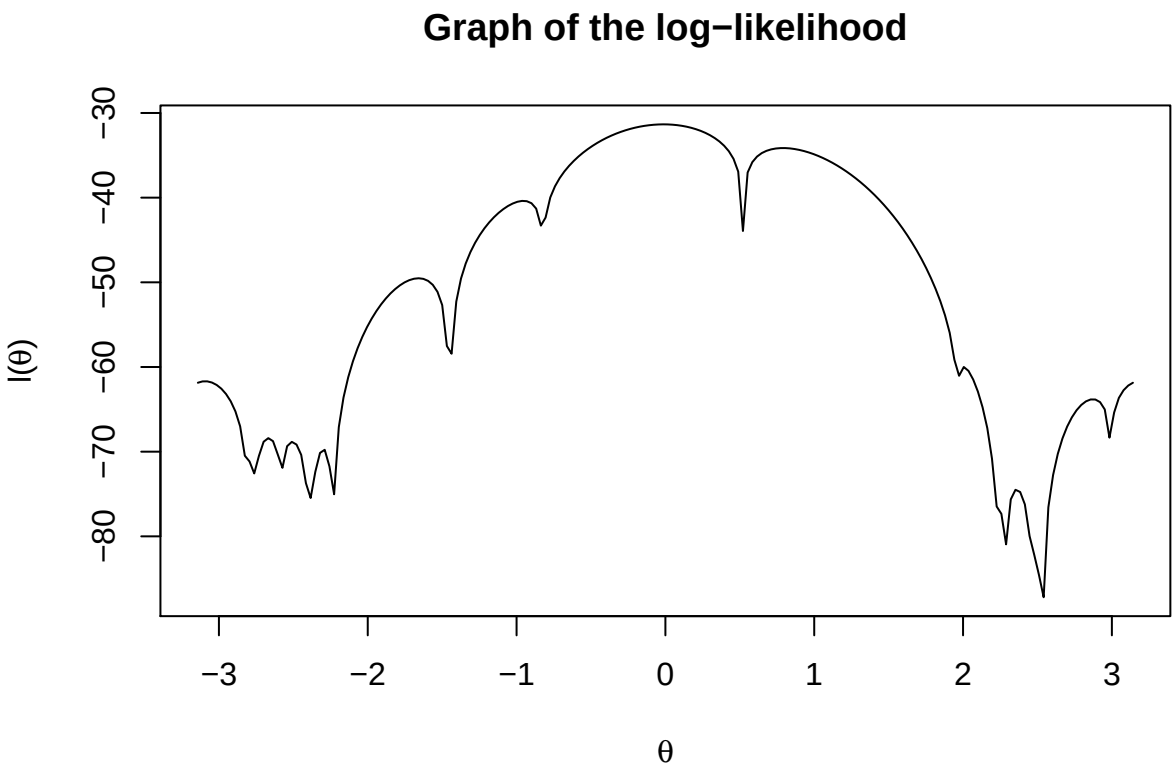
Clearly, the value of θ_n blows up and thus the secant method does not converges for the initial values 0.5 and 0.01.

Problem GH-2.2

```
data = c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96, 2.53, 3.88, 2.22,
         3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)
```

(a)

```
theta = seq(-pi, pi, length=200)

l_theta <- function(theta, x){ #both theta and x are vectors of length 20

    y = rep(0, 200)

    for(i in 1:200){

        for(j in 1:20){

            y[i] = y[i] + log((1-cos(x[j]-theta[i]))/(2*pi))

        }
    }

    return(y)

}

plot(theta, l_theta(theta, data), type="l", xlab=expression(theta),
     ylab=expression(paste('l(', theta, ')')))
title('Graph of the log-likelihood')
```



(b)

```
MME = asin(mean(data) - pi)
cat('MME =',MME)
```

```
## MME = 0.05844061
```

(c)

```
# This function computes the gradient of the log-likelihood
grad <- function (x, t) {

    dl = sum(1/tan((t-x)/2))
  }

# This function computes the Hessian of the log-likelihood
Hess <- function (x, t) {

    ddl = sum(1/(cos(t-x) - 1))

}


Newton <- function (maxit, x, t, tolgrad, tolerr, print_output) {
```

```
    #print_output is a boolean. if TRUE we print the output.

    mre = 0 #arbitrary set to 0 for n=0

    if(print_output){

        cat("Iteration number n", "   Value of theta_n", "     MRE", "    Grad(theta_n)\n")
        cat(sprintf('%2.0f                      %3.3f              %1.1e      %1.1e\n',
                    0, t,mre, grad(x,t)))
    }

    for (it in 1:maxit){

        dl = grad(x,t)
        ddl = Hess(x,t)

        if((abs(dl) < tolgrad) & (mre < tolerr)){

            break
        }

        t_new = t - dl/ddl

        mre = abs(t_new-t)/max(1,abs(t_new))

        if(print_output){
            cat(sprintf('%2.0f                        %12.12f     %1.1e      %1.1e\n',
                        it, t_new, mre, grad(x,t_new)))
        }

        t = t_new

      }

    #we need to store this value later in the code
    return(t)

}


# Run Newton, using various starting values
maxit = 20
tolgrad = 1e-9 ; tolerr = 1e-6


# starting with t = MME
t=MME; Newton(maxit, data, t, tolgrad, tolerr, TRUE)
```

```
## Iteration number n    Value of theta_n      MRE     Grad(theta_n)
##  0                     0.058              0.0e+00      -1.6e+00
##  1                     -0.009098573745     6.8e-02       -6.3e-02
##  2                     -0.011968737913     2.9e-03       -7.2e-05
##  3                     -0.011972002283     3.3e-06       -9.1e-11
##  4                     -0.011972002287     4.1e-12       -4.3e-15
```

```
## [1] -0.011972
```

```
# abline(v=-0.0119718684777, lty=2) #verification of the result


# starting with t = -2.7
t=-2.7; Newton(maxit, data, t, tolgrad, tolerr, TRUE)
```

```
## Iteration number n    Value of theta_n      MRE     Grad(theta_n)
##  0                     -2.700             0.0e+00      2.8e+01
##  1                     -2.674113655831    9.7e-03      5.5e+00
##  2                     -2.666793927068    2.7e-03      7.0e-02
##  3                     -2.666699927130    3.5e-05      7.6e-07
##  4                     -2.666699926101    3.9e-10      1.5e-13
```

```
## [1] -2.6667
```

```
# abline(v=-2.666699927130, lty=2) #verification of the result


# starting with t = 2.7
t=2.7; Newton(maxit, data, t, tolgrad, tolerr, TRUE)
```

```
## Iteration number n    Value of theta_n      MRE     Grad(theta_n)
##  0                     2.700              0.0e+00      3.9e+01
##  1                     2.825724484570     4.4e-02      1.0e+01
##  2                     2.877549108301     1.8e-02      -1.1e+00
##  3                     2.873184456115     1.5e-03      -2.3e-02
##  4                     2.873094549040     3.1e-05      -8.7e-06
##  5                     2.873094514245     1.2e-08      -1.3e-12
```

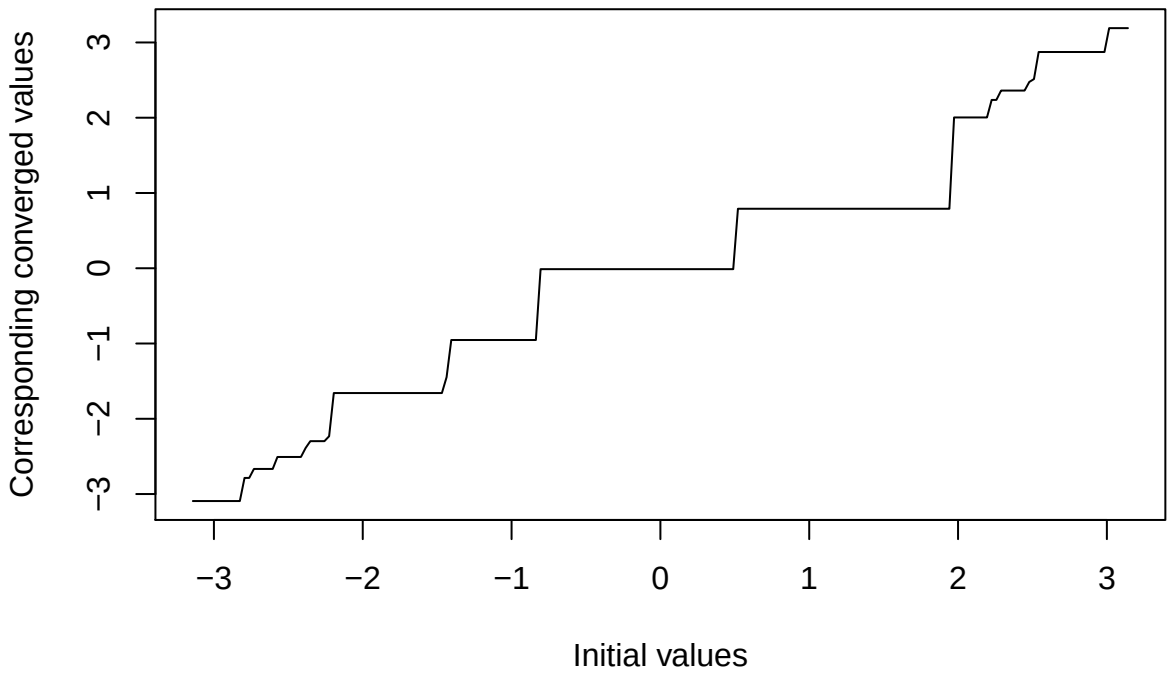
```
## [1] 2.873095
```

```
# abline(v=2.873094549040, lty=2) #verification of the result
```

(d)

```
# Part 1 of the question:

length_vect = 200
t = seq(-pi, pi, length=length_vect)
extrema = list()

for(i in 1:length_vect){

    # We store the starting value and the max it converges to in the same list

    extrema[[i]] = c(t[i], Newton(maxit, data, t[i], tolgrad, tolerr, FALSE))
}

x = rep(0, length_vect)
y = rep(0, length_vect)

for(i in 1:(length_vect)){
```

```
    x[i] = extrema[[i]][1]
    y[i] = extrema[[i]][2]

}

plot(x,y, type='l', xlab='Initial values', ylab='Corresponding converged values')
title("Newton-Raphson algorithm: Starting values vs Converged values")
```




```
# Part 2 of the question
# We are going to partition extrema into a smaller list of elements
# Each element will correspond to an unique outcome of the optimization

#arbitrary first element so we can compare the first max to something
modes = c(-10)
length_modes = length(modes)
start_val_partition = list(c(-10)) #arbitrary first value value


for(i in 1:length_vect){

    # start_val_partition[[length_modes]] = c(start_val_partition[[length_modes]], extrema[[i]][1])
    # We consider two extrema equal if they have up to 4 significant digits in common
    if(abs(modes[length_modes] - extrema[[i]][2]) > 1e-4){

        length_modes = length_modes +1
        modes[length_modes] = extrema[[i]][2]
```

```
        start_val_partition= c(start_val_partition, c(-15)) #arbitrary -15 value
    }

    start_val_partition[[length_modes]] = c(start_val_partition[[length_modes]], extrema[[i]][1])

}


#we delete the arbitrary -15 value
for(i  in 1:length_modes){

    start_val_partition[[i]] = start_val_partition[[i]][-1]

}

# We don't need the arbitrary first element anymore
start_val_partition[[1]] = NULL
modes = modes[-1]
```

We have 19 different local modes when we use 200 starting values between $-\pi$ and π. Not all local modes are "equal", some correspond to more starting values than others.This makes sense since the extrema of the log-likelihood are not homogeneously spread along the x-axis (extrema density high on the left and right of the graph, but low in the middle). The more dense of extrema it is, the more different modes we get. We can observe these modes either graphically on the Starting values vs Converged values graph or by analysing the following partitionned list of starting values `start_val_partition`.

```
print(start_val_partition)
```

```
## [[1]]
##  [1] -3.141593 -3.110019 -3.078445 -3.046871 -3.015297 -2.983724 -2.952150
##  [8] -2.920576 -2.889002 -2.857428 -2.825855
##
## [[2]]
## [1] -2.794281 -2.762707
##
## [[3]]
## [1] -2.731133 -2.699560 -2.667986 -2.636412 -2.604838
##
## [[4]]
## [1] -2.573264 -2.541691 -2.510117 -2.478543 -2.446969 -2.415395
##
## [[5]]
## [1] -2.383822
##
## [[6]]
## [1] -2.352248 -2.320674 -2.289100 -2.257526
##
## [[7]]
## [1] -2.225953
##
## [[8]]
##  [1] -2.194379 -2.162805 -2.131231 -2.099657 -2.068084 -2.036510 -2.004936
##  [8] -1.973362 -1.941788 -1.910215 -1.878641 -1.847067 -1.815493 -1.783919
## [15] -1.752346 -1.720772 -1.689198 -1.657624 -1.626050 -1.594477 -1.562903
```

```
## [22] -1.531329 -1.499755 -1.468181
##
## [[9]]
## [1] -1.436608
##
## [[10]]
##  [1] -1.4050339 -1.3734601 -1.3418863 -1.3103125 -1.2787387 -1.2471649
##  [7] -1.2155911 -1.1840173 -1.1524435 -1.1208697 -1.0892959 -1.0577221
## [13] -1.0261484 -0.9945746 -0.9630008 -0.9314270 -0.8998532 -0.8682794
## [19] -0.8367056
##
## [[11]]
##  [1] -0.80513179 -0.77355799 -0.74198419 -0.71041040 -0.67883660
##  [6] -0.64726281 -0.61568901 -0.58411522 -0.55254142 -0.52096763
## [11] -0.48939383 -0.45782003 -0.42624624 -0.39467244 -0.36309865
## [16] -0.33152485 -0.29995106 -0.26837726 -0.23680347 -0.20522967
## [21] -0.17365588 -0.14208208 -0.11050828 -0.07893449 -0.04736069
## [26] -0.01578690  0.01578690  0.04736069  0.07893449  0.11050828
## [31]  0.14208208  0.17365588  0.20522967  0.23680347  0.26837726
## [36]  0.29995106  0.33152485  0.36309865  0.39467244  0.42624624
## [41]  0.45782003  0.48939383
##
## [[12]]
##  [1] 0.5209676 0.5525414 0.5841152 0.6156890 0.6472628 0.6788366 0.7104104
##  [8] 0.7419842 0.7735580 0.8051318 0.8367056 0.8682794 0.8998532 0.9314270
## [15] 0.9630008 0.9945746 1.0261484 1.0577221 1.0892959 1.1208697 1.1524435
## [22] 1.1840173 1.2155911 1.2471649 1.2787387 1.3103125 1.3418863 1.3734601
## [29] 1.4050339 1.4366077 1.4681815 1.4997553 1.5313291 1.5629029 1.5944767
## [36] 1.6260505 1.6576243 1.6891981 1.7207719 1.7523457 1.7839194 1.8154932
## [43] 1.8470670 1.8786408 1.9102146 1.9417884
##
## [[13]]
## [1] 1.973362 2.004936 2.036510 2.068084 2.099657 2.131231 2.162805 2.194379
##
## [[14]]
## [1] 2.225953 2.257526
##
## [[15]]
## [1] 2.289100 2.320674 2.352248 2.383822 2.415395 2.446969
##
## [[16]]
## [1] 2.478543
##
## [[17]]
## [1] 2.510117
##
## [[18]]
##  [1] 2.541691 2.573264 2.604838 2.636412 2.667986 2.699560 2.731133
##  [8] 2.762707 2.794281 2.825855 2.857428 2.889002 2.920576 2.952150
## [15] 2.983724
##
## [[19]]
## [1] 3.015297 3.046871 3.078445 3.110019 3.141593
```

(e)

The Newton algorithm converges to the local mode -0.9533363 with starting value $\theta_0 = -0.8367056$, but converges to the local mode -0.0119720 with starting value $\theta_0 = -0.8051318$. So these two close starting values converges to two different local modes.