14.273 Industrial Organization: Pset4

Dave Holtz, Jeremy Yang

May 18, 2017

1. Model setup.

   Following the notations in Rust (1987), HZ's flow utility is:

   $$u(x_t, i_t, \theta_1) + \epsilon_t(i_t) = \begin{cases} -RC - c(0, \theta_1) + \epsilon_t(1) & i_t = 1 \\ -c(x_t, \theta_1) + \epsilon_t(0) & i_t = 0 \end{cases}$$

   where RC is the replacement cost, x_t is the observed state variable for mileage, $c(\cdot)$ is cost function and i_t is the decision to replace engine and $\epsilon_t(\cdot)$ is action specific and type I extreme value distributed structural error (or unobserved state variable).

   The state transition probability is given by:

   $$\theta_{3j} = \mathbb{P}(x_{t+1} = x_t + j | x_t, i_t = 0)$$

   $j \in \{0, 1, 2\}$ and if $i_t = 1$ then $x_{t+1} = 0$ with probability 1.

   HZ chooses i_t in every period t to maximize an infinite sum of discounted flow utilities. The maximal value is defined as the value function (suppress the dependency on θ_1, θ_3):

   $$V(x_1, \epsilon_1) := \max_{i_t, t \in \{1,2,..\}} \mathbb{E}[\sum_{t=1}^{\infty} \beta^{t-1}(u(x_t, i_t, \theta_1) + \epsilon_t(i_t))]$$

   Rewrite the value function as in the Bellman optimality form:

   $$V(x_t, \epsilon_t) = \max_{i_t} \ (u(x_t, i_t, \theta_1) + \epsilon_t(i_t)) + \beta \mathbb{E}[V(x_{t+1}, \epsilon_{t+1}) | x_t, i_t]$$

   where the expectation is with respect to (conditional) state transition probability of both x and ϵ, see Rust (1987) equation (4.5). The Bellman equation breaks the dynamic optimization problem into an infinite series of static choices.

2. (1) The choice specific value function can be derived by plugging a specific action into the value function:

$$\tilde{V}(x_t, \epsilon_t, i_t) = \begin{cases} -RC - c(0, \theta_1) + \epsilon_t(1) + \beta\mathbb{E}[V(x_{t+1}, \epsilon_{t+1})|x_t, i_t = 1] \\ -c(x_t, \theta_1) + \epsilon_t(0) + \beta\mathbb{E}[V(x_{t+1}, \epsilon_{t+1})|x_t, i_t = 0] \end{cases}$$

$$V(x_t, \epsilon_t) = \max\{\tilde{V}(x_t, \epsilon_t, 1), \tilde{V}(x_t, \epsilon_t, 0)\}$$

HZ's decision is about trading off the total (future) cost of maintaining an old engine and the lump sum cost of replacing to a new one. The time to replace is the stopping time in this problem, so it can be thought as an optimal stopping time problem where the optimal policy is characterized by a cutoff in x, HZ would choose to replace the engine if x is above that threshold (the threshold depends on realized value of ϵ).

(2) It's clear from 2 (1) that the optimal stopping rule is:

$$\begin{aligned} & -RC - c(0, \theta_1) + \epsilon_t(1) + \beta\mathbb{E}[V(x_{t+1}, \epsilon_{t+1})|x_t, i_t = 1] > \\ & -c(x_t, \theta_1) + \epsilon_t(0) + \beta\mathbb{E}[V(x_{t+1}, \epsilon_{t+1})|x_t, i_t = 0] \end{aligned}$$

or,

$$\tilde{V}(x_t, \epsilon_t, 1) > \tilde{V}(x_t, \epsilon_t, 0)$$

therefore, because the errors are type I extreme value distributed:

$$\mathbb{P}(i_t = 1|x_t) = \frac{\exp(u(x_t, 1, \theta_1) + \beta\mathbb{E}[V_{t+1}|x_t, i_t = 1])}{\sum_{k=\{0,1\}} \exp(u(x_t, k, \theta_1) + \beta\mathbb{E}[V_{t+1}|x_t, i_t = k]} \tag{2.1}$$

where $u(x_t, i_t, \theta_1)$ is defined in 1 and for convenience:

$$V_{t+1} := V(x_{t+1}, \epsilon_{t+1})$$

(3) For discrete x, under the assumption that the errors are type I extreme value distributed, we have (Rust (1987) equation (4.14)):

$$EV(x, i) = \sum_y \log\{\sum_j \exp[u(y, j) + \beta EV(y, j)]\} \cdot p(y|x, i) \tag{2.2}$$

where

$$EV(x, i) := \mathbb{E}[V_{t+1}|x_t, i_t]$$

and x, i are the state and choice of current period and y, j are the state and choice of the next period. Also note that here the transition probability does not depend on x_t but only on j (or Δx). To compute expected value function, we first need to estimate transition probability from the data, this can be done simply by counting:

$$\hat{\theta}_{30} = \frac{\sum_b \sum_t 1_{\{x_{bt+1} - x_{bt} = 0, i_{bt} = 0\}}}{\sum_b \sum_t 1_{\{i_{bt} = 0\}}}$$

$$\hat{\theta}_{31} = \frac{\sum_b \sum_t 1_{\{x_{bt+1} - x_{bt} = 1, i_{bt} = 0\}}}{\sum_b \sum_t 1_{\{i_{bt} = 0\}}}$$

$$\hat{\theta}_{32} = \frac{\sum_b \sum_t 1_{\{x_{bt+1} - x_{bt} = 2, i_{bt} = 0\}}}{\sum_b \sum_t 1_{\{i_{bt} = 0\}}}$$

we compute the expected value function in the inner loop of the nested fixed point algorithm (holding the value of θ fixed), we first guess the initial values of $EV(x, i)$ for all possible values of x, i and use the equation (2.2) to iterate expected value function until it converges. The criterion is:

$$\max_{x,i} |EV^{T+1}(x, i) - EV^T(x, i)| < \eta$$

The plot for $x = 1 - 30$ at the true value of parameters are shown in Figure 1.

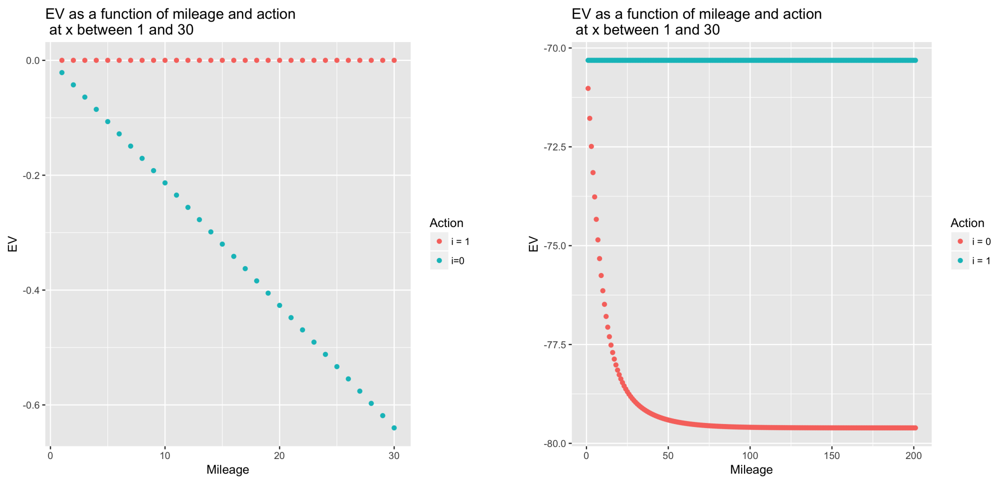

Figure 1: Expected Value Function for $i = 0$ and $i = 1$. Left panel shows results using iterative method, right panel shows provided Rust results.

Interestingly, our EV results are linear in mileage, which is probably not expected. Despite a good amount of debugging, we have been unable to identify a problem. However, it's also unclear how our calculated results for *just* 30 states should compare to the provided EV results, which provide information on 200 states. The first 30 states of the provided Rust EV estimates are decreasing in approximately linear fashion, suggesting our estimates might not be *so* bad. However, the order of magnitude of our EV values (e.g., 10^{-1}) is much smaller than the order of magnitude of EV values in the provided dataset (e.g., ~ 70), suggesting something is probably wrong. However, we don't have any more time to debug this, so we simply moved on.

(4) The provided dataset contains mileage and engine replacement information for 100 buses over 1,000 periods. The table below shows the mean mileage, maximum mileage, minimum mileage, standard deviation of the mileage, the average mileage at engine replacement across all buses and periods, and the

average number of engine replacements for a particular bus over the 1,000 periods.

avg miles	max miles	min miles	s.d. miles	avg replace miles	avg replacements
8.245	33.000	0.000	5.709	15.953	52.980

We might also be interested in understanding how each of these summary statistics vary across buses. For instance, maybe some buses have their engines replaced much more often. In order to study this, Figure 2 shows the distributions of average mileage, maximum mileage, s.d. mileage, avg miles at replacement, and number of replacements across the 100 buses in the sample. In general, these distributions are quite concentrated, suggesting that there are not systematic differences across buses.

3. (1) In the outer loop we search over a grid of values for (θ_1, β, RC), and compute the log likelihood function:

$$\log L = \sum_b \{\sum_t \log\mathbb{P}(i_{bt}|x_{bt}) + \sum_t \log\mathbb{P}(x_{bt}|x_{bt-1}, i_{t-1})\}$$

where b indexes for bus and t indexes for time period. We compute a log likelihood for each combination of values for (θ_1, β, RC) and choose the one that has the maximal value as our maximum likelihood estimation.

(2) In Hotz-Miller's approach, we will estimate the choice specific value function (as opposed to the expected value function as in Rust). We start by noting that conditional choice probability is observed directly from the data:

$$\hat{\mathbb{P}}(i=1|x) = \frac{\sum_b \sum_t 1_{\{i_{bt}=1, x_{bt}=x\}}}{\sum_b \sum_t 1_{\{x_{bt}=x\}}}$$

The choice-specific value function (minus the structural error, and suppressing the dependency on θ_1, θ_3) can be presented recursively in the following form:

$$\tilde{V}(x_t, i_t) = u(x_t, i_t) + \beta\mathbb{E}_{x_{t+1}}[\mathbb{E}_{i_{t+1}}[\mathbb{E}_{\epsilon_{t+1}}[u(x_{t+1}, i_{t+1}) + \epsilon_{t+1} + \beta(\cdots)|i_{t+1}, x_{t+1}]|x_{t+1}]|x_t, i_t]$$

where $(\cdots)$ represents higher (two and above) period forward expectations. In principle it's an infinite loop but in practice we need to stop at some T, for example, when $T = 2$, $(\cdots)$ simplifies to:

$$(\cdots) = \mathbb{E}_{x_{t+2}}[\mathbb{E}_{i_{t+2}}[\mathbb{E}_{\epsilon_{t+2}}[u(x_{t+2}, i_{t+2}) + \epsilon_{t+2}|i_{t+2}, x_{t+2}]|x_{t+2}]|x_{t+1}, i_{t+1}]$$

For simplicity, in the code we use one-period forward simulation where:

$$\tilde{V}(x_t, i_t) = u(x_t, i_t) + \beta\mathbb{E}_{x_{t+1}}[\mathbb{E}_{i_{t+1}}[\mathbb{E}_{\epsilon_{t+1}}[u(x_{t+1}, i_{t+1}) + \epsilon_{t+1}|i_{t+1}, x_{t+1}]|x_{t+1}]|x_t, i_t]$$

it is estimated as:

$$\hat{\tilde{V}}(x_t, i_t) = \frac{1}{S}\sum_s [u(x_t, i_t) + \beta[u(x^s_{t+1}, i^s_{t+1}) + \gamma - \log(\hat{\mathbb{P}}(i^s_{t+1}|x^s_{t+1}))]]$$

where x^s_{t+1} is drawn from the transition probability $\hat{\theta}_{30}, \hat{\theta}_{31}, \hat{\theta}_{32}$, and i^s_{t+1} is drawn from $\hat{\mathbb{P}}(i|x)$, γ is the Euler's constant.

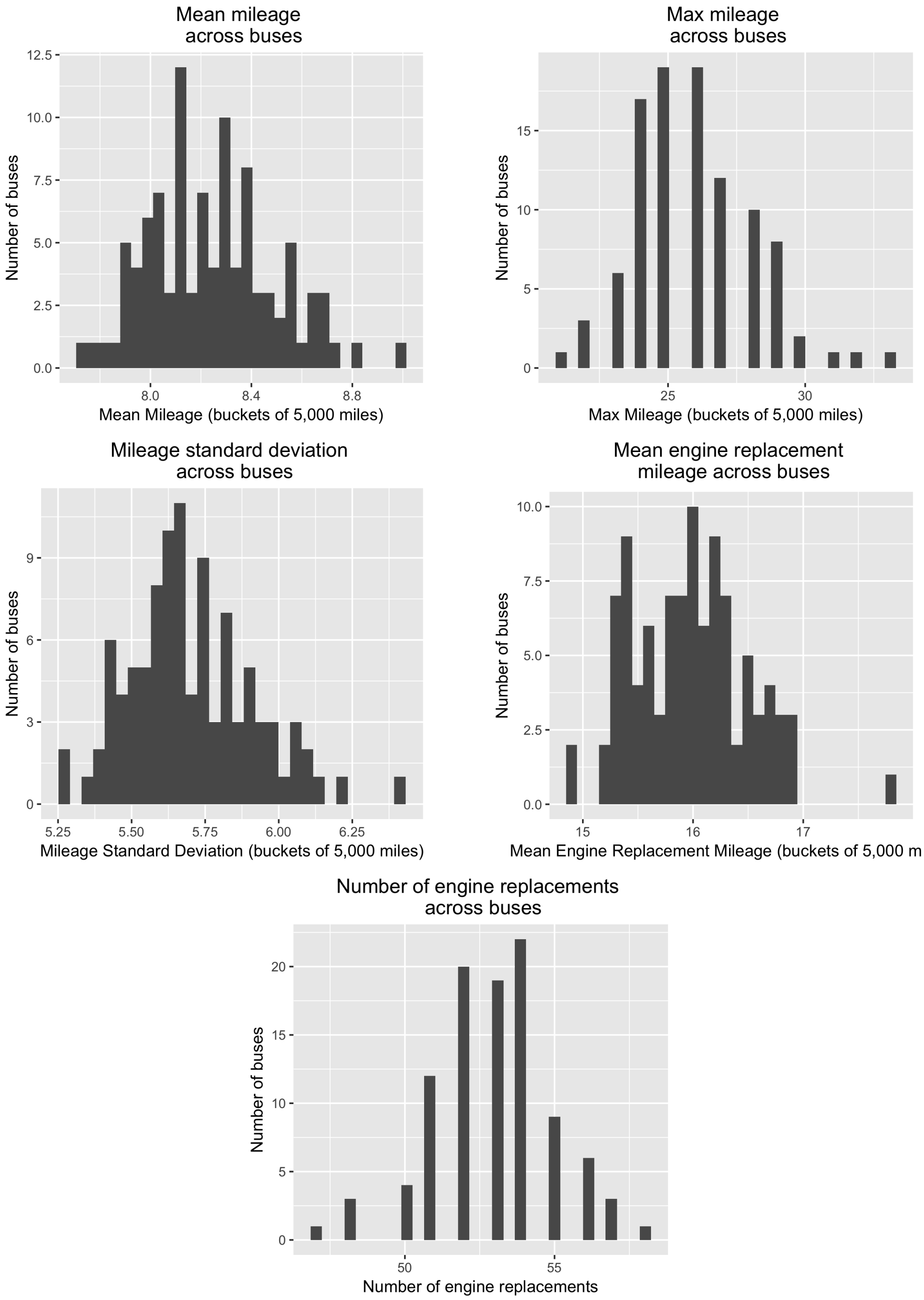

Figure 2: Distribution of various summary statistics across buses.

Appendix: Code

```
#### This code uses the methodologies of both Rust (1987) and Hotz and Miller (1993) to estimate the parameters of a single
#### agent dynamic problem where an agent (Harold Zurcher) must choose when to have the engines replaced in a fleet of buses.

## Import libraries
library(R.matlab)
library(ggplot2)
library(dplyr)

## Read in data
setwd('~/Dropbox (MIT)/MIT/Spring_2017/14.273/HW4/273-pset4/')
data <- readMat('../rust.mat')

## Extract bus replacement events
i <- data$it
## Extract bus mileage counts (in increments of 5,000 miles)
x <- data$xt


## Buses transition from different mileage states, and can jump forward zero, one, or two 5,000 mile buckets. This block
## of code estimates the transition probabilities empirically from the data.

### Initialize empty vectors to hold a count of how many times each jump happens
zero <- vector()
one <- vector()
two <- vector()

### Loop through the 100 buses in the dataset
for (k in 1:100) {

        ### Given a bus k, grab the mileage counts
        xk <- x[,k]
        ### Also grab the engine replacement events
        ik <- i[,k]
        ### Get a modified array which gives the change in mileage buckets from period j to period j+1
        jk <- xk[-1]-xk[-1000]

        ### We only care about periods where i=0 for transition probabilities, since i=1 will always send
        ### x back to 0. This selects out only time periods for this bus where i = 0
        j <- jk[ik==0]

        ### This counts up how many times the mileage bucket counter, x, moves up by 0, 1, or 2 when i=0
        zero[k] <- length(j[j==0])
```

```
        one[k] <- length(j[j==1])
        two[k] <- length(j[j==2])
}

## Estimate the x_t-independent transition probabilities by dividing the number of times for each transition by the
## total number of transitions
theta_30 = sum(zero)/(sum(zero)+sum(one)+sum(two))
theta_31 = sum(one)/(sum(zero)+sum(one)+sum(two))
theta_32 = sum(two)/(sum(zero)+sum(one)+sum(two))

#################
# Question 2.3 #
#################

#### We'll now take the true values of the parameter values as given, and use the method described in Rust (1987) to iteratively
#### estimate the value function (or in this case, the EV function).

## Initialize parameters to their true values
theta_1 = .05
theta_30 = .3
theta_31 = .5
theta_32 = .2
beta =.99
RC = 10

### Define the linear cost function. If an engine is not replaced, the bus incurs cost theta_1*x, so cost
### increases linearly as a bus gets older.
cost <- function(x){
        return (theta_1*x)}

### Define the utility function at mileage x from action i. If the agent chooses to replace the engine in a bus,
### it costs RC. If they choose _not_ to replace the engine, they incur the cost of running the bus at mileage x.
u <- function(x,i){
        -RC*i - cost(x*(1-i))
}


### The value function can be estimated through an iteration procedure. We start with some initial guess for EV,
### calculate EV with an expression that includes our initial guess of EV, and continue iterating until the difference
### between subsequent EV estimates becomes small.

#### value.Iterate is a function to iteratively update the value function according to the methodology in Rust. The function
#### takes as an argument a current estimate of EV, and returns an updated estimate of EV. EV is an x by d matrix - we want the
#### EV values for each decision d at every possible current mileage value x.
```

```r
value.Iterate <- function(EV){

  ### First iterate through each of the 30 x states
  for (x in 1:31){
    ## Update the EV value corresponding to not replacing the engine. There are three contributions here - one from the
    ## j = 0 case, one from the j = 1 case, and one from the j = 2 case. Note the indexing here. When x =1, the state is
    ## equal to 0 (this is the x that needs to be passed into u()), but we want to grab the EV corresponding to the 1st entry.
    EV2[x,1] <- log(exp(u(x-1,0)+beta*EV[x,1])+exp(u(x-1,1)+beta*EV[x,2]))*theta_30
    + log(exp(u(x,0)+beta*EV[x+1,1])+exp(u(x,1)+beta*EV[x+1,2]))*theta_31
    + log(exp(u(x+1,0)+beta*EV[x+2,1])+exp(u(x+1,1)+beta*EV[x+2,2]))*theta_32

    ## Update the EV value corresponding to replacing the engine. When the engine is replaced, x at the next period will
    ## deterministically reset to x = 0.
    EV2[x,2] <- log(exp(u(0,0)+beta*EV[1,1])+exp(u(0,1)+beta*EV[1,2]))
  }

  ## Return the updated EV values.
  return(EV2)
}

### Set a critical value for to measure the deviation between iterative updates of EV. The distance between the two EV matrices
### is the infinity norm of the difference
cri <- 10^(-8)

### Set an initial value for the EV matrix (all 0s, EV), and another EV object to hold the updated estimates, EV2.
EV <- matrix(100,33,2)
EV2 <- matrix(0,33,2)

## While the infinity norm is less than the threshold, iterate
while(max(abs(EV-EV2))>cri){

  ### Set the current EV to the previous updated EV
  EV <- EV2
  ### Compute a new updated EV by iterating on the current EV
  EV2 <- value.Iterate(EV)
}

### Do one last update to set EV equal to the last EV2
EV <- EV2

# get EV(x,i) for x=0,1,2,..,30
### EV contains extra states, which we needed to compute the above computation. Throw them away.
EV <- EV[1:31,]

```

```r
### Plot the EV of both replacing the engine (i = 1) and not replacing the
    engine (i = 0) at every x
### between 1 and 30
df <- data.frame('x'=c(1:30, 1:30),'EV'=c(EV[2:31,1], EV[2:31,2]),'Action'
    = c(rep('i = 0', 30), rep('i = 1', 30)))

### Generate a plot that compares the EV of replacing the engine and not
    replacing the engine
ev_plot <- ggplot(df, aes(x=x, y=EV, color=Action)) + geom_point() + xlab('
    Mileage') + ylab('EV') +
  ggtitle('EV as a function of mileage and action \n at x between 1 and 30'
      ) +
  theme(plot.title = element_text(hjust = 0.5))

### This is a plot to see the EV data in the attached rust matlab file. The
     state space is different than ours (200 states),
### so its hard to compare. Our's is linear (seems wrong), whereas the
    provided data is not. However, the first 30 states
### _do_ look approximately linear, so maybe we're not so far off.

df_rust <- data.frame('x'=c(seq(1,201), seq(1, 201)), 'EV'=c(data$EV[,1],
    data$EV[,2]), 'Action' = c(rep('i = 0', 201),


ev_plot_rust <- ggplot(df_rust, aes(x=x, y=EV, color=Action)) + geom_point
    () + xlab('Mileage') + ylab('EV') +
  ggtitle('Rust dataset EV as a function of mileage and action \n at x
      between 1 and 201') +
  theme(plot.title = element_text(hjust = 0.5))

################
# Question 2.4 #
################

### Calculate the mean mileage, mean time to engine replacement, max
    mileage, min mileage, and sd mileage over the whole sample
mean_x <- mean(x)
mean_engine_replacement_age <- mean(x[i == 1])
max_x <- max(x)
min_x <- min(x)
sd_x <- sd(x)
```

```
avg_replacements <- mean(apply(i, 2, function(x) {sum(x)}))

aggregate_stats <- c(mean_x, max_x, min_x, sd_x, mean_engine_)
aggregate_stats %>%
  round(., 3) %>%
  kable(., format='latex')

### Calculate the per bus mean mileage, mean time to engine replacement,
    max mileage, min mileage, and sd mileage
mean_x_per_bus <- apply(x, 2, function(x) {mean(x)})
max_x_per_bus <- apply(x, 2, function(x) {max(x)})
min_x_per_bus <- apply(x, 2, function(x) {min(x)})
sd_x_per_bus <- apply(x, 2, function(x) {sd(x)})
mean_engine_replacement_per_bus <- apply(x*i, 2, function(x) {sum(x)/sum(x
    != 0)})
replacements <- apply(i, 2, function(x) {sum(x)})


### Collate per bus information into a dataframe
per_bus_statistics <- data.frame(bus = seq(1, 100, 1),
                                  mean_x_per_bus = mean_x_per_bus,
                                  max_x_per_bus = max_x_per_bus,
                                  min_x_per_bus = min_x_per_bus,
                                  sd_x_per_bus = sd_x_per_bus,
                                  mean_engine_replacement_per_bus = mean_
                                      engine_replacement_per_bus,
                               replacements = replacements)

### Create some plots
mean_mileage_plot <- ggplot(per_bus_statistics, aes(x=mean_x_per_bus)) +
    geom_histogram() +
  xlab('Mean Mileage (buckets of 5,000 miles)') + ylab('Number of buses') +
      ggtitle('Mean mileage across buses') +
  theme(plot.title = element_text(hjust = 0.5))
ggsave(mean_mileage_plot, file='mean_mileage_plot.png', height=4, width=4,
    units='in')

max_mileage_plot <- ggplot(per_bus_statistics, aes(x=max_x_per_bus)) + geom
    _histogram() +
  xlab('Max Mileage (buckets of 5,000 miles)') + ylab('Number of buses') +
      ggtitle('Max mileage across buses') +
  theme(plot.title = element_text(hjust = 0.5))
ggsave(max_mileage_plot, file='max_mileage_plot.png', height=4, width=4,
    units='in')

sd_mileage_plot <- ggplot(per_bus_statistics, aes(x=sd_x_per_bus)) + geom_
    histogram() +
  xlab('Mileage Standard Deviation (buckets of 5,000 miles)') + ylab('
      Number of buses') +
  ggtitle('Mileage standard deviation across buses') +
  theme(plot.title = element_text(hjust = 0.5))
ggsave(sd_mileage_plot, file='sd_mileage_plot.png', height=4, width=4,
    units='in')

```

```
time_to_engine_replacement_plot <- ggplot(per_bus_statistics, aes(x=mean_engine_replacement_per_bus)) + geom_histogram() +
  xlab('Mean Engine Replacement Mileage (buckets of 5,000 miles)') + ylab('Number of buses') +
  ggtitle('Mean engine replacement mileage across buses') +
  theme(plot.title = element_text(hjust = 0.5))
ggsave(time_to_engine_replacement_plot, file='time_to_engine_replacement_plot.png', height=4, width=4, units='in')

replacements_plot <- ggplot(per_bus_statistics, aes(x=replacements)) + geom_histogram() +
  xlab('Number of engine replacements') + ylab('Number of buses') +
  ggtitle('Number of engine replacements across buses') +
  theme(plot.title = element_text(hjust = 0.5))
ggsave(replacements_plot, file='replacements_plot.png', height=4, width=4, units='in')


################
# Question 3.1 #
################

### This code will estimate the parameters beta, theta_1, and RC using the nested fixed-point algorithm
### described in Rust.

### A function to compute the probability of Zurcher's choices using the EVs we calculate using the EV calculation
### framework above. The probability of choosing different actions basically acts like a multichoice logit function.
choice.prob.Estimate <- function(){

  ### Initialize an empty matrix to hold choice probability estimates.
  p_i <- matrix(0,31,2)

  ### Iterate through all of the possible mileage states, x.
  for (x in 1:31){
    ## For each mileage state x, calculate the probability that Zurcher will choose i = 0.
    p_i[x,1] <- exp(u(x-1,0)+beta*EV[x,1])/(exp(u(x-1,0)+beta*EV[x,1])+exp(u(x-1,1)+beta*EV[x,2]))
    ## Calculate the probability of choosing i = 1 at each state, which is just 1 - P(i = 0).
    p_i[x,2] <- 1- p_i[x,1]
  }

  ### Return the updated p_i object
  return(p_i)
}

### A function to calculate the total log likelihood of the observed data given a set of parameters. This method assumes that
### the probabilities across periods and buses are independent, so we can just add up all of the log probabilities.
```

```r
log.likelihood.Compute <- function() {

  ### Initialize 0-valued variables to hold the log choice probability, the log transition probability,
  ### and the sum of the two.
  log_choice_prob <- 0
  log_transition_prob <- 0
  total <- 0

  ### Iterate over buses
  for (bus in 1:100) {
    ### Iterate over time periods
    for (t in 1:999) {
      ### We special case mileage states greater than 30, since they are a bit strange in our data. Otherwise, we calculate
      ### the choice probability using the current value of p_i according to the EV values we calculated to get the
      ### choice probability. Take the log and add it to the current running value.
      if (x[t,bus] <= 30){
        log_choice_prob <- log(p_i[x[t,bus]+1,i[t,bus]+1]) + log_choice_prob
      ### Do the same thing for our special cased, x > 30 case.
      } else {
        log_choice_prob <- log(p_i[31,i[t,bus]+1]) + log_choice_prob
      }

      ### Calculate over the transitions for each bus the sum of the log transition probabilities. We have our estimates of
      ### theta_3 given the empirical transition probabilities. So we can just grab that for each observed transition and add it
      ### to the total log transition probability.

      ### First we do the j = 0 case.
      if (x[t+1,bus]-x[t,bus]==0) {
        log_transition_prob <- log(theta_30)+log_transition_prob
      ### Then the j = 1 case.
      } else if (x[t+1,bus]-x[t,bus]==1) {
        log_transition_prob <- log(theta_31)+log_transition_prob
      ### And finally the j = 2 case.
      } else if (x[t+1,bus]-x[t,bus]==2) {
        log_transition_prob <- log(theta_32)+log_transition_prob
      }
    }

    ### Now, get the total log likelihood by adding up all of the transition components and the choice components.
    total <- (log_choice_prob+log_transition_prob) + total
  }
  return (total)
}

### Now we're actually going to use the nested fixed point algorithm to get the maximum likelihood estimates of the parameters
```

```r
### that we care about. This process has three steps.

### Step 1: We would calculate theta_30, theta_31, and theta_31 directly from the data. This step is not in the loop, and we've
### actually already done this and it doesn't change, so we don't need to do it again.

### Step 2: Next, we are going to set up a grid over values of theta_1, beta, and RC that we will calculate the
### log likelihood to determine the maximum likelihood parameter values. We'll also initialize a dataframe
### to hold the parameter values and the log likelihoods.

theta_1_range <- seq(.01,.10,.01)
beta_range <- seq(.90,.99,.01)
RC_range <- seq(6,15,1)
likelihood <- data.frame('theta_1'=rep(0),'beta'=rep(0),'RC'=rep(0),'log.likelihood'=rep(0))

### Step 3: Now we actually do the nested fixed point computation.

### Loop through theta_1
for (theta_1 in theta_1_range) {
  ### Loop through beta
  for (beta in beta_range) {
    ### Loop through RC
    for (RC in RC_range) {
      print(paste(c(theta_1, beta, RC), collapse=' '))

      ### Initialize the EV functions to the initial values we used above.
      EV <- matrix(100,33,2)
      EV2 <- matrix(0,33,2)

      ### Iteratively compute the EV values.
      while(max(abs(EV-EV2))>cri){
        EV <- EV2
        EV2 <- value.Iterate(EV)
      }

      EV <- EV2
      EV <- EV[1:31,]

      ### Given these values of EV, calculated the choice probabilities
      p_i <- choice.prob.Estimate()

      ### Given the EV values, the choice probabilities and the parameters, calculate
      ### the log-likelihood of the data.
      likelihood <- rbind(likelihood,c(theta_1,beta,RC,log.likelihood.Compute()))
    }
  }
}
```

```
### Retrieve the row in the likelihood dataframe corresponding to the maximum likelihood estimate
likelihood <- likelihood[-1,]
parameter_estimates <- likelihood[which.max(likelihood[,4]),]

### Use these parameters and get the relevant estimate of EV and p_i
theta_1 = parameter_estimates$theta_1
beta = parameter_estimates$beta
RC = parameter_estimates$RC

EV <- matrix(100,33,2)
EV2 <- matrix(0,33,2)
### Iteratively compute the EV values.
while(max(abs(EV-EV2))>cri){
  EV <- EV2
  EV2 <- value.Iterate(EV)
}
EV <- EV2
EV <- EV[1:31,]
### Given these values of EV, calculated the choice probabilities
p_i <- choice.prob.Estimate()
### Given the EV values, the choice probabilities and the parameters, calculate
### the log-likelihood of the data.
likelihood <- rbind(likelihood,c(theta_1,beta,RC,log.likelihood.Compute()))

save(EV, p_i, likelihood, parameter_estimates, file='rust_estimate.Rdata')

##################
# Question 3.2 #
##################

### Now we wil get estimates of the parameters using the Hotz and Miller conditional choice probability approach. This will
### allow us to compare these parameter estimates to those obtained using the Rust approach.

### First, we need to calculate the probability of the agent choosing either i = 0 or i = 1 based on the state that they find
### a given bus in, x, at some time period t. This will be the baseline that we use to try and find the best parameter values
### (i.e., which parameter values minimize the infinity norm between these true probabilities and the estimated probabilities)

### The probability matrix
p_ix <- matrix(0,33,2)
### The vector of how often the agent chooses i=1 given state x
ones <- vector()
### The vector of how often the agent finds a bus in state x
total <- vector()

### Loop through the states
for (state in 0:32){

```

```
  ### For a given state, a will track how many times i = 1 and b will track how many times that state occurs.
  ### Initialize them to 0 for the given state.
  a <- 0
  b <- 0

  ### Loop over the buses
  for (bus in 1:100){

    ### Increment how many times the agent chooses i = 1 in state x
    a <- sum(i[which(x[,bus]==state),bus]) + a
    ### Increment how many times the state x occurs
    b <- length(i[which(x[,bus]==state),bus]) + b
  }

  ### Add the most recent estimates to the vector.
  ones[state+1] <- a
  total[state+1] <- b
}

### Based on the ones and total vectors, updated the choice probability matrix.
p_ix[,1] <- 1-ones/total
p_ix[,2] <- ones/total

### The function below uses the Hotz and Miller method to estimate V and p_ix_hat for every state and period
### given a set of model parameters (beta, theta_1, and RC).
approximate.V_pixhat <- function() {
  ### Initialize an empty valuation matrix
  V <- matrix(0,33,2)
  ### Initialize an empty conditional choice probability matrix
  p_ix_hat <- matrix(0,33,2)

  ### Iterate through the states
  for (state in 0:30){
    ### Initialize a and b, which will basically track a running total of V for different choices over simulations, to 0.
    a = 0
    b = 0
    ### Iterate through the simulations. Note that ideal we would probably want to go more than one time step into the
    ### future. However, because of the limitations in our dataset, we only go one time step forward. This is mainly because
    ### it's unclear how we would draw i (the choice) for states that do not appear in our data (i.e., x = 34).
    for (s in 1:S){
      ## Conditional on choosing i = 0, simulate the next state that a given bus will end up in by drawing from the
      ## transition probabilities.
      x_prime_0 = state + sample(c(0,1,2),1,replace = T, prob = c(theta_30,theta_31,theta_32))
      ## Conditional on choosing i = 0 and ending up in some state in the next time period, randomly simulate a draw from
```

```r
      ## i based on the conditional choice probabilities
      i_prime_0 = sample(c(0,1),1,replace=T,prob = c(p_ix[x_prime_0+1,1],p_ix[x_prime_0+1,2]))
      # Figure out the expected utility from this truncated sequence of choices.
      a = (u(state,0) + beta*(u(x_prime_0,i_prime_0)+gamma-log(p_ix[x_prime_0+1,i_prime_0+1]))) + a

      ## Conditional on choosing i = 1, we don't need to simulate the next state that a bus will end up in. It will always
      ## be x = 0. So we jump right to simulating the draw from i for x = 0.
      i_prime_1 = sample(c(0,1),1,replace=T,prob = c(p_ix[1,1],p_ix[1,2]))
      ## Figure out the expected utility from this truncated sequence of choices.
      b = (u(state,1) + beta*(u(0,i_prime_1)+gamma-log(p_ix[1,i_prime_1+1]))) + b
    }

    ## Set the value of V to be the average over all S of our simulations for both the i = 0 and i = 1 choices.
    V[state+1,1] = a/S
    V[state+1,2] = b/S
    ## Use the multinomial logit-esque probability expression to figure out the probability of choosing i = 0 or i = 1
    ## given that the bus is in state x.
    p_ix_hat[state+1,1] <- exp(V[state+1,1])/(exp(V[state+1,1])+exp(V[state+1,2]))
    p_ix_hat[state+1,2] <- 1- p_ix_hat[state+1,1]
  }

  # Put final output into a list and return it
  results <- list('V' = V, 'p_ix_hat' = p_ix_hat)
  return(results)
}

### Specify a number of constants that will be used in the Hotz and Miller algorithm:
### S: The number of "simulations" to do per state / decision
### gamma: This should be Euler's constant
### theta_1_range: The range of theta_1 values to test
### beta_range: The range of beta_values to test
### RC_range: The range of RC values to test
S = 1000
gamma = .577
theta_1_range <- seq(.01,.10,.01)
beta_range <- seq(.90,.99,.01)
RC_range <- seq(6,15,1)

### Initialize a dataframe to hold different parameter combinations and the infinity-norm between the actual conditional
### choice probabilities and the estimated ones
difference <- data.frame('theta_1'=rep(0),'beta'=rep(0),'RC'=rep(0),'difference'=rep(0))
```

```
### Loop through theta_1
for (theta_1 in theta_1_range) {
  ### Loop through theta_2
  for (beta in beta_range) {
    ### Loop through RC
    for (RC in RC_range) {
      # Check progress
      print(paste(c(theta_1, beta, RC), collapse=' '))

      ### Get estimates of V and P_ix_hat using the Hotz and Miller method
      v_and_p_ix_hat <- approximate.V_pixhat()
      V = v_and_p_ix_hat$V
      p_ix_hat <- v_and_p_ix_hat$p_ix_hat

      ### Now that we have a full conditional choice probability matrix, calculate the infinity norm (i.e., largest
      ### absolute difference between the empirical conditional choice probabilities and those estimated with the
      ### given parameters)
      difference <- rbind(difference,c(theta_1,beta,RC,max(abs(p_ix[1:31,]-p_ix_hat[1:31,]))))
    }
  }
}

### Find the set of parameters that minimizes this difference
difference <- difference[-1,]
parameter_estimates <- difference[which.min(difference[,4]),]

### Use these parameters and get the relevant estimate of V and p_ix_hat
theta_1 = parameter_estimates$theta_1
beta = parameter_estimates$beta
RC = parameter_estimates$RC
best_guesses <- approximate.V_pixhat()
V <- best_guesses$V
p_ix_hat <- best_guesses$p_ix_hat

save(V, p_ix_hat, difference, parameter_estimates, file='hotz_and_miller_estimate.Rdata')

################
# Question 3.3 #
################



################
# Question 3.4 #
################

### This function simulates, for one agent, a sequence of state transitions and also engine replacement decisions
simulate_sequence <- function(n_periods) {
```

```
  ### Initialize empty vectors to hold states and engine replacement transitions
  x_values <- rep(0, n_periods)
  i_values <- rep(0, n_periods)
  ### Every bus starts at state 0
  x_values[1] <- 0
  ### Go through the progression
  for (j in 1:length(x_values)) {
    ### Make a decision based on current state
    i_values[j] = sample(c(0,1),1,replace=T,prob = c(p_ix_hat[x_values[j] + 1,1],p_ix_hat[x_values[j] + 1,2]))
    ### If decision is to not replace, continue on and increment x randomly
    if (i_values[j] == 0) {
      x_values[j+1] = x_values[j] + sample(c(0,1,2),1,replace = T, prob = c(theta_30,theta_31,theta_32))
    ### If decision is to replace, reset state to 0
    } else {
      x_values[j+1] = 0
    }
  }
  ## Generate a decision for the last period, even though we never see the fruits of that decision
  i_values[length(i_values)] = sample(c(0,1),1,replace=T,prob = c(p_ix_hat[x_values[length(x_values)] + 1,1],
                                                                  p_ix_hat[x_values[length(x_values)] + 1,2]))
  ### Return the states and replacement decisions in a list
  results <- list('x_values' = x_values, 'i_values' = i_values)
  return(results)
}

### Given a set of parameters, this function generates period-by-period demand estimates for new buses (e.g.,
### how many buses will get their engine replaced in each period)
estimate_demand <- function(n_sims, n_buses, n_periods) {

  ### Initialize a vector to hold simulated demand
  simulated_demand_total <- rep(0, n_periods)

  ### Run a bunch of simulations and simulate engine replacement decisions
  for (j in 1:n_sims) {
    simulated_demand_total = simulated_demand_total + simulate_sequence(n_periods)$i_values
  }

```

```
  ### Divide by the number of sims to get averages, multiply by number of buses (this works because
  ###buses are independent). Then return what we get.
  return((n_buses/n_sims)*simulated_demand_total)
}

### Get demand as a function of RC for the first bus

## Specify the range of RCs, as well as constants.
RC_range = seq(0, 15, .25)
n_periods = 15
n_sims = 1000
n_buses = 100
load('hotz_and_miller_estimate.Rdata')

## Initialize an empty dataframe to hold results
estimated_demand_df <- data.frame(time_period = c(),
                                  RC = c(),
                                  demand = c(),
                                  engine = c())

# Loop through the RCs, then estimate the probabilities using the Rust method, then do simulation.
for (j in RC_range) {
  RC = j

  ### Set a critical value for to measure the deviation between iterative updates of EV. The distance between the two EV matrices
  ### is the infinity norm of the difference
  cri <- 10^(-8)

  ### Set an initial value for the EV matrix (all 0s, EV), and another EV object to hold the updated estimates, EV2.
  EV <- matrix(100,33,2)
  EV2 <- matrix(0,33,2)

  ## While the infinity norm is less than the threshold, iterate
  while(max(abs(EV-EV2))>cri){

    ### Set the current EV to the previous updated EV
    EV <- EV2
    ### Compute a new updated EV by iterating on the current EV
    EV2 <- value.Iterate(EV)
  }

  ### Do one last update to set EV equal to the last EV2
  EV <- EV2

  # get EV(x,i) for x=0,1,2,..,30
  ### EV contains extra states, which we needed to compute the above computation. Throw them away.
  EV <- EV[1:31,]

  ### Get estimated probability based on the above EV
```

```
  p_ix_hat <- choice.prob.Estimate()
  ### Estimate demand using that probability
  estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)

  ### Add this estimate to a temp dataframe
  estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
    ,
                                          RC = rep(RC, n_periods),
                                          demand = estimated_demand,
                                          engine = rep('Engine 1', n_periods))
  ### Collate temp dataframe to full dataframe
  estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
    temp)

}

### Reset theta_1 to the "new engine", redo the exercise above.
theta_1 = .02

### Loop through RCs
for (j in RC_range) {
  RC = j

  ### Set a critical value for to measure the deviation between iterative
      updates of EV. The distance between the two EV matrices
  ### is the infinity norm of the difference
  cri <- 10^(-8)

  ### Set an initial value for the EV matrix (all 0s, EV), and another EV
      object to hold the updated estimates, EV2.
  EV <- matrix(100,33,2)
  EV2 <- matrix(0,33,2)

  ## While the infinity norm is less than the threshold, iterate
  while(max(abs(EV-EV2))>cri){

    ### Set the current EV to the previous updated EV
    EV <- EV2
    ### Compute a new updated EV by iterating on the current EV
    EV2 <- value.Iterate(EV)
  }

  ### Do one last update to set EV equal to the last EV2
  EV <- EV2

  # get EV(x,i) for x=0,1,2,..,30
  ### EV contains extra states, which we needed to compute the above
      computation. Throw them away.
  EV <- EV[1:31,]

  ### Get probability estimates based on EV
  p_ix_hat <- choice.prob.Estimate()
  ### Estimate demand
  estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)
```

```
    ### Add to temp dataframe
    estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
        ,
                                           RC = rep(RC, n_periods),
                                           demand = estimated_demand,
                                           engine = rep('Engine 2', n_periods
                                               ))
    ### Collate to full dataframe
    estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
        temp)

}

### For a reduced set of RCs, see the period-by-period demand
estimated_demand_df %>%
  filter(RC %in% c(1, 3, 5, 7, 10)) %>%
ggplot(., aes(x=time_period, y=demand, color=as.factor(RC))) + geom_line()
    +
  facet_wrap(~engine)

### Aggregate over periods to get demand as a function of RC for different
    thetas.
estimated_demand_df %>%
  group_by(RC, engine) %>%
  summarise(total_demand = sum(demand)) %>%
  ungroup() %>%
  ggplot(., aes(x=RC, y=total_demand, color=engine)) + geom_line()


##################
# Question 3.5 #
##################
```

Listing 1: ./rust.R