Econ 220B PS 2

AC Watt

2021-12-17

Contents

Part a

Construct sample statistics similar to Tabel IIa and IIb on page 1003.

Table 1: Summary of Replacement Data (Subsample of buses for which at least 1 replacement occurred)

	Mileage at Replacement				Elapsed Time (Months)				
Bus Group	Max	Min	Mean	Standard Deviation	Max	Min	Mean	Standard Deviation	Number of Observations
1	0	0	0	0	0	0	0.0	0.0	0
2	0	0	0	0	0	0	0.0	0.0	0
3	273,400	124,800	199,733	37,459	74	38	59.1	10.9	27
4	387,300	121,300	257,336	65,477	116	28	73.7	23.3	33
5	322,500	118,000	245,290	60,257	127	31	85.4	29.7	11
6	237,200	82,400	150,785	61,006	127	49	74.7	35.2	7
7	331,800	121,000	208,962	48,980	104	41	68.3	16.9	27
8	297,500	132,000	186,700	43,956	104	36	58.4	22.2	19
Full Sample	387,300	82,400	216,354	60,475	127	28	68.1	22.4	124

Table 2: Summary of Censored Data (subsample of buses for which no replacements occurred)

	Mileage at May 1, 1985				Elapsed Time (Months)				
Bus Group	Max	Min	Mean	Standard Deviation	Max	Min	Mean	Standard Deviation	Number of Observations
1	120,151	65,643	100,116	12,929	25	25	25.0	0.0	15
2	161,748	142,009	151,182	8,529	49	49	49.0	0.0	4
3	280,802	199,626	250,766	21,324	70	70	70.0	0.0	21
4	352,450	310,910	337,221	17,802	117	117	117.0	0.0	5
5	326,843	326,843	326,843	0	126	126	126.0	0.0	1
6	299,040	232,395	265,263	33,331	126	126	126.0	0.0	3
7	0	0	0	0	0	0	0.0	0.0	0
8	0	0	0	0	0	0	0.0	0.0	0
Full Sample	352,450	65,643	207,781	85,207	126	25	63.9	33.5	49

Part b

Estimate his model using a two-step procedure. Please briefly document your estimation procedure in words. Create a table similar to the first five columns of Table IX on page 1021. If you forward simulate the value function, see Pakes Pollard (1989) for reference on how to draw errors and compute confidence intervals.

I ran out of time to attempt the H-M two step method. Alas, I spent all my time on part C. I hope to come back and finish this next Summer because it seems like the H-M method is of more practical use! I'll probably try to use it to estimate my model of firm costs in reaction to Purple Air pollution sensor adoption.

Part c

Estimate his model using a nested fixed point procedure. Create a table similar to the first five columns of Table IX on page 1021.

Estimating θ_3

1. Break the mileage into 5,000 mile bins
2. Create g-bins of 0, 1, and 2 for mileage in the intervals [0, 5000), [5000, 10,000) and [10,000, $+\infty$).
3. Define the cost functions to be estimated
4. Estimate θ_3 using Maximum Likelihood Does the mileage stay in the current bin, or move up to the next bin, conditional on no replacement?

Assuming the buses within a group are independent and come from the same discrete transition probability distribution, then the likelihood function is just the product of the likelihood functions for all the buses in the group, which are in turn, the product of the probabilities of observing the transitions in the data.

For group g with n_g buses and T_g periods in it, the joint likelihood function ℓ_g^1 is

$$\ell_g^1(x_{1,1},...,x_{T_g,1},...,x_{T_g,n_g},i_{1,1},...,i_{T,n_g}|x_{0,1},...,x_{0,n_g},i_{0,1},...,i_{0,n_g},\theta) = \prod_{k=1}^{n_g}\prod_{t=1}^{T_g} p(x_{t,k}|x_{t-1,k},i_{t-1,k},\theta_3)$$

Estimating this using ML gives us the table below:

Table 3: Partial Table V: Within Group ML Estimates of Mileage Transition Probabilities

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8
theta31	0.188	0.39	0.273	0.394	0.487	0.618	0.601	0.723
theta32	0.798	0.60	0.717	0.593	0.508	0.382	0.396	0.277
theta33	0.014	0.01	0.010	0.013	0.005	0.000	0.002	0.000

Estimating the fixed point EV

Let N be the number of mileage bins (N=90 here and in Rust's paper). Following the directions in Rust's programming manual, we need to do the following to estimate the fixed point EV at a given value of θ:

1. Define starting values for θ_1, θ_3, RC (tr in Rust's paper), β, and the cost function. I am using the single-parameter linear cost function, as Rust does in his programming manual examples. $c(x,\theta_1) = \theta_1 x$
2. Generate the $N \times N$ markov transition matrix P by applying the ML-estimated θ_3 transition probabilities to the $N \times N$ matrix, where the i,j element represents the probabilty of transitioning from the i^{th} mileage bin to the j^{th} bin.
3. Generate the $\frac{dc}{d\theta_1}$ vector (vector of the derivative evaluated at each mileage bin).
4. Generate the $N \times N$ value function matrix T_θ based on current values of θ.
5. Use the sequential approximation to get closer to the EV fixed point (updating EV by applying T_θ repeatedly).
6. Generate p_k (the $N \times 1$ values for $P(0|x,\theta)$) and T'_θ (the partial derivative of T_θ with respect to EV).
7. Use p_k and T'_θ in the Newton Kantorovich approximation to get within the desired accuracy threshold for EV.

Estimating RC, θ_1, θ_3

Now that we have the fixed point $EV_\theta(x)$, we need climb the hill to reach the loglikelihood-maximizing θ parameter. This uses the BHHH method. I was unable to fix an issue I was having in the final part of the BHHH method.

To implement the BHHH method, I:

1. Define starting values for θ, $\lambda 0$, $\lambda 1$, `df0` and `df1`. The last two are parameters that help determine the optimal step size $\lambda 2$.
2. Find the fixed point EV for these starting values.
3. Find the loglikelihood (LL), it's θ derivative, and the Hessian matrix (H).
4. Using H and the derivative of LL, I find the new direction of θ.
5. Using $\lambda 0$, $\lambda 1$, `df0` and `df1`, I find the correct step size $\lambda 2$.
6. Update θ with a $\lambda 2$ sized step in the new θ direction.
7. Update the fixed point $EV_\theta(x)$ and repeat the above steps.

I could not fix a bug that send the BHHH method off in the wrong direction. I am attaching my code, which I wrote from scratch using the Rust GAUSS programming manual as my guide and translating some of his chunks of code into R.

As a sidenote, I had a lot of fun programming this. I spent a significant amount of time doing it over the last month and learned a lot about how fast different methods of programming can work (I was able to update my EV fixed point estimation using Rust's manual, changing it from taking several minutes to less than a hundredth of a second). Matrix algorithms are much more efficient. I'm planning to take a matrix algorithms course taught by James Demmel in the next couple years because it seems like such an important skill.

I wish I had started with the H-M two step first, since that's the more updated method, but this was really fun reading through Rust's paper and manual in excruciating detail.

The basic parameter iteration in BHHH given in equations (3.7), (3.10), and (3.13) have a translation into GAUSS code given by

Appendix A: R Code

```
rm(list=ls())
knitr::opts_chunk$set(echo = F)
# stargazer table type (html, latex, or text)
# Change to latex when outputting to PDF, html when outputting to html
table_type = "latex"
Cache = TRUE

# Packages
library(stargazer)
library(ggplot2)
library(readxl)
library(tidyverse)
library(lubridate)
library(kableExtra)
library(docstring)
# Load top 11 rows only to get replacement dates and mileage
f = 'rust_data_clean.xlsx'
# Load additional tab with column names in it (tab header_names does not appear in the original data)
headers = read_excel(f, sheet='header_names', col_names='header')
sheets = excel_sheets(f)[1:8]

get_headers = function(f, sheets, i) {
    # Return dataframe of transformed i'th sheet
    read_excel(f, sheet=sheets[i], n_max=11, col_names=F, col_types="numeric") %>%
    t() %>%
    data.frame(stringsAsFactors = F) %>%
    mutate(sheet = sheets[i],
           group = i)
}

# Initialize dataframe
replacement_dates = get_headers(f, sheets, 1)

# Fill dataframe with other sheets
for (i in 2:8) {
    df_ = get_headers(f, sheets, i)
    replacement_dates = rbind(replacement_dates, df_)
}

# Pivot dataframe and create variables
replacement_dates = replacement_dates %>%
    rename(!!set_names(paste0("X", 1:11), str_replace_all(headers$header, " ", "_"))) %>%
    mutate(purchase_date = ymd(str_c(purchase_year, purchase_month, "-01")),
           replace_date1 = ymd(str_c(engine_replacement_1_year, engine_replacement_1_month, "-01")),
           replace_date2 = ymd(str_c(engine_replacement_2_year, engine_replacement_2_month, "-01")),
           pivot_months_to_replace1 = interval(purchase_date, replace_date1) %/% months(1) + 1,
           pivot_months_to_replace2 = interval(replace_date1, replace_date2) %/% months(1) + 1,
           pivot_replace_mileage1 = ifelse(engine_replacement_1_odometer_reading == 0, NA,
                                           engine_replacement_1_odometer_reading),
           pivot_replace_mileage2 = ifelse(engine_replacement_2_odometer_reading == 0, NA,
                                           engine_replacement_2_odometer_reading - engine_replacement_1
```

```
          initial_odometer_reading_date = ymd(str_c(initial_odometer_reading_year, initial_odometer_rea
    select(bus_number, purchase_date, replace_date1, pivot_months_to_replace1, replace_date2, pivot_mont

# Make Table IIa
replacement_dates %>%
    pivot_longer(starts_with("pivot_"),
                 names_to = c(".value", "replacement"),
                 names_pattern = "pivot_(.*)(.)",
                 values_drop_na = T
    ) %>%
    mutate(group = as.character(group)) %>%
    bind_rows(mutate(., group = "Full Sample")) %>%
    group_by(group) %>%
    summarize(max_mileage = max(replace_mileage, na.rm=T),
              min_mileage = min(replace_mileage, na.rm=T),
              mean_mileage = as.integer(mean(replace_mileage, na.rm=T)),
              sd_mileage = as.integer(sd(replace_mileage, na.rm=T)),
              max_months = max(months_to_replace, na.rm=T),
              min_months = min(months_to_replace, na.rm=T),
              mean_months = mean(months_to_replace, na.rm=T),
              sd_months = sd(months_to_replace, na.rm=T),
              n = n()) %>%
    complete(group = c(as.character(1:8), "Full Sample")) %>%
    mutate(across(everything(), ~replace_na(.x, 0))) %>%
    kbl(caption = "Summary of Replacement Data (Subsample of buses for which at least 1 replacement occu
        col.names = c('Bus Group', 'Max', 'Min', 'Mean', 'Standard Deviation',
                      'Max', 'Min', 'Mean', 'Standard Deviation', 'Number of Observations'),
        align = 'cc', digits = 1, booktabs = T, format.args = list(big.mark = ",")
    ) %>%
    kable_styling(latex_options = "HOLD_position") %>%
    column_spec(1, width = "1.5cm") %>%
    column_spec(c(2:4, 6:8), width = "1cm") %>%
    column_spec(c(5, 9), width = "1.5cm") %>%
    column_spec(10, width = "2cm") %>%
    add_header_above(c(" " = 1, "Mileage at Replacement" = 4, "Elapsed Time (Months)" = 4, " " = 1)) %>%
    row_spec(8, hline_after = T)

# Load rest of data below row 11 to get mileage readings
f = 'rust_data_clean.xlsx'
sheets = excel_sheets(f)[1:8]

get_data = function(f, sheet_, replacement_dates) {
    # Return transformed dataframe from sheet
    read_excel(f, sheet=sheet_, skip=11,
               col_names=paste0('bus_', filter(replacement_dates, sheet==sheet_)$bus_number),
               col_types="numeric") %>%
        mutate(sheet = sheet_,
               m = row_number()) %>%
        pivot_longer(starts_with("bus_"),
                     names_to = "bus_number",
                     names_prefix = 'bus_') %>%
        mutate(bus_number = as.numeric(bus_number),
               date = filter(replacement_dates, sheet==sheet_, bus_number==bus_number)$initial_odometer_
```

```
        rename(c('mileage' = 'value'))
}

# Initialize dataframe
data = get_data(f, sheets[1], replacement_dates)

# Get rest of sheets
for (i in 2:8) {
    df_ = get_data(f, sheets[i], replacement_dates)
    data = rbind(data, df_)
}


# Merge actual replacement mileage from data at replacement date (not states odometer reading at replac
replacement_dates = replacement_dates %>%
    left_join(data %>% select(-m),
              by = c('bus_number'='bus_number', 'sheet' = 'sheet', 'replace_date1' = 'date')) %>%
    rename(mileage_replace1 = mileage) %>%
    left_join(data %>% select(-m),
              by = c('bus_number'='bus_number', 'sheet' = 'sheet', 'replace_date2' = 'date')) %>%
    rename(mileage_replace2 = mileage)

# Join on variables from headers and create new variables
data = data %>%
    arrange(sheet, bus_number) %>%
    left_join(replacement_dates %>% select(bus_number,
                                           initial_odometer_reading_date,
                                           replace_date1,
                                           replace_date2,
                                           mileage_replace1,
                                           mileage_replace2,
                                           group),
              by = 'bus_number') %>%
    mutate(date2 = initial_odometer_reading_date + months(m - 1),
           group = as.character(group),
           replaced = ifelse((date == replace_date1 & !is.na(replace_date1)) | (date == replace_date2 &
           x = case_when(date > replace_date2 ~ mileage - mileage_replace2,
                         date > replace_date1 ~ mileage - mileage_replace1,
                         TRUE ~ mileage))

# Make Table IIa
data %>%
    filter(is.na(replace_date1)) %>%
    arrange(date) %>%
    group_by(group) %>%
    filter(date == last(date), .by_group = TRUE) %>%
    bind_rows(mutate(., group = "Full Sample")) %>%
    summarize(max_mileage = max(mileage, na.rm=T),
              min_mileage = min(mileage, na.rm=T),
              mean_mileage = as.integer(mean(mileage, na.rm=T)),
              sd_mileage = as.integer(sd(mileage, na.rm=T)),
              max_months = max(m, na.rm=T),
              min_months = min(m, na.rm=T),
```

```
            mean_months = mean(m, na.rm=T),
            sd_months = sd(m, na.rm=T),
            n = n()) %>%
    complete(group = c(as.character(1:8), "Full Sample")) %>%
    mutate(across(everything(), ~replace_na(.x, 0))) %>%
    kbl(caption = "Summary of Censored Data (subsample of buses for which no replacements occurred)",
        col.names = c('Bus Group', 'Max', 'Min', 'Mean', 'Standard Deviation',
                      'Max', 'Min', 'Mean', 'Standard Deviation', 'Number of Observations'),
        align = 'cc', digits = 1, booktabs = T, format.args = list(big.mark = ",")
    ) %>%
    kable_styling(latex_options = "HOLD_position") %>%
    column_spec(1, width = "1.5cm") %>%
    column_spec(c(2:4, 6:8), width = "1cm") %>%
    column_spec(c(5, 9), width = "1.5cm") %>%
    column_spec(10, width = "2cm") %>%
    add_header_above(c(" " = 1, "Mileage at May 1, 1985" = 4, "Elapsed Time (Months)" = 4, " " = 1)) %>%
    row_spec(8, hline_after = T)

# Create 5,000 mile bins
data = data %>%
    mutate(x5000 = cut(x,
                       breaks = seq(0, max(data$x)+5000, by=5000),
                       labels = F,
                       include.lowest = T)) %>%
    # Create g = number of x bins increased from previous period
    group_by(bus_number) %>%
    mutate(g = c(0,diff(x5000)),
           g = ifelse(g < 0, 0, g))

#########################################################
#                PART C
#########################################################

# Estimate theta3 using ML
library(stats4)

t1 = 0.34
t2 = 0.33
t3 = 0.33
theta3.start = c(t1, t2, t3)
data_temp = data %>%
    # Remove the initial month, and the months when replacements happened
    filter(m != 1, replaced != 1) %>%
    arrange(date) %>%
    group_by(group, bus_number) %>%
    distinct(mileage, .keep_all = T) %>%
    ungroup() %>%
    arrange(group, bus_number, date)


single_llh = function(theta3, g_vec) {
    #' Return product of all the transition probabilities
    #' @param theta3 vector of 3 transition probabilities
```

```
    #' @param g_vec vector of g values, how many x-bins did this bus advance this
    #' period? between 0 and 2 for our sample

    # convert g_vec to indicies of the theta3 vector (theta3[1] = probability of moving zero x-bins)
    index = as.vector(g_vec) + 1
    # Estimate joint probability conditional on theta3 (product of each bus
    # event of advancing the specified number of bins)
    s = 0
    for (idx in index) {
        s = s + log(theta3[idx])
    }
    return(s)
}


minus_joint_llh = function(theta3, df) {
    #' Return product of all transition probabilities for all buses in group

    # If any theta is negative, return NA to prevent using this as solution
    # if (min(theta3) < 0) return(Inf)
    # if (max(theta3) > 1) return(Inf)
    # if (sum(theta3) > 1) return(Inf)

    # Make sure theta3 items sum to 1
    # theta3 = theta3 / sum(theta3)
    # Iterate through buses, get log likelihood for each bus, add up all bus LL's
    buses = unique(df$bus_number)
    nLL = 0
    for (bus in buses) {
        gvec = filter(df, bus_number == bus)$g
        nLL = nLL - single_llh(theta3, gvec) # changing to negative LL using (-)
    }
    penalty1 = (sum(theta3) - 1)^2 * 1e6
    penalty2 = ifelse(max(theta3) > 1, (max(theta3) - 1)^2 * 1e8, 0)
    penalty3 = ifelse(min(theta3) < 0, (min(theta3) - 0)^2 * 1e8, 0)
    # print(nLL + penalty1 + penalty2 + penalty3)
    return(nLL + penalty1 + penalty2 + penalty3) # + penalty1 + penalty2 + penalty3
}

# data_g = data_temp %>% filter(group == 8)
# out1 = nlm(minus_joint_llh, theta3.start, df = data_g, print.level = 1)
# sum(out1$estimate)


# use nlm() to minimize -log(likelihood(param))
theta3_df = data.frame(theta31=double(), theta32=double(), theta33=double(), group=integer())
for (group_ in 1:8) {
    data_g = data_temp %>% filter(group == group_)
    out = nlm(minus_joint_llh, theta3.start, df = data_g, print.level = 1)
    est = out$estimate / sum(out$estimate)
    df_ = data.frame(theta31 = est[1], theta32 = est[2], theta33 = est[3], group=group_)
    theta3_df = rbind(theta3_df, df_)
}
```

```r
# Make theta3 ML estimate table
theta3_df %>% select(theta31, theta32, theta33) %>% t() %>%
    kbl(caption = "Partial Table V: Within Group ML Estimates of Mileage Transition Probabilities",
        col.names = paste('Group', 1:8),
        align = 'cc', digits = 3, booktabs = T, format.args = list(big.mark = ",")
    ) %>%
    kable_styling(latex_options = "HOLD_position")

# other optimization functions with constraints
# optim
out2 = optim(theta3.start, minus_joint_llh, method = "L-BFGS-B", df = data_g, lower = c(0,0,0), upper =
sum(out2$par)


# nloptr
library('nloptr')
# iterate over groups (temp group = 8)
data_g = data_temp %>% filter(group == 8)
# Objective Function
minimize_f = function(theta3) minus_joint_llh(theta3, data_g)

# Equality constraints
eval_g_eq = function(theta3) theta3[1] + theta3[2] + theta3[3] - 1

# Set optimization options.
local_opts <- list( "algorithm" = "NLOPT_LD_MMA", "xtol_rel" = 1.0e-6 )
opts <- list( "algorithm"= "NLOPT_LD_AUGLAG",
              "xtol_rel"= 1.0e-6,
              "maxeval"= 1000,
              "local_opts" = local_opts,
              "print_level" = 0 )

# Optimize
nloptr(x0 = theta3.start,
       eval_f = minimize_f,
       lb = c(0,0,0),
       ub = c(1,1,1),
       eval_g_eq = eval_g_eq,
       opts = opts
)

# mle
data_temp %>%
    filter(group == 1) %>%
    '$'(g) %>%
    tabulate()

    fit1 = mle(minus_joint_llh, theta3 = theta3.start,
         method = "L-BFGS-B", lower = c(0,0,0), upper = c(1,1,1),
         fixed = list(df = data_g))

nLL_wrapper = function(theta3 = theta3.start) {
    return(data %>%
```

```
        arrange(date) %>%
        group_by(group, bus_number) %>%
        distinct(mileage, .keep_all = T) %>%
        ungroup() %>%
        arrange(group, bus_number, date) %>%
        minus_joint_llh(theta3, data = ., group = 1)
    )
}

mle(nLL_wrapper, start = list(theta3 = theta3.start))


# Need to normalize all cost functions so c(0, theta1) = 0 (pg. 1015)
# Linear
c0 = function(xvec, theta1) sapply(xvec, function(x) 0.001*theta1[1]*x)
# Polynomial
c1 = function(xvec, theta1) sapply(xvec, function(x) theta1[1]*x + theta1[2]*x^2 + theta1[3]*x^3)
# exponential
c2 = function(xvec, theta1) sapply(xvec, function(x) theta1[1]*( exp(theta1[2]*x) - 1 ))
# hyperbolic
c3 = function(xvec, theta1) sapply(xvec, function(x) theta1[1]/(91-x) - theta1[1]/(91))
# square root
c3 = function(xvec, theta1) sapply(xvec, function(x) theta1[1]*x^0.5)

U = function(x, i, cost_fun, theta1, RC) {
    #' Returns utility
    #'
    #' @param cost_fun function, returning cost, with inputs x, theta1
    #' @param theta1 vector, matching the necessary dimensionality for cost_fun
    #' @param d integer, 0 or 1, represents no engine replacement or replacement
    #' @param RC float, expected cost of engine replacement
    #' @param e1 float, unobserved replacement cost shock
    #' @param e0 float, unobserved continuation (reported engine quality) shock
    ufun = function(x, i) {
        if (i == 1) { # replaced the engine
            return( -RC - cost_fun(0, theta1))
        }
        if (i == 0) { # did not replace the engine
            return( -cost_fun(x, theta1))
        }
    }
    return(ufun)
}
N = 90

theta3.fun = function(groups, theta3_df) {
    #' Return theta3 vector for group_ from theta3 dataframe
    df = theta3_df %>% filter(group %in% groups)
    theta3 = c(df$theta31, df$theta32)
    return(theta3)
}

P.fun = function(theta3) {
```

```
    #' Return an N x N markov transition matrix from theta3 probabilities
    Pmat <- matrix(0, nrow = N, ncol = N)
    for(col in 1:N){
        for (row in 1:N) {
            if (row == col & row < N) {  # Fill in the diagonal with theta31
                Pmat[row, col] = theta3[1]
            }
            if (row == col-1 & row < N-1) {  # Fill in the upper off-diagonal with theta32
                Pmat[row, col] = theta3[2]
            }
            if (row == col-2 & row < N-1) {  # Fill in the upper off-diagonal with theta32
                Pmat[row, col] = theta3[3]
            }
        }
    }
    # Some manual entry
    Pmat[N, N] = 1
    Pmat[N-1, N] = 1 - theta3[2]
    return(Pmat)
}

g.fun = function(y, theta3) {
    #' Return theta3 probability for value y (# of bins jumped this period)
    if (y==0) return(theta3[1])
    if (y==1) return(theta3[2])
    if (y==2) return(theta3[3])
    else print('WRONG VALUE OF y')
}

T.EV = function(P, c, beta, EV, RC) {
    #' Return new nx1 EV vector
    #' @param P nxn markov transition matrix (no engine replacement)
    #' @param c nx1 cost vector, given theta1
    #' @param beta discount factor
    #' @param EV old EV vector
    #' @param RC engine replacement cost
    # To avoid overflow (EV will become too negative, so the log will evaluate to 0,
    #   subtracting the minimum is equivalent and reduces the magnitidue of EV,
    #   see (3.20) in the Rust manual)
    K = max(-c + beta*EV)
    # discrete integral over transition probabilities (integral 3.20)
    return(
        P %*% log(
            exp( -c + beta*EV - K) +
            exp( -RC - c[1] + beta*EV[1] - K)
        ) + K
    )
}

pk.fun = function(c, beta, EV, RC) {
    #' Return new nx1 values for P(0|x, theta)
    #' @param c nx1 cost vector, given theta1
    #' @param beta discount factor
```

```
    #' @param EV old EV vector
    #' @param RC engine replacement cost
    return(
        1/(
            1 + exp(c - beta*EV - RC - c[1] + beta*EV[1])
        )
    )
}


EV.sequential = function(theta1, theta3, c, RC, P, beta,
                  iteration_max = 10000, verbose = F,
                  accuracy_threshold = 1e-6) {
    #' Return sequential approximation EV given theta1, theta3, cost function, and RC
    #' @param theta1 kx1 vector of theta1 parameters, k=# of params of cost_fun
    #' @param theta3 3x1 vector of theta3 transition probabilities
    #' @param cost_fun cost function that takes scalar x and theta1 kx1 vector
    #' @param RC scalar, engine replacement cost
    #' @param iteration_max integer, max number of iterations for approximating EV
    #' @param verbose bool, TRUE means print more information during iterations
    #' @param accuracy_threshold float, small number to get below when iterating
    #'      toward EV fixed point

    x = 1:N;  EVold = rep(0, N);
    start_t = proc.time()
    # norms = c()

    # Iterate toward fixed point EV
    for (k in 1:iteration_max) {
        # Get new EV
        EVnew = T.EV(P, c, beta, EVold, RC)
        # Calculate distance metric
        diff = abs(EVold - EVnew)
        sup_norm = max(diff)
        # print(paste('iteration', k, ' norm:', sup_norm))
        if (k%%100 == 0 & verbose) print(paste('iteration', k, ' norm:', sup_norm))
        if (sup_norm < accuracy_threshold) {
            if (verbose) {
                print(paste('WITHIN TOLERANCE! EV function approximated after', k, 'sequential iteratio
                print(paste('Total seconds:', proc.time()[3] - start_t[3]))
                print(paste('Average seconds per iteration:',
                            (proc.time()[3] - start_t[3]) / iteration_max ))
            }
            return(list(EVnew=EVnew, EVold=EVold))
        }
        EVold = EVnew
    }
    print(paste('MAX ITERATIONS REACHED!', iteration_max, 'EV function approximated to', sup_norm))
    print(paste('Total seconds:', proc.time()[3] - start_t[3]))
    print(paste('Average seconds per iteration:',
                (proc.time()[3] - start_t[3]) / iteration_max ))
    return(list(EVnew=EVnew, EVold=EVold))
    # Plot norm distance vs iterations, log-log scale
```

```
    # data.frame(y=norms) %>%
    #     mutate(x=row_number()) %>%
    #     ggplot() + geom_line(aes(x,y)) +
    #     xlab('iterations (log10 scale)') + ylab('Sup Norm (log10 scale)') +
    #     ggtitle('Sup Norm between EV and EV Sequential update') +
    #     scale_y_continuous(trans='log10') + scale_x_continuous(trans='log10')
}


Tprime.fun = function(beta, P, pk) {
    return(beta*(cbind(
        1 - rowSums( P[,2:N] * pk[2:N] ),  # firstcol
        P[,2:N] * pk[2:N]  # n-1 right cols
    )))
}


EV.nk = function(EVold, Tprime, c, dc, RC, P, beta, pk,
                 iteration_max = 1000, verbose = F,
                 accuracy_threshold = 1e-10, print_every=100) {
    #' Return Newton-Kantorovich approximated EV
    x = 1:N
    start_t = proc.time()
    # Iterate toward fixed point EV
    for (k in 1:iteration_max) {
        # Get new EV
        EVnew = EVold - solve(diag(N) - Tprime) %*% (EVold - T.EV(P, c, beta, EVold, RC))
        # Calculate distance metric
        sup_norm = max(abs(EVold - EVnew))
        if (k%%print_every == 0 & verbose) print(paste('iteration', k, ' norm:', sup_norm))
        if (sup_norm < accuracy_threshold) {
            if (verbose) {
                print(paste('WITHIN TOLERANCE! EV function approximated after', k, 'NK iterations'))
                print(paste('Total seconds:', proc.time()[3] - start_t[3]))
                print(paste('Average seconds per iteration:',
                            (proc.time()[3] - start_t[3]) / iteration_max ))
            }
            # Derivative of T w.r.t. beta (not needed, since we're not changing beta)
            # dTdb = (-beta * (1-beta) * (EVnew[1] + P %*% (EVnew - EVnew[1]) * pk))

            # Derivative of T w.r.t. RC
            dTdRC = P %*% pk - 1
            # Derivative of T w.r.t. theta1
            d = (P %*% dc)
            # quadratic
            # dTdtheta1 = -matrix(c((d[,1] * pk), (d[,2] * pk)), nrow=N, ncol=2)
            # linear
            dTdtheta1 = -(d * pk)
            # Derivative of T w.r.t. theta3
            K = max(-c + beta*EVnew)
            dtp = log(exp(-c + beta*EVnew - K) + exp(-RC - c[1] + beta*EVnew[1] - K)) + K
            dTdtheta3 = cbind(
                c(dtp[1:(N-2)] - dtp[3:N], dtp[(N-1)] - dtp[N], 0),
```

```
                c(dtp[2:(N-1)] - dtp[3:N], 0, 0)
            )

            # Bind all parameter derivative vectors together
            dTdtheta = cbind(dTdRC, dTdtheta1, dTdtheta3)
            # Derivative of EV w.r.t. theta
            dEV = solve(diag(N) - Tprime) %*% dTdtheta

            return(list(EVnew=EVnew, EVold=EVold, dEV=dEV))
        }
        EVold = EVnew
    }
    print(paste('MAX ITERATIONS REACHED!', iteration_max, 'EV function approximated to', sup_norm))
    print(paste('Total seconds:', proc.time()[3] - start_t[3]))
    print(paste('Average seconds per iteration:',
                (proc.time()[3] - start_t[3]) / iteration_max ))
    return(list(EVnew=EVnew, EVold=EVold))
}


EV.fun = function(theta1 = c(3),
                  theta3 = theta3.fun(groups = c(1), theta3_df),
                  beta = 0.9999,
                  RC = 10,
                  costfun = c0,
                  verbose = T,
                  verboseinner = F,
                  iteration_max = 1000) {
    x = 1:N
    c = costfun(x, theta1)
    theta3 = c(theta3, 1-sum(theta3))
    # Derivative of cost function w.r.t. theta1
    # dc_quad = matrix(c(exp(theta1[2]*x), theta1[1]*x*exp(theta1[2]*x)), nrow=N, ncol=2)
    dc_lin = c / theta1[1]
    # Generate the P matrix
    P = P.fun(theta3)
    # Get close to EV fixed point using sequential approximation
    EV_out = EV.sequential(theta1, theta3, c, RC, P, beta,
                           verbose=verboseinner, accuracy_threshold = 1e-3,
                           iteration_max = iteration_max*10)

    pk = pk.fun(c, beta, EV_out$EVnew, RC)
    Tprime = Tprime.fun(beta, P, pk)
    # Get really close to EV fixed point using NK method
    EVnew = EV.nk(EV_out$EVnew, Tprime, c, dc_lin, RC, P, beta, pk,
                  verbose = verbose, accuracy_threshold = 1e-6,
                  iteration_max = iteration_max)

    return(list(
        cost = c,
        dc = dc_lin,
        P = P,
        pk = pk,
```

```
        EV = EVnew$EVnew,
        dEV = EVnew$dEV
    ))

}



##############################################
#           BHHH method begins
##############################################
# Evaluates the loglikelihood function `ll`, its derivatives `dll`, and the
# information matrix (the cumulative outer product of the derivatives of the
# individual loglikelihood terms). Consider first the evaluation of the
# loglikelihood function. The general formula for the full and partial likelihood
# function is given in equation (2.5) and (2.8) in chapter 2. The R code
# (translated from GAUSS code) for these functions is given below:
# Calculating log-likelihood and derivative dLL
ll.fun = function(data, theta3, pk) {

    # Create T*M x 1 vector of P(0|x^m_t) for t in T and m in M (^1_1, ^1_2, ..., ^1_T, ^2_1, ...)
        # Take X bins from data, use to select indices of pk vector
    dt2 = data %>%
        filter(group < 5) %>%
        arrange(bus_number, m) %>%
        '$'(x5000)
    lp = pk[dt2]

    # Create 2 x 1 vector of logs of parameters
    # log of probabilities that (x_t - x_{t-1}) = 0 or = 1
    theta3 = c(theta3, 1-sum(theta3))
    lnp = log(theta3)

    # Create full log likelihood from some of partial log likelihoods
    dt3 = (data %>% filter(group<5) %>% arrange(bus_number, m))$g  # number of x bins increased from pr
    dt1 = (data %>% filter(group<5) %>% arrange(bus_number, m))$replaced  # d_t, action (0 = not replac
    ll = sum(lnp[dt3+1] +  # log[Pr(x_t | x_{t-1}, d_t, theta)]
        log(lp + (1-2*lp)*dt1))  # log[Pr(d_t | x_t, theta)]

    return(list(
        lp = lp,
        dt1 = dt1,
        dt2 = dt2,
        dt3 = dt3,
        lnp = lnp,
        ll = ll
    ))
}

# Hessian and derivative of LogLike
HdLL = function(EVout, llout) {
    dt1=llout$dt1; dt2=llout$dt2; lp=llout$lp;
```

```
    dev = EVout$dEV  # derivatives of EV w.r.t RC, theta1, theta31, theta32
    dc = EVout$dc  # derivatives of cost w.r.t theta1

    # Build derivatives of individual loglikelihood terms (d1v) from (3.29) in manual
    d1v = matrix(0, length(dt2), dim(dev)[2])
    d1v[,1] = 1
    d1v[,2] = dc[dt2]  # cost derivative evaluated at sample xbins
    d1v = d1v + t(replicate(length(dt2), dev[1,])) - dev[dt2,]  # dEV(0) - dEV(x) for x = sample xbins
    d1v = d1v * (lp - 1 + dt1)  # Pr(0|x_t) - 1 + d_t (replacement decision, 1=replaced)

    # Information matrix h from (3.30)
    H = t(d1v) %*% d1v

    dLL = colSums(d1v)

    return(list(H = H, dLL = dLL))
}


# BHHH algorithm from (3.26)
theta_search = function(theta1_start = c(3),
                        theta3_start = theta3.fun(groups = c(1), theta3_df),
                        beta = 0.9999,
                        RC_start = 10,
                        max_theta_iterations = 10,
                        accuracy_threshold = 1e-6,
                        lambda1 = 0.0001,
                        lambda0 = 0) {
    # starting values
    start_t = proc.time()
    df0 = NA
    EVout = EV.fun(theta1 = theta1_start,
                   theta3 = theta3_start,
                   beta = beta,
                   RC = RC_start)
    theta_new = c(RC_start, theta1_start, theta3_start[1:2])

    for (k in 1:max_theta_iterations) {

        llout = ll.fun(data, theta_new[3:4], EVout$pk)
        hdll = HdLL(EVout, llout)

        Dtheta = solve(hdll$H) %*% hdll$dLL
        df1 = t(hdll$dLL) %*% Dtheta
        if (is.na(df0)) df0 = 1.5*df1  # only in the first loop

        lambda2 = lambda1 - (lambda1 - lambda0)*df1/(df1 - df0)  # dfk = dlambda_k/dlambda
        lambda2 = lambda2[1,1]

        theta_old = theta_new
        theta_new = theta_old + lambda2*Dtheta  # Dtheta = D(theta_old)

        df0 = df1
```

```
        df1 = t(hdll$dLL) %*% Dtheta

        theta_norm = max(abs(theta_new - theta_old))
        print(paste('theta norm after', k, 'BHHH iterations:', theta_norm))

        print('theta:')
        print(theta_new)

        # Exit condition
        if (theta_norm < accuracy_threshold) {
            if (verbose) {
                print(paste('WITHIN TOLERANCE! EV function approximated after', k, 'sequential iteratio
                print(paste('Total seconds:', proc.time()[3] - start_t[3]))
                print(paste('Average seconds per iteration:',
                            (proc.time()[3] - start_t[3]) / iteration_max ))
                print('theta:')
                print(theta_new)
            }
            return(list(theta=theta_new, ll=llout$ll))
        }

        EVout = EV.fun(theta1 = theta_new[2],
                       theta3 = theta_new[3:4],
                       beta = beta,
                       RC = theta_new[1],
                       verbose = F,
                       verboseinner = F)

    }


    print(paste('MAX ITERATIONS REACHED!', max_theta_iterations, 'theta approximated to', theta_norm))
    print(paste('Total seconds:', proc.time()[3] - start_t[3]))
    print(paste('Average seconds per iteration:',
                (proc.time()[3] - start_t[3]) / max_theta_iterations ))
    print(theta_new)
    return(list(theta=theta_new, ll=llout$ll))

}


# Execute the search
theta = theta_search(theta1_start = c(3),
                         theta3_start = theta3.fun(groups = c(1), theta3_df),
                         max_theta_iterations = 100,
                         accuracy_threshold = 1e-6,
                         lambda1 = 0.0001,
                         lambda0 = 0)

# This starts heading toward a negative value for theta32. Haven't been able to
# find an easy way to bound the parameter search. Probably an issue with the step
# size or possibly the initial values of df0, df1, lambda0, and lambda1.
# I could not find docuemtnation on how to choose the right starting values.
```