
14.273 Industrial Organization: Problem Set 4

Dave Holtz, Jeremy Yang

May 18, 2017

1. Model setup.

Following the notations in Rust [2], 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):

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

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 [2] 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:

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

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 [2] equation (4.14)):

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

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's left panel. The provided EV values for the Rust dataset are provided in Figure 1's right panel. The two match well, suggesting that our implementation is working as expected. There is some strange behavior in the tail (for mileage $\in (29, 30)$). We suspect this is truncation error from not having data for higher states.

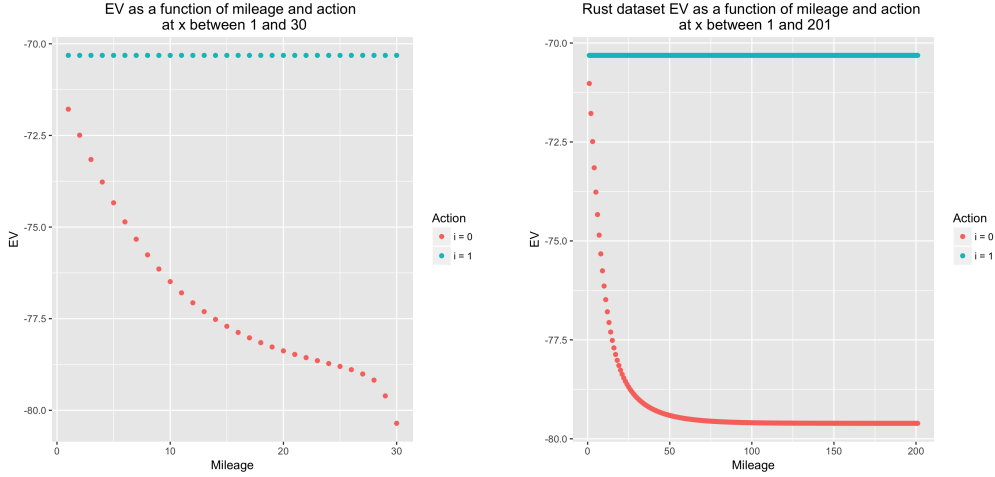


Figure 1: Expected Value Function for $i = 0$ and $i = 1$. Left panel shows results using iterative method, right panel shows provided Rust results.

- (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.

The final, bottom right plot in 2 also shows the empirically observed conditional choice probability as a function of state (mileage) that Harold Zurcher actually acts on. At a high level, Zurcher's has to make the investment decision of when to replace a given bus's engine. The mean replacement mileage plot suggests that on average he replaces a bus's engine after about 80,000 miles. The conditional choice probability plot suggests that the likelihood he increases the engine is practically zero until the bus hits 50,000 miles, after which the probability that the bus has its engine replaced climbs quickly. By the time a bus has 150,000 miles on it, it has a 50% probability of having its engine changed in a given time period.

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

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

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 set of parameters that maximizes the log-likelihood of the data. The maximum likelihood parameters obtained with the Rust [2] method are:

$$\begin{aligned} \theta_1 &= 0.05 \\ \beta &= 0.99 \\ RC &= 10 \end{aligned}$$

The maximum-likelihood parameters that we obtain using the Rust method are exactly identical to the "true" parameters.

- (2) In Hotz-Miller's [1] 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:

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

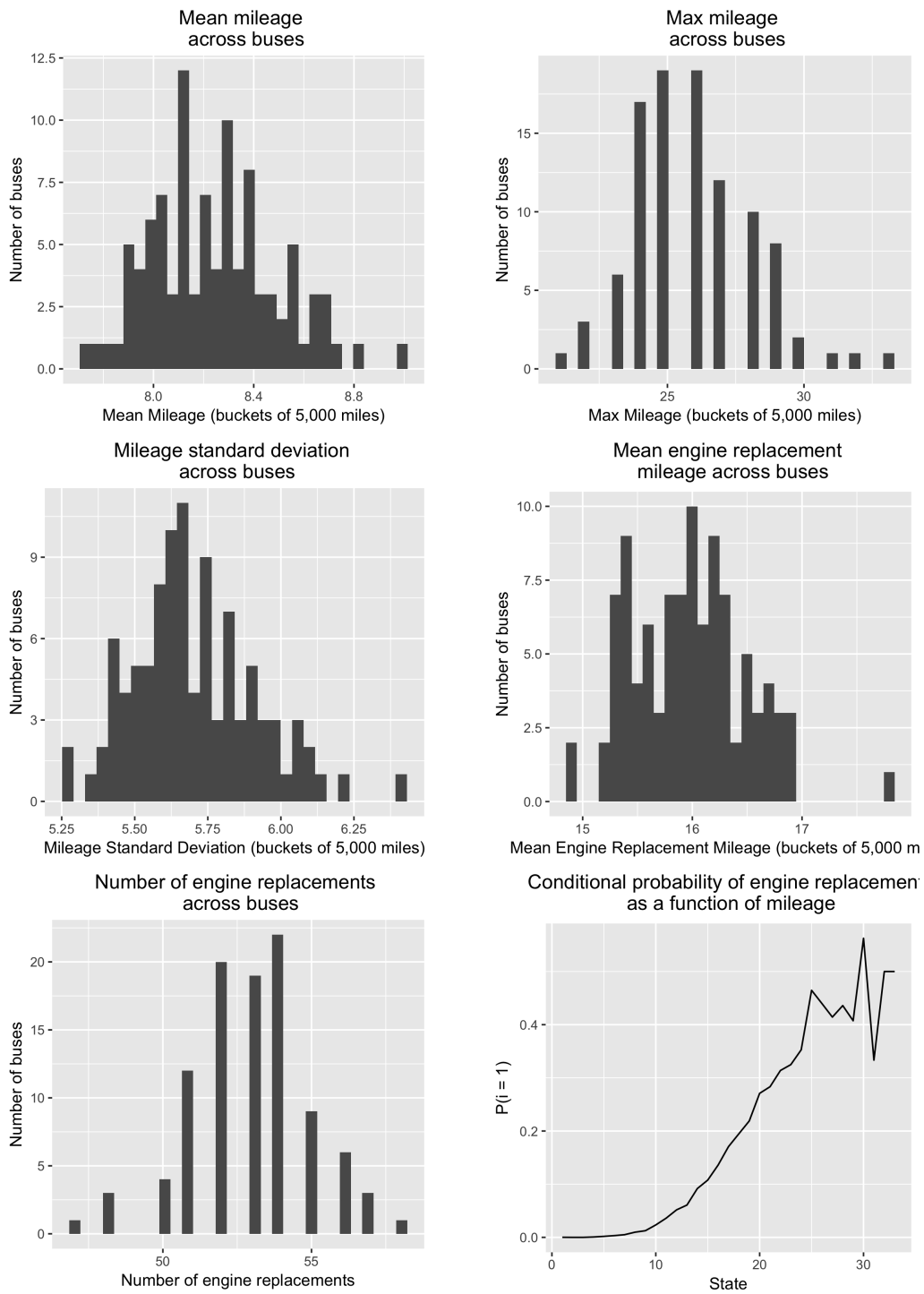


Figure 2: Distribution of various summary statistics across buses, as well as the empirical conditional choice probability for Zurcher.

where $(\cdot \cdot \cdot)$ 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 $T = 2$, $(\cdot \cdot \cdot)$ simplifies to:

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

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

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

it is estimated as:

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

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

We only go one period forward because we only observe data for states up to $x_t = 33$. It is possible for larger T that we would encounter a state that is not in our dataset. When this occurs, it's unclear what value should be used as the conditional choice probability. While we avoid this issue by setting $T = 2$, this does reduce the precision of our estimates.

The estimates we obtain using the Hotz and Miller [1] method are

$$\begin{aligned} \theta_1 &= 0.09 \\ \beta &= 0.92 \\ RC &= 6. \end{aligned}$$

We expect that these estimates are likelihood less precise than those obtained via the Rust [2] method (in this particular case) because we so severely truncate the sum in the Hotz and Miller [1] method. The Rust approach relies on parametric assumptions that in this particular case appear to be satisfied, which is why it outperforms Hotz and Miller. However, in general, Hotz and Miller may be a "safer" approach (particularly when as a practitioner you are able to include more higher order terms in your sums).

- (3) In order to determine which engine HZ prefers, we simply need to look at HZ's value function for both engines at $t = 0$ (which corresponds to $x_t = 0$ for all buses). There are a number of different mileage evolution paths that any given bus could take. However, the ex ante value function at time $= 0$ provides a weighted average of all of these scenarios. So at the outset, he will prefer whichever engine provides the most value in expectation. Given our estimation, HZ prefers the new engine over the old one, as

$$EV_{old}(x = 0, i = 1) = -71.026 < -65.214 = EV_{new}(x = 0, i = 1).$$

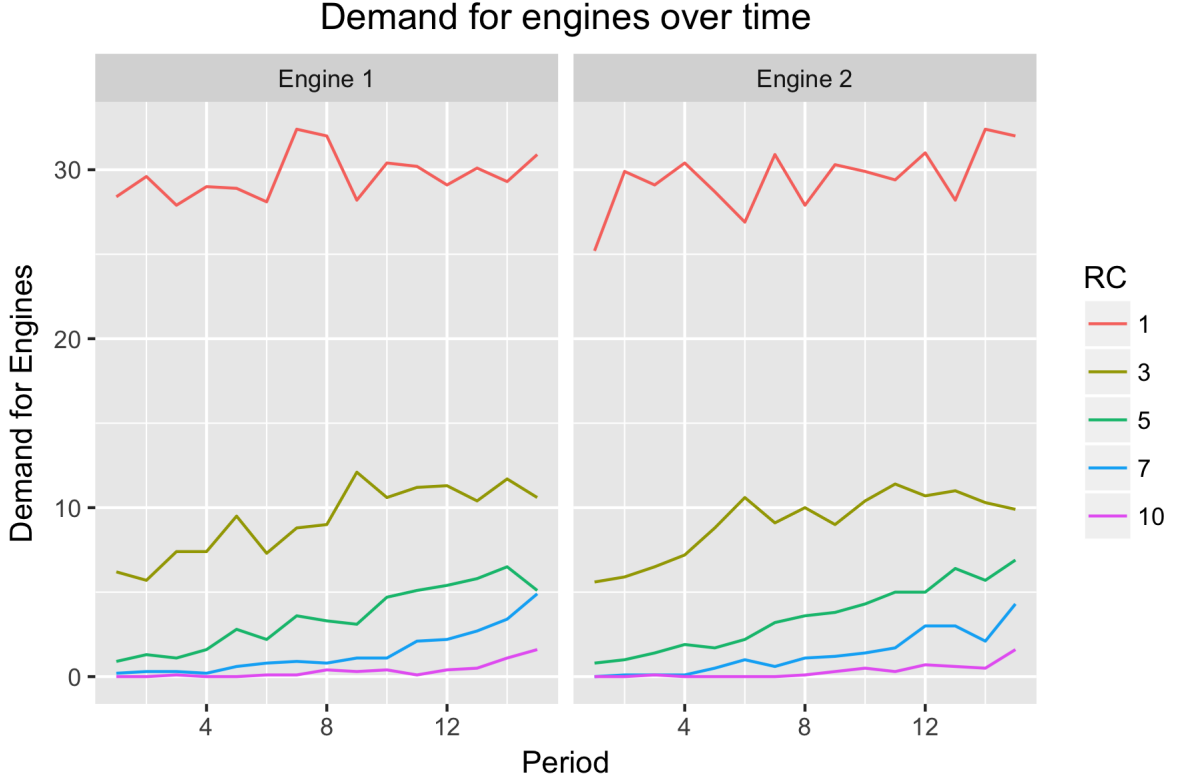


Figure 3: The demand for engines across a fleet of 100 buses as a function of period (over the first 15 periods) for different values of RC. Unsurprisingly, when RC is lower, HZ is much more willing to change bus engines.

- (4) We want to compute HZ’s demand function for the two buses, which we will denote as engine 1 ($\theta_1 = 0.05, RC = 10$) and engine 2 ($\theta_1 = 0.02, RC = 20$) as a function of RC. In order to do so, we obtain conditional choice probability estimates, $\hat{\mathbb{P}}(i = 1|x)$ by using the Rust [2] method to iterate EV values. We use the Rust methodology because the Hotz and Miller [1] methodology depends on the observed conditional choice probabilities, which we know do *not* correspond to the counterfactual engine 2.

With those conditional choice probability estimates for the two engines in hand, we run 1,000 simulations of a bus’s state transitions (and HZ’s corresponding engine replacement decisions) over the first 15 periods. This allows us to get an expected, per-bus demand for engines over the first 15 periods. In order to get the expected demand that HZ has for engines across all buses, we simply multiply this figure by 100. So the expected demand (as a function of period t) is:

$$D(t) = 100 \times \sum_{x_t} \hat{\mathbb{P}}(i = 1|x = x_t) \hat{\mathbb{P}}(x = x_t|t) \quad (1)$$

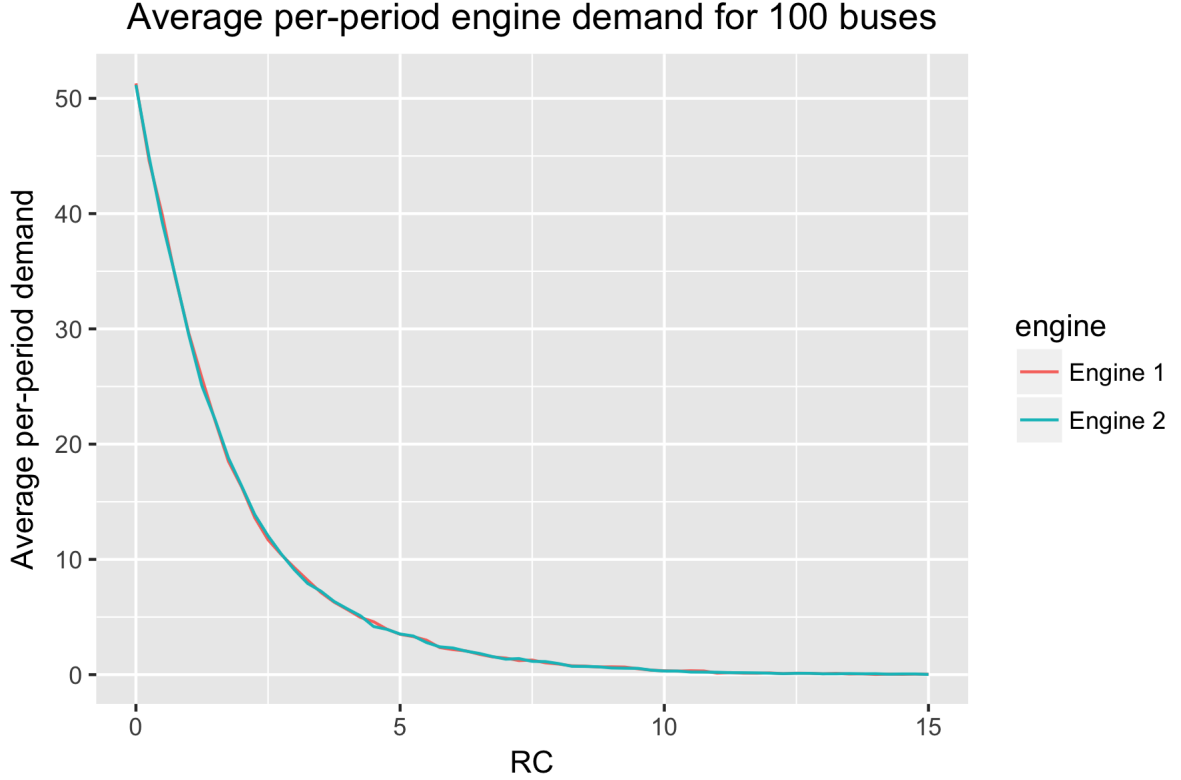


Figure 4: Aggregate per-period demand for new engines across a fleet of 100 buses as a function of RC . For engine 1, $\theta_1 = 0.05$. For engine 2, $\theta_1 = 0.02$.

HZ's demand for engines as a function of the period, t for a few values of RC can be found in Figure 3. The average per-period demand for the two engines (averaged across 15 periods) for different values of RC can be found in Figure 4. It's worth noting that the demand curves for the two engines appear almost identical - there are only small differences (on the order of a tenths of an engine) between the two. This could be a true difference, or it could be simulation error. Although we're not sure why these demand curves are so similar, we have two hypotheses:

1. Our Rust EV estimates are highly sensitive to initialization. It's possible our estimates for engine 2 are not correct.
 2. The increase in RC from engine 1 to engine 2 is almost perfectly offset by the decrease in θ_1 , creating two extremely similar demand curves.
- (5) To determine the total value of the engines, assuming marginal cost RC , we can simply compute the total area to the right of a given RC in a demand curve that looks like Figure 4. This area will give the total surplus that HZ gets from the engine in a given period. In order to get a total value, we simply multiply this by the number of periods we want to consider. Mathematically,

it is socially optimal to produce the more efficient engine if the total value is greater than the total cost:

$$V_{engine}(RC) - C(RC) = n \cdot \int_{RC}^{\infty} D_{engine}(p) dp - n \cdot RC \cdot D_{engine}(RC) - c > 0 \quad (2)$$

where $n = 15$ is the number of periods and $D(p)$ is the amount of demand that HZ would have a given RC and c is the fixed R&D cost. Let's assume that c is 0 for engine 1 (the status quo engine). Figure 5 shows the value produced by both engines as a function of RC , assuming that $c = 0$. We can see here that, similarly to demand, the value produced by the two engines is very similar.

Value produced by each engine as a function of RC ($c=0$)

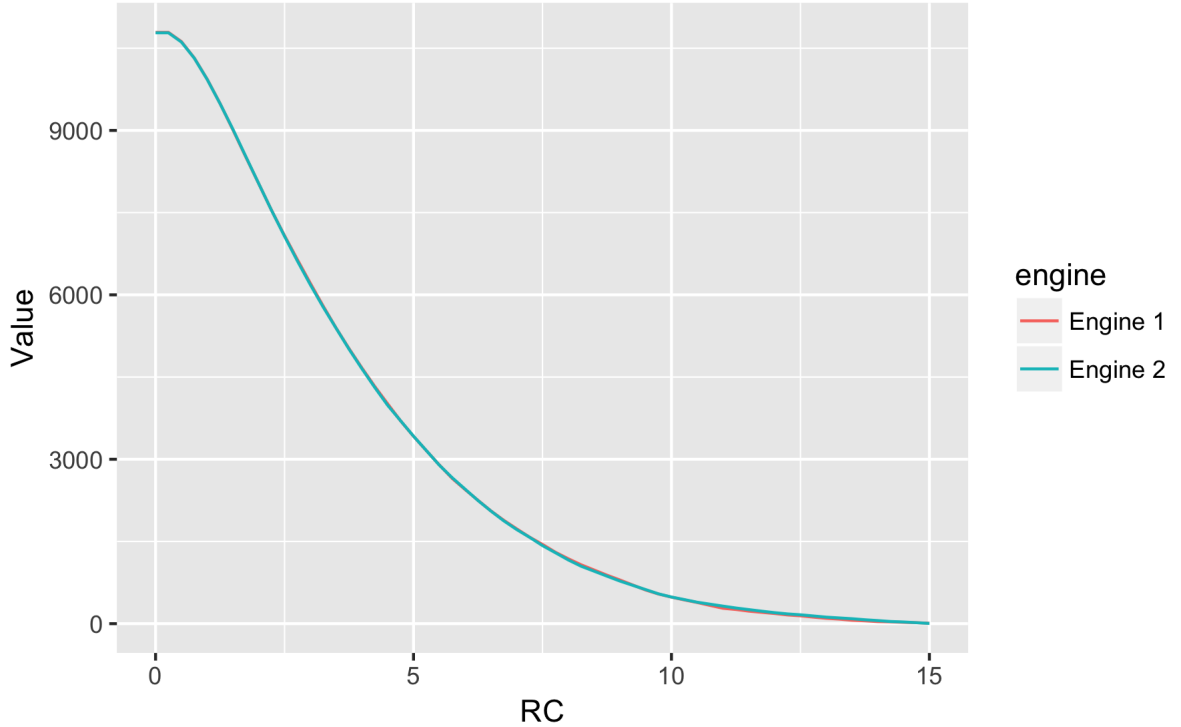


Figure 5: Value produced by the two different engines as a function of RC . We assume that $c = 0$ (i.e., R&D costs are zero) for both engines. For engine 1, $\theta_1 = 0.05$. For engine 2, $\theta_1 = 0.02$.

Theoretically, the maximum justifiable development would be $value_2 - value_1$, if that quantity were positive (otherwise it is zero). For engine 1 at $RC = 10$, the total value is 2,447.1. For engine 2 at $RC = 20$, the total value is 0 (because our estimated demand is zero at $RC = 20$). In light of this, there is no R&D cost at which development of engine 2 would be justified.

It's interesting that engine 2 appears to provide less value, even though according to our answer in (3.3), ZH prefers the new engine. This might suggest that 15 periods is not enough time to accurately assess the value produced by both of the engines.

APPENDIX: CODE

```
1 ##### This code uses the methodologies of both Rust (1987) and Hotz and
   Miller (1993) to estimate the parameters of a single
2 ##### agent dynamic problem where an agent (Harold Zurcher) must choose when
   to have the engines replaced in a fleet of buses.
3
4 ## Import libraries
5 library(R.matlab)
6 library(ggplot2)
7 library(dplyr)
8
9 ## Read in data
10 setwd('~ /Dropbox_ /MIT /MIT /Spring_2017 /14.273 /HW4 /273-pset4 /')
11 data <- readMat(' ../rust.mat')
12 gamma = .577
13
14 ## Extract bus replacement events
15 i <- data$it
16 ## Extract bus mileage counts (in increments of 5,000 miles)
17 x <- data$xt
18
19
20 ## Buses transition from different mileage states , and can jump forward
   zero , one , or two 5,000 mile buckets. This block
21 ## of code estimates the transition probabilities empirically from the data
   .
22
23 ##### Initialize empty vectors to hold a count of how many times each jump
   happens
24 zero <- vector()
25 one <- vector()
26 two <- vector()
27
28 ##### Loop through the 100 buses in the dataset
29 for (k in 1:100) {
30
31     ##### Given a bus k, grab the mileage counts
32     xk <- x[,k]
33     ##### Also grab the engine replacement events
34     ik <- i[,k]
35     ##### Get a modified array which gives the change in mileage buckets
       from period j to period j+1
36     jk <- xk[-1]-xk[-1000]
37
38     ##### We only care about periods where i=0 for transition
       probabilities , since i=1 will always send
39     ##### x back to 0. This selects out only time periods for this bus
       where i = 0
40     j <- jk[ik==0]
41
42     ##### This counts up how many times the mileage bucket counter , x ,
       moves up by 0, 1, or 2 when i=0
```

```

43     zero[k] <- length(j[j==0])
44     one[k] <- length(j[j==1])
45     two[k] <- length(j[j==2])
46 }
47
48 ## Estimate the x_t-independent transition probabilities by dividing the
49   number of times for each transition by the
50   ## total number of transitions
51 theta_30 = sum(zero)/(sum(zero)+sum(one)+sum(two))
52 theta_31 = sum(one)/(sum(zero)+sum(one)+sum(two))
53 theta_32 = sum(two)/(sum(zero)+sum(one)+sum(two))
54 #####
55 # Question 2.3 #
56 #####
57
58 ##### We'll now take the true values of the parameter values as given, and
59   use the method described in Rust (1987) to iteratively
60   ##### estimate the value function (or in this case, the EV function).
61
62 ## Initialize parameters to their true values
63 theta_1 = .05
64 theta_30 = .3
65 theta_31 = .5
66 theta_32 = .2
67 beta = .99
68 RC = 10
69
70 ##### Define the linear cost function. If an engine is not replaced, the bus
71   incurs cost theta_1*x, so cost
72   ##### increases linearly as a bus gets older.
73   cost <- function(x){
74     return (theta_1*x)}
75
76 ##### Define the utility function at mileage x from action i. If the agent
77   chooses to replace the engine in a bus,
78   ##### it costs RC. If they choose _not_ to replace the engine, they incur the
79   cost of running the bus at mileage x.
80   u <- function(x,i){
81     -RC*i - cost(x*(1-i))
82   }
83
84 ##### The value function can be estimated through an iteration procedure. We
85   start with some initial guess for EV,
86   ##### calculate EV with an expression that includes our initial guess of EV,
87   and continue iterating until the difference
88   ##### between subsequent EV estimates becomes small.
89
90 ##### value.Iterate is a function to iteratively update the value function
91   according to the methodology in Rust. The function
92   ##### takes as an argument a current estimate of EV, and returns an updated
93   estimate of EV. EV is an x by d matrix - we want the

```

```

87 ##### EV values for each decision d at every possible current mileage value
88     x.
89 value.Iterate <- function(EV){
90     ### First iterate through each of the 30 x states
91     for (x in 1:31){
92         ## Update the EV value corresponding to not replacing the engine. There
93         ## are three contributions here – one from the
94         ## j = 0 case, one from the j = 1 case, and one from the j = 2 case.
95         ## Note the indexing here. When x =1, the state is
96         ## equal to 0 (this is the x that needs to be passed into u()), but we
97         ## want to grab the EV corresponding to the 1st entry.
98         EV2[x,1] <- log(exp(u(x-1,0)+beta*EV[x,1])+exp(u(x-1,1)+beta*EV[x,2]))*
99             theta_30 +
100             log(exp(u(x,0)+beta*EV[x+1,1])+exp(u(x,1)+beta*EV[x+1,2]))*theta_31
101             +
102             log(exp(u(x+1,0)+beta*EV[x+2,1])+exp(u(x+1,1)+beta*EV[x+2,2]))*theta_
103             32
104         ## Update the EV value corresponding to replacing the engine. When the
105         ## engine is replaced, x at the next period will
106         ## deterministically reset to x = 0.
107         EV2[x,2] <- log(exp(u(0,0)+beta*EV[1,1])+exp(u(0,1)+beta*EV[1,2]))
108     }
109     ## Return the updated EV values.
110     return(EV2)
111 }
112
113 ##### Set a critical value for to measure the deviation between iterative
114 ##### updates of EV. The distance between the two EV matrices
115 ##### is the infinity norm of the difference
116 cri <- 10^(-8)
117
118 ##### Set an initial value for the EV matrix (all 0s, EV), and another EV
119 ##### object to hold the updated estimates, EV2.
120 EV <- matrix(0,33,2)
121 EV2 <- matrix(-80,33,2)
122
123 ## While the infinity norm is less than the threshold, iterate
124 while(max(abs(EV-EV2))>cri){
125     ### Set the current EV to the previous updated EV
126     EV <- EV2
127     ### Compute a new updated EV by iterating on the current EV
128     EV2 <- value.Iterate(EV)
129 }
130
131 ##### Do one last update to set EV equal to the last EV2
132 EV <- EV2
133
134 # get EV(x,i) for x=0,1,2,...,30
135 ##### EV contains extra states, which we needed to compute the above
136 ##### computation. Throw them away.

```

```

130 EV <- EV[1:31,]
131
132 ### Plot the EV of both replacing the engine (i = 1) and not replacing the
133     engine (i = 0) at every x
134 ### between 1 and 30
135 df <- data.frame('x'=c(1:30, 1:30), 'EV'=c(EV[2:31,1], EV[2:31,2]), 'Action'
136     = c(rep('i_0', 30), rep('i_1', 30)))
137
138 ### Generate a plot that compares the EV of replacing the engine and not
139     replacing the engine
140 ev_plot <- ggplot(df, aes(x=x, y=EV, color=Action)) + geom_point() + xlab('
141     Mileage') + ylab('EV') +
142     ggtitle('EV as a function of mileage and action\n at x between 1 and 30'
143     ) +
144     theme(plot.title = element_text(hjust = 0.5))
145 ggsave(ev_plot, file='ev_plot.png', height=6, width=6, units='in')
146
147
148 ### This is a plot to see the EV data in the attached rust matlab file. The
149     state space is different than ours (200 states),
150 ### so its hard to compare. Our's is linear (seems wrong), whereas the
151     provided data is not. However, the first 30 states
152 ### _do_ look approximately linear, so maybe we're not so far off.
153
154 df_rust <- data.frame('x'=c(seq(1,201), seq(1, 201)), 'EV'=c(data$EV[,1],
155     data$EV[,2]), 'Action' = c(rep('i_0', 201),
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

158 mean_x <- mean(x)
159 mean_engine_replacement_age <- mean(x[i == 1])
160 max_x <- max(x)
161 min_x <- min(x)
162 sd_x <- sd(x)
163 avg_replacements <- mean(apply(i, 2, function(x) {sum(x)}))
164
165 aggregate_stats <- c(mean_x, max_x, min_x, sd_x, mean_engine_)
166 aggregate_stats %>%
167   round(., 3) %>%
168   kable(., format='latex')
169
170 ### Calculate the per bus mean mileage, mean time to engine replacement,
171   max mileage, min mileage, and sd mileage
172 mean_x_per_bus <- apply(x, 2, function(x) {mean(x)})
173 max_x_per_bus <- apply(x, 2, function(x) {max(x)})
174 min_x_per_bus <- apply(x, 2, function(x) {min(x)})
175 sd_x_per_bus <- apply(x, 2, function(x) {sd(x)})
176 mean_engine_replacement_per_bus <- apply(x*i, 2, function(x) {sum(x)/sum(x
177   != 0)})
178 replacements <- apply(i, 2, function(x) {sum(x)})
179
180 ### Collate per bus information into a dataframe
181 per_bus_statistics <- data.frame(bus = seq(1, 100, 1),
182   mean_x_per_bus = mean_x_per_bus,
183   max_x_per_bus = max_x_per_bus,
184   min_x_per_bus = min_x_per_bus,
185   sd_x_per_bus = sd_x_per_bus,
186   mean_engine_replacement_per_bus = mean_
187     engine_replacement_per_bus,
188   replacements = replacements)
189
190 ### Create some plots
191 mean_mileage_plot <- ggplot(per_bus_statistics, aes(x=mean_x_per_bus)) +
192   geom_histogram() +
193   xlab('Mean_Mileage_(buckets_of_5,000_miles)') + ylab('Number_of_buses') +
194   ggtitle('Mean_mileage_\n across_buses') +
195   theme(plot.title = element_text(hjust = 0.5))
196 ggsave(mean_mileage_plot, file='mean_mileage_plot.png', height=4, width=4,
197   units='in')
198
199 max_mileage_plot <- ggplot(per_bus_statistics, aes(x=max_x_per_bus)) + geom_
200   histogram() +
201   xlab('Max_Mileage_(buckets_of_5,000_miles)') + ylab('Number_of_buses') +
202   ggtitle('Max_mileage_\n across_buses') +
203   theme(plot.title = element_text(hjust = 0.5))
204 ggsave(max_mileage_plot, file='max_mileage_plot.png', height=4, width=4,
205   units='in')
206
207 sd_mileage_plot <- ggplot(per_bus_statistics, aes(x=sd_x_per_bus)) + geom_
208   histogram() +
209   xlab('Mileage_Standard_Deviation_(buckets_of_5,000_miles)') + ylab('
210     Number_of_buses') +

```

```

201 ggtitle('Mileage_standard_deviation_\n_across_buses') +
202 theme(plot.title = element_text(hjust = 0.5))
203 ggsave(sd_mileage_plot, file='sd_mileage_plot.png', height=4, width=4,
204         units='in')
205
206 time_to_engine_replacement_plot <- ggplot(per_bus_statistics, aes(x=mean_
207     engine_replacement_per_bus)) + geom_histogram() +
208     xlab('Mean_Engine_Replacement_Mileage_(buckets_of_5,000_miles)') + ylab('
209     Number_of_buses') +
210     ggtitle('Mean_engine_replacement_\n_mileage_across_buses') +
211     theme(plot.title = element_text(hjust = 0.5))
212 ggsave(time_to_engine_replacement_plot, file='time_to_engine_replacement_
213     plot.png', height=4, width=4, units='in')
214
215 replacements_plot <- ggplot(per_bus_statistics, aes(x=replacements)) + geom
216     _histogram() +
217     xlab('Number_of_engine_replacements') + ylab('Number_of_buses') +
218     ggtitle('Number_of_engine_replacements_\n_across_buses') +
219     theme(plot.title = element_text(hjust = 0.5))
220 ggsave(replacements_plot, file='replacements_plot.png', height=4, width=4,
221         units='in')
222
223 #####
224 # Question 3.1 #
225 #####
226
227 ### This code will estimate the parameters beta, theta_1, and RC using the
228     nested fixed-point algorithm
229 ### described in Rust.
230
231
232 ### A function to compute the probability of Zurcher's choices using the
233     EVs we calculate using the EV calculation
234 ### framework above. The probability of choosing different actions
235     basically acts like a multichoice logit function.
236 choice_prob.Estimate <- function(){
237
238     ### Initialize an empty matrix to hold choice probability estimates.
239     p_i <- matrix(0,31,2)
240
241     ### Iterate through all of the possible mileage states, x.
242     for (x in 1:31){
243         ## For each mileage state x, calculate the probability that Zurcher
244             will choose i = 0.
245         p_i[x,1] <- exp(u(x-1,0)+beta*EV[x,1]) / (exp(u(x-1,0)+beta*EV[x,1])+exp(
246             u(x-1,1)+beta*EV[x,2]))
247         ## Calculate the probability of choosing i = 1 at each state, which is
248             just 1 - P(i = 0).
249         p_i[x,2] <- 1- p_i[x,1]
250     }
251
252     ### Return the updated p_i object
253     return(p_i)
254 }

```



```

243
244 #### A function to calculate the total log likelihood of the observed data
      given a set of parameters. This method assumes that
245 #### the probabilities across periods and buses are independent, so we can
      just add up all of the log probabilities.
246 log.likelihood.Compute <- function() {
247
248     #### Initialize 0-valued variables to hold the log choice probability, the
      log transition probability,
249     #### and the sum of the two.
250     log_choice_prob <- 0
251     log_transition_prob <- 0
252     total <- 0
253
254     #### Iterate over buses
255     for (bus in 1:100) {
256         #### Iterate over time periods
257         for (t in 1:999) {
258             #### We special case mileage states greater than 30, since they are a
              bit strange in our data. Otherwise, we calculate
259             #### the choice probability using the current value of p_i according
              to the EV values we calculated to get the
260             #### choice probability. Take the log and add it to the current
              running value.
261             if (x[t,bus] <= 30){
262                 log_choice_prob <- log(p_i[x[t,bus]+1,i[t,bus]+1]) + log_choice_
                  prob
263             #### Do the same thing for our special cased, x > 30 case.
264             } else {
265                 log_choice_prob <- log(p_i[31,i[t,bus]+1]) + log_choice_prob
266             }
267
268             #### Calculate over the transitions for each bus the sum of the log
              transition probabilities. We have our estimates of
269             #### theta_3 given the empirical transition probabilities. So we can
              just grab that for each observed transition and add it
270             #### to the total log transition probability.
271
272             #### First we do the j = 0 case.
273             if (x[t+1,bus]-x[t,bus]==0) {
274                 log_transition_prob <- log(theta_30)+log_transition_prob
275             #### Then the j = 1 case.
276             } else if (x[t+1,bus]-x[t,bus]==1) {
277                 log_transition_prob <- log(theta_31)+log_transition_prob
278             #### And finally the j = 2 case.
279             } else if (x[t+1,bus]-x[t,bus]==2) {
280                 log_transition_prob <- log(theta_32)+log_transition_prob
281             }
282         }
283
284         #### Now, get the total log likelihood by adding up all of the
              transition components and the choice components.
285         total <- (log_choice_prob+log_transition_prob) + total
286     }

```

```

287     return (total)
288 }
289
290 ### Now we're actually going to use the nested fixed point algorithm to get
291     the maximum likelihood estimates of the parameters
292 ### that we care about. This process has three steps.
293
294 ### Step 1: We would calculate theta_30, theta_31, and theta_31 directly
295     from the data. This step is not in the loop, and we've
296 ### actually already done this and it doesn't change, so we don't need to
297     do it again.
298
299 ### Step 2: Next, we are going to set up a grid over values of theta_1,
300     beta, and RC that we will calculate the
301 ### log likelihood to determine the maximum likelihood parameter values. We
302     'll also initialize a dataframe
303 ### to hold the parameter values and the log likelihoods.
304
305 theta_1_range <- seq(.01,.10,.01)
306 beta_range <- seq(.90,.99,.01)
307 RC_range <- seq(6,15,1)
308 likelihood <- data.frame('theta_1'=rep(0), 'beta'=rep(0), 'RC'=rep(0), 'log.
309     likelihood'=rep(0))
310
311 ### Step 3: Now we actually do the nested fixed point computation.
312
313 ### Loop through theta_1
314 for (theta_1 in theta_1_range) {
315     ### Loop through beta
316     for (beta in beta_range) {
317         ### Loop through RC
318         for (RC in RC_range) {
319             print(paste(c(theta_1, beta, RC), collapse='_'))
320
321             ### Initialize the EV functions to the initial values we used above.
322             EV <- matrix(0,33,2)
323             EV2 <- matrix(-80,33,2)
324
325             ### Iteratively compute the EV values.
326             while(max(abs(EV-EV2))>cri){
327                 EV <- EV2
328                 EV2 <- value.Iterate(EV)
329             }
330
331             EV <- EV2
332             EV <- EV[1:31,]
333
334             ### Given these values of EV, calculate the choice probabilities
335             p_i <- choice.prob.Estimate()
336
337             ### Given the EV values, the choice probabilities and the parameters,
338             calculate
339             ### the log-likelihood of the data.

```

```

333     likelihood <- rbind(likelihood , c(theta_1, beta , RC, log.likelihood .
334         Compute()))
335   }
336 }
337
338 #### Retrieve the row in the likelihood dataframe corresponding to the
339     maximum likelihood estimate
340 likelihood <- likelihood[-1,]
341 parameter_estimates <- likelihood[which.max(likelihood[,4]) ,]
342
343 #### Use these parameters and get the relevant estimate of EV and p_i
344 theta_1 = parameter_estimates$theta_1
345 beta = parameter_estimates$beta
346 RC = parameter_estimates$RC
347
348 EV <- matrix(0,33,2)
349 EV2 <- matrix(-80,33,2)
350 #### Iteratively compute the EV values.
351 while(max(abs(EV-EV2))>cri){
352     EV <- EV2
353     EV2 <- value.Iterate(EV)
354 }
355 EV <- EV2
356 EV <- EV[1:31,]
357 #### Given these values of EV, calculated the choice probabilities
358 p_i <- choice.prob.Estimate()
359 #### Given the EV values , the choice probabilities and the parameters,
360 calculate
361 the log-likelihood of the data.
362 likelihood <- rbind(likelihood , c(theta_1, beta , RC, log.likelihood .Compute()))
363
364 save(EV, p_i, likelihood , parameter_estimates , file='rust_estimate.Rdata')
365
366 #####
367 # Question 3.2 #
368 #####
369
370 #### Now we will get estimates of the parameters using the Hotz and Miller
371 conditional choice probability approach. This will
372 allow us to compare these parameter estimates to those obtained using
373 the Rust approach.
374
375 #### First , we need to calculate the probability of the agent choosing
376 either i = 0 or i = 1 based on the state that they find
377 a given bus in , x, at some time period t. This will be the baseline
378 that we use to try and find the best parameter values
379 (i.e., which parameter values minimize the infinity norm between these
380 true probabilities and the estimated probabilities)
381
382 #### The probability matrix
383 p_ix <- matrix(0,33,2)
384 #### The vector of how often the agent chooses i=1 given state x
385 ones <- vector()

```

```

379 ##### The vector of how often the agent finds a bus in state x
380 total <- vector()
381
382 ##### Loop through the states
383 for (state in 0:32){
384
385     ##### For a given state, a will track how many times i = 1 and b will track
386         how many times that state occurs.
387     ##### Initialize them to 0 for the given state.
388     a <- 0
389     b <- 0
390
391     ##### Loop over the buses
392     for (bus in 1:100){
393
394         ##### Increment how many times the agent chooses i = 1 in state x
395         a <- sum(i[which(x[,bus]==state),bus]) + a
396         ##### Increment how many times the state x occurs
397         b <- length(i[which(x[,bus]==state),bus]) + b
398     }
399
400     ##### Add the most recent estimates to the vector.
401     ones[state+1] <- a
402     total[state+1] <- b
403 }
404
405 ##### Based on the ones and total vectors, updated the choice probability
406 matrix.
407 p_ix[,1] <- 1-ones/total
408 p_ix[,2] <- ones/total
409
410 ##### Plot conditional choice probabilities
411 p_ix_df <- as.data.frame(p_ix)
412 p_ix_df$state <- as.numeric(rownames(p_ix_df))
413 names(p_ix_df) <- c('P(i=0)', 'P(i=1)', 'State')
414 ccp_plot <- p_ix_df %>%
415     ggplot(., aes(x=State, y='P(i = 1)')) + geom_line() +
416     ggtitle('Conditional_probability_of_engine_replacement\nas_a_function_of_mileage') +
417     theme(plot.title = element_text(hjust = 0.5))
418 ggsave(ccp_plot, file='ccp_plot.png', height=4, width=4, units='in')
419
420 ##### The function below uses the Hotz and Miller method to estimate V and p_
421 ix_hat for every state and period
422 ##### given a set of model parameters (beta, theta_1, and RC).
423 approximate.V_pihat <- function() {
424     ##### Initialize an empty valuation matrix
425     V <- matrix(0,33,2)
426     ##### Initialize an empty conditional choice probability matrix
427     p_ix_hat <- matrix(0,33,2)
428
429     ##### Iterate through the states
430     for (state in 0:30){

```

```

428     ### Initialize a and b, which will basically track a running total of V
      for different choices over simulations, to 0.
429     a = 0
430     b = 0
431     ### Iterate through the simulations. Note that ideal we would probably
      want to go more than one time step into the
432     ### future. However, because of the limitations in our dataset, we only
      go one time step forward. This is mainly because
433     ### it's unclear how we would draw i (the choice) for states that do
      not appear in our data (i.e., x = 34).
434     for (s in 1:S){
435         ## Conditional on choosing i = 0, simulate the next state that a
          given bus will end up in by drawing from the
436         ## transition probabilities.
437         x_prime_0 = state + sample(c(0,1,2),1,replace = T, prob = c(theta_30,
          theta_31,theta_32))
438         ## Conditional on choosing i = 0 and ending up in some state in the
          next time period, randomly simulate a draw from
439         ## i based on the conditional choice probabilities
440         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]))
441         # Figure out the expected utility from this truncated sequence of
          choices.
442         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
443
444         ## Conditional on choosing i = 1, we don't need to simulate the next
          state that a bus will end up in. It will always
445         ## be x = 0. So we jump right to simulating the draw from i for x =
          0.
446         i_prime_1 = sample(c(0,1),1,replace=T,prob = c(p_ix[1,1],p_ix[1,2]))
447         ## Figure out the expected utility from this truncated sequence of
          choices.
448         b = (u(state,1) + beta*(u(0,i_prime_1)+gamma-log(p_ix[1,i_prime_1+1]
          ))) + b
449     }
450
451     ## Set the value of V to be the average over all S of our simulations
      for both the i = 0 and i = 1 choices.
452     V[state+1,1] = a/S
453     V[state+1,2] = b/S
454     ## Use the multinomial logit-esque probability expression to figure out
      the probability of choosing i = 0 or i = 1
455     ## given that the bus is in state x.
456     p_ix_hat[state+1,1] <- exp(V[state+1,1])/(exp(V[state+1,1])+exp(V[state
      +1,2]))
457     p_ix_hat[state+1,2] <- 1- p_ix_hat[state+1,1]
458 }
459
460 # Put final output into a list and return it
461 results <- list('V' = V, 'p_ix_hat' = p_ix_hat)
462 return(results)
463 }
464

```

```

465 ##### Specify a number of constants that will be used in the Hotz and Miller
      algorithm:
466 ##### S: The number of "simulations" to do per state / decision
467 ##### gamma: This should be Euler's constant
468 ##### theta_1_range: The range of theta_1 values to test
469 ##### beta_range: The range of beta values to test
470 ##### RC_range: The range of RC values to test
471 S = 1000
472 theta_1_range <- seq(.01,.10,.01)
473 beta_range <- seq(.90,.99,.01)
474 RC_range <- seq(6,15,1)
475
476 ##### Initialize a dataframe to hold different parameter combinations and the
      infinity-norm between the actual conditional
477 choice probabilities and the estimated ones
478 difference <- data.frame('theta_1'=rep(0), 'beta'=rep(0), 'RC'=rep(0), '
      difference'=rep(0))
479
480 ##### Loop through theta_1
481 for (theta_1 in theta_1_range) {
482   ##### Loop through theta_2
483   for (beta in beta_range) {
484     ##### Loop through RC
485     for (RC in RC_range) {
486       # Check progress
487       print(paste(c(theta_1, beta, RC), collapse='_'))
488
489       ##### Get estimates of V and P_ix_hat using the Hotz and Miller method
490       v_and_p_ix_hat <- approximate.V_pixhat()
491       V = v_and_p_ix_hat$V
492       p_ix_hat <- v_and_p_ix_hat$p_ix_hat
493
494       ##### Now that we have a full conditional choice probability matrix,
495       calculate the infinity norm (i.e., largest
496       ##### absolute difference between the empirical conditional choice
497       probabilities and those estimated with the
498       ##### given parameters)
499       difference <- rbind(difference, c(theta_1, beta, RC, max(abs(p_ix[1:31,] -
500       p_ix_hat[1:31,]))))
501     }
502   }
503 }
504
505 ##### Find the set of parameters that minimizes this difference
506 difference <- difference[-1,]
507 parameter_estimates <- difference[which.min(difference[,4]),]
508
509 ##### Use these parameters and get the relevant estimate of V and p_ix_hat
510 theta_1 = parameter_estimates$theta_1
511 beta = parameter_estimates$beta
512 RC = parameter_estimates$RC
513 best_guesses <- approximate.V_pixhat()
514 V <- best_guesses$V
515 p_ix_hat <- best_guesses$p_ix_hat

```

```

513 |
514 | save(V, p_ix_hat, difference, parameter_estimates, file='hotz_and_miller_
    | estimate.Rdata')
515 |
516 | #####
517 | # Question 3.3 #
518 | #####
519 |
520 | ### old engine
521 | theta_1 = .05
522 | RC = 10
523 |
524 | ### apply Rust's approach
525 | cri <- 10^(-8)
526 | EV <- matrix(0,33,2)
527 | EV2 <- matrix(-80,33,2)
528 |
529 | while(max(abs(EV-EV2))>cri){
530 |   ### Set the current EV to the previous updated EV
531 |   EV <- EV2
532 |   ### Compute a new updated EV by iterating on the current EV
533 |   EV2 <- value.Iterate(EV)
534 | }
535 |
536 | ### Do one last update to set EV equal to the last EV2
537 | EV_old <- EV2
538 |
539 | ### new engine
540 | theta_1 = .02
541 | RC = 20
542 |
543 | ### repeat the exercise above
544 | cri <- 10^(-8)
545 | EV <- matrix(0,33,2)
546 | EV2 <- matrix(-80,33,2)
547 |
548 | while(max(abs(EV-EV2))>cri){
549 |   ### Set the current EV to the previous updated EV
550 |   EV <- EV2
551 |   ### Compute a new updated EV by iterating on the current EV
552 |   EV2 <- value.Iterate(EV)
553 | }
554 |
555 | ### Do one last update to set EV equal to the last EV2
556 | EV_new <- EV2
557 |
558 | ### Returns which engine is preferred (old or new)
559 | c('old','new')[which.max(c(EV_old[1,1],EV_new[1,1]))]
560 |
561 | #####
562 | # Question 3.4 #
563 | #####
564 |

```

```

565 ##### This function simulates, for one agent, a sequence of state transitions
      and also engine replacement decisions
566 simulate_sequence <- function(n_periods) {
567   ##### Initialize empty vectors to hold states and engine replacement
      transitions
568   x_values <- rep(0, n_periods)
569   i_values <- rep(0, n_periods)
570   ##### Every bus starts at state 0
571   x_values[1] <- 0
572   ##### Go through the progression
573   for (j in 1:length(x_values)) {
574     ##### Make a decision based on current state
575     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]))
576     ##### If decision is to not replace, continue on and increment x randomly
577     if (i_values[j] == 0) {
578       x_values[j+1] = x_values[j] + sample(c(0,1,2),1,replace = T, prob = c
      (theta_30,theta_31,theta_32))
579     ##### If decision is to replace, reset state to 0
580     } else {
581       x_values[j+1] = 0
582     }
583   }
584   ## Generate a decision for the last period, even though we never see the
      fruits of that decision
585   i_values[length(i_values)] = sample(c(0,1),1,replace=T,prob = c(p_ix_hat[
      x_values[length(x_values)] + 1,1],
586                                     p_ix_hat[
      x_
      values
      [
      length
      (x_
      values
      )] +
      1,2])
      )
587   ##### Return the states and replacement decisions in a list
588   results <- list('x_values' = x_values, 'i_values' = i_values)
589   return(results)
590 }
591
592 ##### Given a set of parameters, this function generates period-by-period
      demand estimates for new buses (e.g.,
593 ##### how many buses will get their engine replaced in each period)
594 estimate_demand <- function(n_sims, n_buses, n_periods) {
595
596   ##### Initialize a vector to hold simulated demand
597   simulated_demand_total <- rep(0, n_periods)
598
599   ##### Run a bunch of simulations and simulate engine replacement decisions
600   for (j in 1:n_sims) {
601     simulated_demand_total = simulated_demand_total + simulate_sequence(n_
      periods)$i_values

```



```

602 }
603
604 ### Divide by the number of sims to get averages , multiply by number of
        buses (this works because
605 ###buses are independent). Then return what we get.
606 return((n_buses/n_sims)*simulated_demand_total)
607 }
608
609 ### Get demand as a function of RC for the first bus
610
611 ## Specify the range of RCs, as well as constants.
612 RC_range = seq(0, 15, .25)
613 n_periods = 15
614 n_sims = 1000
615 n_buses = 100
616 load('rust_estimate.Rdata')
617
618 ## Initialize an empty dataframe to hold results
619 estimated_demand_df <- data.frame(time_period = c(),
620                                   RC = c(),
621                                   demand = c(),
622                                   engine = c())
623
624 # Loop through the RCs, then estimate the probabilities using the Rust
        method, then do simulation.
625 for (j in RC_range) {
626     RC = j
627
628     ### Set a critical value for to measure the deviation between iterative
        updates of EV. The distance between the two EV matrices
629     ### is the infinity norm of the difference
630     cri <- 10^(-8)
631
632     ### Set an initial value for the EV matrix (all 0s, EV), and another EV
        object to hold the updated estimates, EV2.
633     EV <- matrix(0,33,2)
634     EV2 <- matrix(-80,33,2)
635
636     ## While the infinity norm is less than the threshold, iterate
637     while(max(abs(EV-EV2))>cri){
638
639         ### Set the current EV to the previous updated EV
640         EV <- EV2
641         ### Compute a new updated EV by iterating on the current EV
642         EV2 <- value.Iterate(EV)
643     }
644
645     ### Do one last update to set EV equal to the last EV2
646     EV <- EV2
647
648     # get EV(x,i) for x=0,1,2,...,30
649     ### EV contains extra states, which we needed to compute the above
        computation. Throw them away.
650     EV <- EV[1:31,]

```

```

651
652 ### Get estimated probability based on the above EV
653 p_ix_hat <- choice.prob.Estimate()
654 ### Estimate demand using that probability
655 estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)
656
657 ### Add this estimate to a temp dataframe
658 estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
659                                     ,
660                                     RC = rep(RC, n_periods),
661                                     demand = estimated_demand,
662                                     engine = rep('Engine_1', n_periods))
663 ### Collate temp dataframe to full dataframe
664 estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
665                               temp)
666 }
667 ### Reset theta_1 to the "new engine", redo the exercise above.
668 theta_1 = .02
669
670 ### Loop through RCs
671 for (j in RC_range) {
672   RC = j
673
674   ### Set a critical value for to measure the deviation between iterative
675   updates of EV. The distance between the two EV matrices
676   ### is the infinity norm of the difference
677   cri <- 10^(-8)
678
679   ### Set an initial value for the EV matrix (all 0s, EV), and another EV
680   object to hold the updated estimates, EV2.
681   EV <- matrix(0,33,2)
682   EV2 <- matrix(-80,33,2)
683
684   ## While the infinity norm is less than the threshold, iterate
685   while(max(abs(EV-EV2))>cri){
686     ### Set the current EV to the previous updated EV
687     EV <- EV2
688     ### Compute a new updated EV by iterating on the current EV
689     EV2 <- value.Iterate(EV)
690   }
691
692   ### Do one last update to set EV equal to the last EV2
693   EV <- EV2
694
695   # get EV(x,i) for x=0,1,2,...,30
696   ### EV contains extra states, which we needed to compute the above
697   computation. Throw them away.
698   EV <- EV[1:31,]
699
700   ### Get probability estimates based on EV
701   p_ix_hat <- choice.prob.Estimate()

```

```

700   ### Estimate demand
701   estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)
702
703   ### Add to temp dataframe
704   estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
705     ,
706     RC = rep(RC, n_periods),
707     demand = estimated_demand,
708     engine = rep('Engine_2', n_periods)
709   )
710
711   ### Collate to full dataframe
712   estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
713     temp)
714 }
715
716   ### For a reduced set of RCs, see the period-by-period demand
717   per_period_demand_plot <- estimated_demand_df %>%
718     filter(RC %in% c(1, 3, 5, 7, 10)) %>%
719     mutate(RC = as.factor(RC)) %>%
720     ggplot(., aes(x=time_period, y=demand, color=RC)) + geom_line() +
721     facet_wrap(~engine) + xlab('Period') + ylab('Demand_for_Engines') +
722     ggtitle('Demand_for_engines_over_time') +
723     theme(plot.title = element_text(hjust = 0.5))
724   ggsave(per_period_demand_plot, file='per_period_demand_plot.png', height=4,
725     width=6, units='in')
726
727   ### Aggregate over periods to get demand as a function of RC for different
728     thetas.
729   aggregate_demand_plot <- estimated_demand_df %>%
730     group_by(RC, engine) %>%
731     summarise(total_demand = sum(demand)/n_periods) %>%
732     ungroup() %>%
733     ggplot(., aes(x=RC, y=total_demand, color=engine)) + geom_line() + xlab('
734     RC') +
735     ylab('Average_per-period_demand') + ggtitle('Average_per-period_engine_
736     demand_for_100_buses') +
737     theme(plot.title = element_text(hjust = 0.5))
738   ggsave(aggregate_demand_plot, file='aggregate_demand_plot.png', height=4,
739     width=6, units='in')
740
741   #####
742   # Question 3.5 #
743   #####
744
745   n_periods = 15
746
747   ### for a given RC and engine, average demand across time periods
748   engine_value <- estimated_demand_df %>%
749     group_by(RC, engine) %>%
750     summarize(avg_demand=mean(demand))

```

```

744 # Initialize an empty dataframe to hold value as a function of RC for both
      engines
745 engine_value_df <- data.frame(RC = c(), engine_value = c(), engine = c())
746
747 ## Loop over RC values
748 for (j in RC_range) {
749   # Get the engine 1 value at this RC
750   engine_1_value <- engine_value %>%
751     filter(engine=='Engine_1',RC>=j) %>%
752     summarize(value = RC*avg_demand) %>%
753     summarize(total_value = sum(value)) %>%
754     as.numeric()* n_periods
755
756   # Get the engine 2 value at this RC
757   engine_2_value <- engine_value %>%
758     filter(engine=='Engine_2',RC>=j) %>%
759     summarize(value = RC*avg_demand) %>%
760     summarize(total_value = sum(value)) %>%
761     as.numeric()* n_periods
762
763   # Add these two engine values to the dataframe
764   engine_value_df <- rbind(engine_value_df, data.frame(RC = j, engine_value
     = engine_1_value, engine = 'Engine_1'))
765   engine_value_df <- rbind(engine_value_df, data.frame(RC = j, engine_value
     = engine_2_value, engine = 'Engine_2'))
766 }
767
768 # Create a plot of value as a function of RC for both engines
769 value_plot <- ggplot(engine_value_df, aes(x=RC, y=engine_value, color=
     engine)) + geom_line() +
770   xlab('RC') + ylab('Value') + ggtitle('Value_produced_by_each_engine_as_a_
     function_of_RC_(c=0)') +
771   theme(plot.title = element_text(hjust = 0.5))
772 ggsave(value_plot, file='value_plot.png', height=4, width=6, units='in')
773
774 ### for engine 1 (original one), calculate value at its true RC
775 engine_value %>%
776   filter(engine=='Engine_1',RC>=6) %>%
777   summarize(value = RC*avg_demand) %>%
778   summarize(total_value = sum(value)) %>%
779   as.numeric()* n_periods
780
781 ### for engine 2 (new one), calculate value at its true RC
782 engine_value %>%
783   filter(engine=='Engine_2',RC>=20) %>%
784   summarize(value = RC*avg_demand) %>%
785   summarize(total_value = sum(value)) %>%
786   as.numeric()* n_periods

```

Listing 1: ./rust.R

REFERENCES

- [1] V Joseph Hotz and Robert A Miller. Conditional choice probabilities and the estimation of dynamic models. *The Review of Economic Studies*, 60(3):497–529, 1993.
- [2] John Rust. Optimal replacement of gmc bus engines: An empirical model of harold zurcher. *Econometrica: Journal of the Econometric Society*, pages 999–1033, 1987.