

---

# 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  $\theta_1, \theta_3$ ):

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

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  $\epsilon$ , 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  $\epsilon$ ).

- (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 (1987) 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  $\Delta x$ ). 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  $\theta$  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.

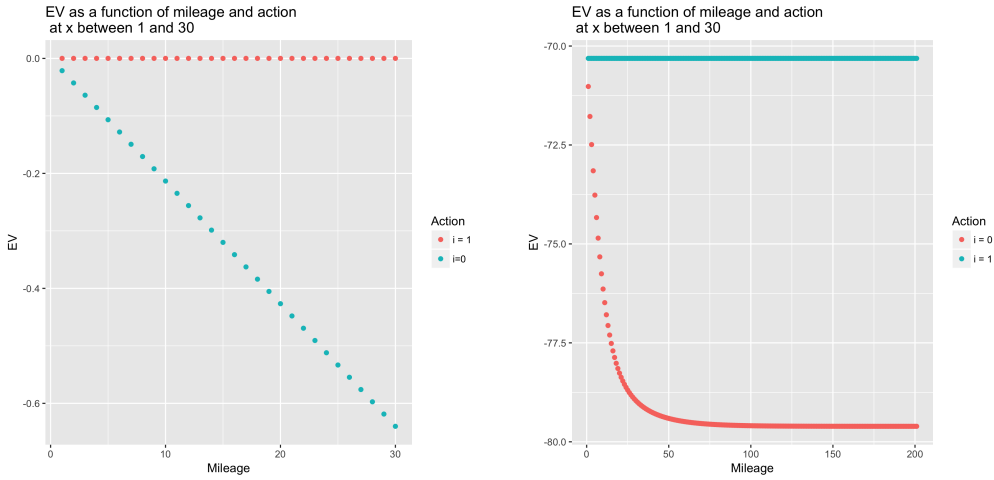


Figure 1: Expected Value Function for  $i = 0$  and  $i = 1$ . Left panel shows results using iterative method, right panel shows provided Rust results.

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.,  $\sim 70$ ), 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.

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  $(\theta_1, \beta, RC)$ , 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  $(\theta_1, \beta, RC)$  and choose the set of parameters that maximizes the log-likelihood of the data. The maximum likelihood parameters obtained with the Rust method are:

$$\begin{aligned} \theta_1 &= 0.1 \\ \beta &= 0.99 \\ RC &= 6 \end{aligned}$$

- (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:

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

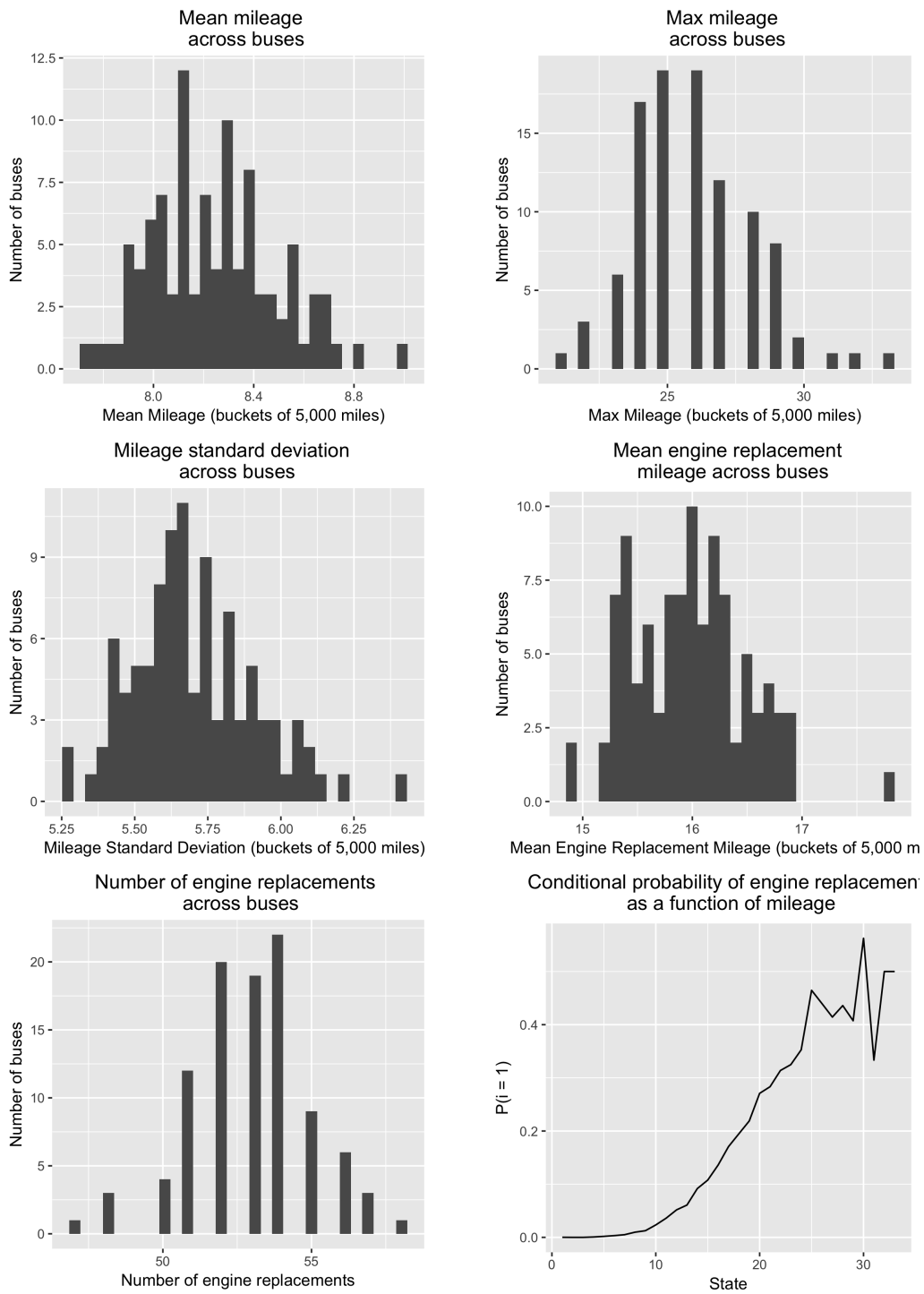


Figure 2: Distribution of various summary statistics across buses, as well as the empirical conditional choice probability for Zurcher.

The choice-specific value function (minus the structural error, and suppressing the dependency on  $\theta_1, \theta_3$ ) can be presented recursively in the following form:

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

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

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

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:

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

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  $\hat{\mathbb{P}}(i | x)$ ,  $\gamma$  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.

- (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 original engine over the new one.
- (4) We want to compute HZ's demand function for the two buses, which we will denote as engine 1 ( $\theta_1 = 0.09, RC = 6$ ) 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 method to iterate EV values. We use the Rust methodology because the Hotz and Miller 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:

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

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 - although you cannot distinguish them in the figure, there are small differences (on the order of a tenths of an engine). 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. Whatever our EV estimation issue, it is rearing its ugly head again and making these demand curves very similar.
  2. The increase in  $RC$  from engine 1 to engine 2 is almost perfectly offset by the decrease in  $\theta_1$ , 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.

For engine 1, the total value is 343.3. For engine 2, the total value is 0 (because our estimated demand is zero at  $RC = 20$ ).

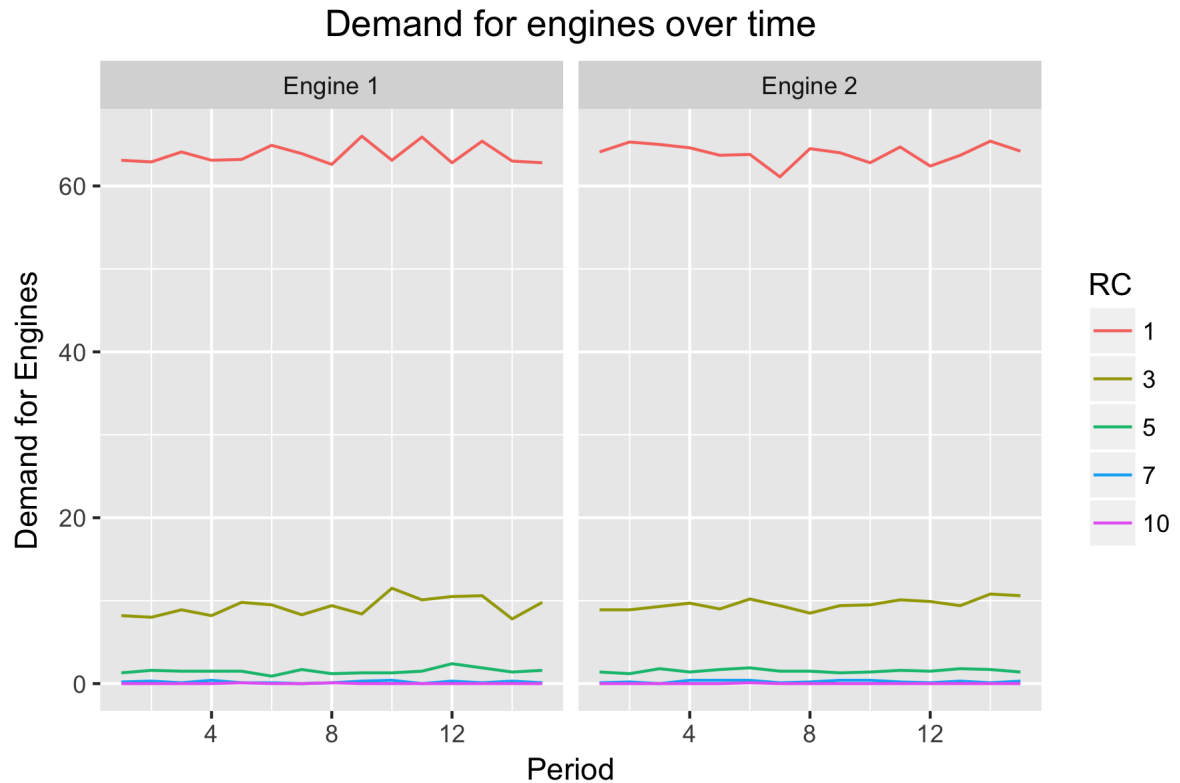


Figure 3: The demand for engines across a fleet of 100 buses as a function of period (over the first 15 periods) for different values of RC. Unsurprisingly, when RC is lower, HZ is much more willing to change bus engines.

## 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)

```



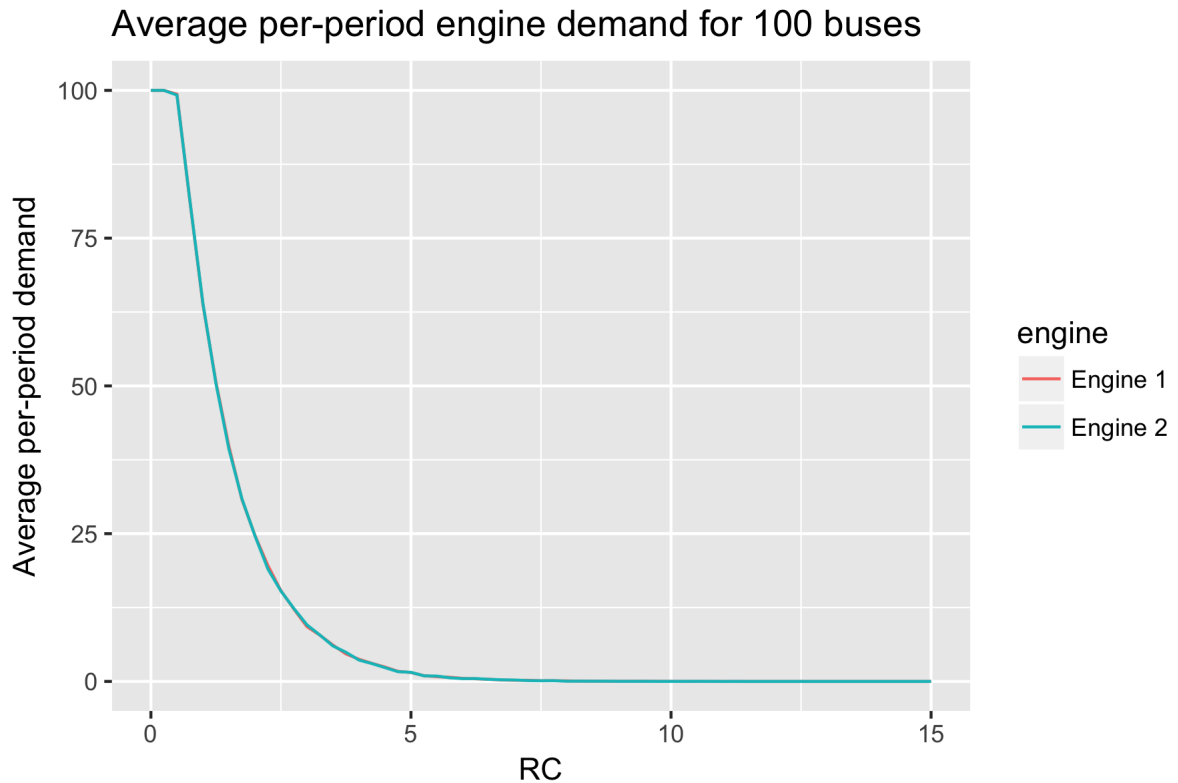


Figure 4: Aggregate per-period demand for new engines across a fleet of 100 buses as a function of RC. For engine 1,  $\theta_1 = 0.09$ . For engine 2,  $\theta_1 = 0.02$ .

```

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
36     from period j to period j+1
37     jk <- xk[-1]-xk[-1000]
38
39     ### We only care about periods where i=0 for transition
40     probabilities, since i=1 will always send
41     ### x back to 0. This selects out only time periods for this bus
42     where i = 0
43     j <- jk[ik==0]
44
45     ### This counts up how many times the mileage bucket counter, x,
46     moves up by 0, 1, or 2 when i=0
47     zero[k] <- length(j[j==0])
48     one[k] <- length(j[j==1])
49     two[k] <- length(j[j==2])
50 }
51
52 ## Estimate the x_t-independent transition probabilities by dividing the
53 number of times for each transition by the
54 ## total number of transitions
55 theta_30 = sum(zero)/(sum(zero)+sum(one)+sum(two))
56 theta_31 = sum(one)/(sum(zero)+sum(one)+sum(two))
57 theta_32 = sum(two)/(sum(zero)+sum(one)+sum(two))
58
59 #####
60 # Question 2.3 #
61 #####
62
63 ##### We'll now take the true values of the parameter values as given, and
64 use the method described in Rust (1987) to iteratively
65 ##### estimate the value function (or in this case, the EV function).
66
67 ## Initialize parameters to their true values
68 theta_1 = .05
69 theta_30 = .3
70 theta_31 = .5
71 theta_32 = .2
72 beta = .99
73 RC = 10
74
75 ##### Define the linear cost function. If an engine is not replaced, the bus
76 incurs cost theta_1*x, so cost
77 ##### increases linearly as a bus gets older.
78 cost <- function(x){
79     return (theta_1*x)}
80
81 ##### Define the utility function at mileage x from action i. If the agent
82 chooses to replace the engine in a bus,
83 ##### it costs RC. If they choose _not_ to replace the engine, they incur the
84 cost of running the bus at mileage x.
85 u <- function(x,i){
86     -RC*i - cost(x*(1-i))
87 }
88
89

```

```

80
81 ##### The value function can be estimated through an iteration procedure. We
      start with some initial guess for EV,
82 ##### calculate EV with an expression that includes our initial guess of EV,
      and continue iterating until the difference
83 ##### between subsequent EV estimates becomes small.
84
85 ##### value.Iterate is a function to iteratively update the value function
      according to the methodology in Rust. The function
86 ##### 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
87 ##### EV values for each decision d at every possible current mileage value
      x.
88 value.Iterate <- function(EV){
89
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
          are three contributions here – one from the
93         ## 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
94         ## 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.
95         EV2[x,1] <- log(exp(u(x-1,0)+beta*EV[x,1])+exp(u(x-1,1)+beta*EV[x,2]))*
          theta_30 + gamma
96         + log(exp(u(x,0)+beta*EV[x+1,1])+exp(u(x,1)+beta*EV[x+1,2]))*theta_31
          + gamma
97         + log(exp(u(x+1,0)+beta*EV[x+2,1])+exp(u(x+1,1)+beta*EV[x+2,2]))*theta_
          32 + gamma
98
99         ## Update the EV value corresponding to replacing the engine. When the
          engine is replaced, x at the next period will
100        ## deterministically reset to x = 0.
101        EV2[x,2] <- log(exp(u(0,0)+beta*EV[1,1])+exp(u(0,1)+beta*EV[1,2])) +
          gamma
102    }
103
104    ## Return the updated EV values.
105    return(EV2)
106 }
107
108 ##### Set a critical value for to measure the deviation between iterative
      updates of EV. The distance between the two EV matrices
109 ##### is the infinity norm of the difference
110 cri <- 10^(-8)
111
112 ##### Set an initial value for the EV matrix (all 0s, EV), and another EV
      object to hold the updated estimates, EV2.
113 EV <- matrix(100,33,2)
114 EV2 <- matrix(0,33,2)
115
116 ## While the infinity norm is less than the threshold, iterate
117 while(max(abs(EV-EV2))>cri){
118

```

```

119   ### Set the current EV to the previous updated EV
120   EV <- EV2
121   ### Compute a new updated EV by iterating on the current EV
122   EV2 <- value.Iterate(EV)
123 }
124
125 ### Do one last update to set EV equal to the last EV2
126 EV <- EV2
127
128 # get EV(x,i) for x=0,1,2,...,30
129 ### EV contains extra states, which we needed to compute the above
    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
    engine (i = 0) at every x
133 ### between 1 and 30
134 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)))
135
136 ### Generate a plot that compares the EV of replacing the engine and not
    replacing the engine
137 ev_plot <- ggplot(df, aes(x=x, y=EV, color=Action)) + geom_point() + xlab('
    Mileage') + ylab('EV') +
138   ggtitle('EV as a function of mileage and action \n at x between 1 and 30'
    ) +
139   theme(plot.title = element_text(hjust = 0.5))
140 ggsave(ev_plot, file='ev_plot.png', height=6, width=6, units='in')
141
142 ### This is a plot to see the EV data in the attached rust matlab file. The
    state space is different than ours (200 states),
143 ### so its hard to compare. Our's is linear (seems wrong), whereas the
    provided data is not. However, the first 30 states
144 ### _do_ look approximately linear, so maybe we're not so far off.
145
146 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),
147
148
149
150
151
152
153
154
155
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

```

```

149   ggtitle('Rust_dataset_EV_as_a_function_of_mileage_and_action\n_at_x_
      between_1_and_201') +
150   theme(plot.title = element_text(hjust = 0.5))
151   ggsave(ev_plot_rust, file='ev_plot_rust.png', height=6, width=6, units='in'
      )
152
153   #####
154   # Question 2.4 #
155   #####
156
157   ### Calculate the mean mileage, mean time to engine replacement, max
      mileage, min mileage, and sd mileage over the whole sample
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,
      max mileage, min mileage, and sd mileage
171   mean_x_per_bus <- apply(x, 2, function(x) {mean(x)})
172   max_x_per_bus <- apply(x, 2, function(x) {max(x)})
173   min_x_per_bus <- apply(x, 2, function(x) {min(x)})
174   sd_x_per_bus <- apply(x, 2, function(x) {sd(x)})
175   mean_engine_replacement_per_bus <- apply(x*i, 2, function(x) {sum(x)/sum(x
      != 0)})
176   replacements <- apply(i, 2, function(x) {sum(x)})
177
178
179   ### Collate per bus information into a dataframe
180   per_bus_statistics <- data.frame(bus = seq(1, 100, 1),
      mean_x_per_bus = mean_x_per_bus,
181     max_x_per_bus = max_x_per_bus,
182     min_x_per_bus = min_x_per_bus,
183     sd_x_per_bus = sd_x_per_bus,
184     mean_engine_replacement_per_bus = mean_
185       engine_replacement_per_bus,
186     replacements = replacements)
187
188   ### Create some plots
189   mean_mileage_plot <- ggplot(per_bus_statistics, aes(x=mean_x_per_bus)) +
      geom_histogram() +
190     xlab('Mean_Mileage_(buckets_of_5,000_miles)') + ylab('Number_of_buses') +
      ggtitle('Mean_mileage\n_across_buses') +
191     theme(plot.title = element_text(hjust = 0.5))
192   ggsave(mean_mileage_plot, file='mean_mileage_plot.png', height=4, width=4,
      units='in')
193

```

```

194 max_mileage_plot <- ggplot(per_bus_statistics , aes(x=max_x_per_bus)) + geom_
    _histogram() +
195   xlab('Max_Mileage_(buckets_of_5,000_miles)') + ylab('Number_of_buses') +
    ggtitle('Max_mileage_\n_across_buses') +
196   theme(plot.title = element_text(hjust = 0.5))
197 ggsave(max_mileage_plot , file='max_mileage_plot.png' , height=4, width=4,
    units='in')
198
199 sd_mileage_plot <- ggplot(per_bus_statistics , aes(x=sd_x_per_bus)) + geom_
    _histogram() +
200   xlab('Mileage_Standard_Deviation_(buckets_of_5,000_miles)') + ylab('
    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,
    units='in')
204
205 time_to_engine_replacement_plot <- ggplot(per_bus_statistics , aes(x=mean_
    engine_replacement_per_bus)) + geom_histogram() +
206   xlab('Mean_Engine_Replacement_Mileage_(buckets_of_5,000_miles)') + ylab('
    Number_of_buses') +
207   ggtitle('Mean_engine_replacement_\n_mileage_across_buses') +
208   theme(plot.title = element_text(hjust = 0.5))
209 ggsave(time_to_engine_replacement_plot , file='time_to_engine_replacement_
    plot.png' , height=4, width=4, units='in')
210
211 replacements_plot <- ggplot(per_bus_statistics , aes(x=replacements)) + geom_
    _histogram() +
212   xlab('Number_of_engine_replacements') + ylab('Number_of_buses') +
213   ggtitle('Number_of_engine_replacements_\n_across_buses') +
214   theme(plot.title = element_text(hjust = 0.5))
215 ggsave(replacements_plot , file='replacements_plot.png' , height=4, width=4,
    units='in')
216
217
218 #####
219 # Question 3.1 #
220 #####
221
222 ### This code will estimate the parameters beta , theta_1, and RC using the
    nested fixed-point algorithm
223 ### described in Rust.
224
225 ### A function to compute the probability of Zurcher's choices using the
    EVs we calculate using the EV calculation
226 ### framework above. The probability of choosing different actions
    basically acts like a multichoice logit function.
227 choice_prob.Estimate <- function(){
228
229   ### Initialize an empty matrix to hold choice probability estimates.
230   p_i <- matrix(0,31,2)
231
232   ### Iterate through all of the possible mileage states , x.
233   for (x in 1:31){

```

```

234     ## For each mileage state x, calculate the probability that Zurcher
      will choose i = 0.
235     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]))
236     ## Calculate the probability of choosing i = 1 at each state, which is
      just 1 - P(i = 0).
237     p_i[x,2] <- 1- p_i[x,1]
238   }
239
240   ### Return the updated p_i object
241   return(p_i)
242 }
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
285   transition components and the choice components.
286   total <- (log_choice_prob+log_transition_prob) + total
287 }
288 return (total)
289 }
290
291 ### Now we're actually going to use the nested fixed point algorithm to get
292 the maximum likelihood estimates of the parameters
293 that we care about. This process has three steps.
294
295 ### Step 1: We would calculate theta_30, theta_31, and theta_31 directly
296 from the data. This step is not in the loop, and we've
297 actually already done this and it doesn't change, so we don't need to
298 do it again.
299
300 ### Step 2: Next, we are going to set up a grid over values of theta_1,
301 beta, and RC that we will calculate the
302 log likelihood to determine the maximum likelihood parameter values. We
303 'll also initialize a dataframe
304 to hold the parameter values and the log likelihoods.
305
306 theta_1_range <- seq(.01,.10,.01)
307 beta_range <- seq(.90,.99,.01)
308 RC_range <- seq(6,15,1)
309 likelihood <- data.frame( 'theta_1'=rep(0), 'beta'=rep(0), 'RC'=rep(0), 'log.
310 likelihood'=rep(0))
311
312 ### Step 3: Now we actually do the nested fixed point computation.
313
314 ### Loop through theta_1
315 for (theta_1 in theta_1_range) {
316   ### Loop through beta
317   for (beta in beta_range) {
318     ### Loop through RC
319     for (RC in RC_range) {
320       print(paste(c(theta_1, beta, RC), collapse='_'))
321
322       ### Initialize the EV functions to the initial values we used above.
323       EV <- matrix(100,33,2)
324       EV2 <- matrix(0,33,2)
325
326       ### Iteratively compute the EV values.
327       while(max(abs(EV-EV2))>cri){
328         EV <- EV2
329         EV2 <- value.Iterate(EV)

```



```

323     }
324
325     EV <- EV2
326     EV <- EV[1:31,]
327
328     ### Given these values of EV, calculated the choice probabilities
329     p_i <- choice.prob.Estimate()
330
331     ### Given the EV values, the choice probabilities and the parameters,
332     calculate
333     ### the log-likelihood of the data.
334     likelihood <- rbind(likelihood, c(theta_1, beta, RC, log.likelihood.
335     Compute()))
336   }
337 }
338
339 ### Retrieve the row in the likelihood dataframe corresponding to the
340 maximum likelihood estimate
341 likelihood <- likelihood[-1,]
342 parameter_estimates <- likelihood[which.max(likelihood[,4]),]
343
344 ### Use these parameters and get the relevant estimate of EV and p_i
345 theta_1 = parameter_estimates$theta_1
346 beta = parameter_estimates$beta
347 RC = parameter_estimates$RC
348
349 EV <- matrix(100,33,2)
350 EV2 <- matrix(0,33,2)
351 ### Iteratively compute the EV values.
352 while(max(abs(EV-EV2))>cri){
353   EV <- EV2
354   EV2 <- value.Iterate(EV)
355 }
356 EV <- EV2
357 EV <- EV[1:31,]
358 ### Given these values of EV, calculated the choice probabilities
359 p_i <- choice.prob.Estimate()
360 ### Given the EV values, the choice probabilities and the parameters,
361 calculate
362 ### the log-likelihood of the data.
363 likelihood <- rbind(likelihood, c(theta_1, beta, RC, log.likelihood.Compute()))
364
365 save(EV, p_i, likelihood, parameter_estimates, file='rust_estimate.Rdata')
366
367 #####
368 # Question 3.2 #
369 #####
370
371 ### Now we will get estimates of the parameters using the Hotz and Miller
372 conditional choice probability approach. This will
373 ### allow us to compare these parameter estimates to those obtained using
374 the Rust approach.

```

```

371 ##### First, we need to calculate the probability of the agent choosing
372     either i = 0 or i = 1 based on the state that they find
373 ##### a given bus in, x, at some time period t. This will be the baseline
374     that we use to try and find the best parameter values
375 ##### (i.e., which parameter values minimize the infinity norm between these
376     true probabilities and the estimated probabilities)
377
378 ##### The probability matrix
379 p_ix <- matrix(0,33,2)
380 ##### The vector of how often the agent chooses i=1 given state x
381 ones <- vector()
382 ##### The vector of how often the agent finds a bus in state x
383 total <- vector()
384
385 ##### Loop through the states
386 for (state in 0:32){
387
388     ##### For a given state, a will track how many times i = 1 and b will track
389         how many times that state occurs.
390     ##### Initialize them to 0 for the given state.
391     a <- 0
392     b <- 0
393
394     ##### Loop over the buses
395     for (bus in 1:100){
396
397         ##### Increment how many times the agent chooses i = 1 in state x
398         a <- sum(i[which(x[,bus]==state),bus]) + a
399         ##### Increment how many times the state x occurs
400         b <- length(i[which(x[,bus]==state),bus]) + b
401     }
402
403     ##### Add the most recent estimates to the vector.
404     ones[state+1] <- a
405     total[state+1] <- b
406 }
407
408 ##### Based on the ones and total vectors, updated the choice probability
409     matrix.
410 p_ix[,1] <- 1-ones/total
411 p_ix[,2] <- ones/total
412
413 ##### Plot conditional choice probabilities
414 p_ix_df <- as.data.frame(p_ix)
415 p_ix_df$state <- as.numeric(rownames(p_ix_df))
416 names(p_ix_df) <- c('P(i=0)', 'P(i=1)', 'State')
417 ccp_plot <- p_ix_df %>%
418     ggplot(., aes(x=State, y='P(i = 1)')) + geom_line() +
419     ggtitle('Conditional_probability_of_engine_replacement_n_as_a_function_
420         of_mileage') +
421     theme(plot.title = element_text(hjust = 0.5))
422 ggsave(ccp_plot, file='ccp_plot.png', height=4, width=4, units='in')

```

```

418 ##### The function below uses the Hotz and Miller method to estimate V and p_
      ix_hat for every state and period
419 ##### given a set of model parameters (beta, theta_1, and RC).
420 approximate.V_pihat <- function() {
421   ##### Initialize an empty valuation matrix
422   V <- matrix(0,33,2)
423   ##### Initialize an empty conditional choice probability matrix
424   p_ix_hat <- matrix(0,33,2)
425
426   ##### Iterate through the states
427   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
457       +1,2]))
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
466   algorithm:
467   ##### S: The number of "simulations" to do per state / decision
468   ##### gamma: This should be Euler's constant
469   ##### theta_1_range: The range of theta_1 values to test
470   ##### beta_range: The range of beta values to test
471   ##### RC_range: The range of RC values to test
472   S = 1000
473   theta_1_range <- seq(.01,.10,.01)
474   beta_range <- seq(.90,.99,.01)
475   RC_range <- seq(6,15,1)
476
477   ##### Initialize a dataframe to hold different parameter combinations and the
478   infinity-norm between the actual conditional
479   choice probabilities and the estimated ones
480   difference <- data.frame('theta_1'=rep(0),'beta'=rep(0),'RC'=rep(0),'
481     difference'=rep(0))
482
483   ##### Loop through theta_1
484   for (theta_1 in theta_1_range) {
485     ##### Loop through theta_2
486     for (beta in beta_range) {
487       ##### Loop through RC
488       for (RC in RC_range) {
489         # Check progress
490         print(paste(c(theta_1, beta, RC), collapse='_'))
491
492         ##### Get estimates of V and P_ix_hat using the Hotz and Miller method
493         v_and_p_ix_hat <- approximate.V_pixhat()
494         V = v_and_p_ix_hat$V
495         p_ix_hat <- v_and_p_ix_hat$p_ix_hat
496
497         ##### Now that we have a full conditional choice probability matrix,
498         calculate the infinity norm (i.e., largest
499         absolute difference between the empirical conditional choice
500         probabilities and those estimated with the
501         ##### given parameters)
502         difference <- rbind(difference, c(theta_1, beta, RC, max(abs(p_ix[1:31,] -
503           p_ix_hat[1:31,]))))
504       }
505     }
506   }

```

```

502 ### Find the set of parameters that minimizes this difference
503 difference <- difference[-1,]
504 parameter_estimates <- difference[which.min(difference[,4]),]
505
506 ### Use these parameters and get the relevant estimate of V and p_ix_hat
507 theta_1 = parameter_estimates$theta_1
508 beta = parameter_estimates$beta
509 RC = parameter_estimates$RC
510 best_guesses <- approximate.V_pixhat()
511 V <- best_guesses$V
512 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
521
522 #####
523 # Question 3.4 #
524 #####
525
526 ### This function simulates, for one agent, a sequence of state transitions
    and also engine replacement decisions
527 simulate_sequence <- function(n_periods) {
528   ### Initialize empty vectors to hold states and engine replacement
    transitions
529   x_values <- rep(0, n_periods)
530   i_values <- rep(0, n_periods)
531   ### Every bus starts at state 0
532   x_values[1] <- 0
533   ### Go through the progression
534   for (j in 1:length(x_values)) {
535     ### Make a decision based on current state
536     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]))
537     ### If decision is to not replace, continue on and increment x randomly
538     if (i_values[j] == 0) {
539       x_values[j+1] = x_values[j] + sample(c(0,1,2),1,replace = T, prob = c
    (theta_30,theta_31,theta_32))
540     } else {
541       x_values[j+1] = 0
542     }
543   }
544 }
545 ## Generate a decision for the last period, even though we never see the
    fruits of that decision
546 i_values[length(i_values)] = sample(c(0,1),1,replace=T,prob = c(p_ix_hat[
    x_values[length(x_values)] + 1,1],
547                                     p_ix_hat[
    x_

```

```

values
[
length
(x_
values
)] +
1,2])
)

548   ### Return the states and replacement decisions in a list
549   results <- list('x_values' = x_values, 'i_values' = i_values)
550   return(results)
551 }
552
553   ### Given a set of parameters, this function generates period-by-period
554   demand estimates for new buses (e.g.,
555   ### how many buses will get their engine replaced in each period)
556   estimate_demand <- function(n_sims, n_buses, n_periods) {
557
558     ### Initialize a vector to hold simulated demand
559     simulated_demand_total <- rep(0, n_periods)
560
561     ### Run a bunch of simulations and simulate engine replacement decisions
562     for (j in 1:n_sims) {
563       simulated_demand_total = simulated_demand_total + simulate_sequence(n_
564         periods)$i_values
565     }
566
567     ### Divide by the number of sims to get averages, multiply by number of
568     buses (this works because
569     ### buses are independent). Then return what we get.
570     return((n_buses/n_sims)*simulated_demand_total)
571   }
572
573   ### Get demand as a function of RC for the first bus
574
575   ## Specify the range of RCs, as well as constants.
576   RC_range = seq(0, 15, .25)
577   n_periods = 15
578   n_sims = 1000
579   n_buses = 100
580   load('hotz_and_miller_estimate.Rdata')
581
582   ## Initialize an empty dataframe to hold results
583   estimated_demand_df <- data.frame(time_period = c(),
584                                     RC = c(),
585                                     demand = c(),
586                                     engine = c())
587
588   # Loop through the RCs, then estimate the probabilities using the Rust
589   method, then do simulation.
590   for (j in RC_range) {
591     RC = j

```

```

589     ### Set a critical value for to measure the deviation between iterative
        updates of EV. The distance between the two EV matrices
590     ### is the infinity norm of the difference
591     cri <- 10^(-8)
592
593     ### Set an initial value for the EV matrix (all 0s, EV), and another EV
        object to hold the updated estimates, EV2.
594     EV <- matrix(100,33,2)
595     EV2 <- matrix(0,33,2)
596
597     ## While the infinity norm is less than the threshold, iterate
598     while(max(abs(EV-EV2))>cri){
599
600         ### Set the current EV to the previous updated EV
601         EV <- EV2
602         ### Compute a new updated EV by iterating on the current EV
603         EV2 <- value.Iterate(EV)
604     }
605
606     ### Do one last update to set EV equal to the last EV2
607     EV <- EV2
608
609     # get EV(x,i) for x=0,1,2,...,30
610     ### EV contains extra states, which we needed to compute the above
        computation. Throw them away.
611     EV <- EV[1:31,]
612
613     ### Get estimated probability based on the above EV
614     p_ix_hat <- choice.prob.Estimate()
615     ### Estimate demand using that probability
616     estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)
617
618     ### Add this estimate to a temp dataframe
619     estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
        ,
620                                           RC = rep(RC, n_periods),
621                                           demand = estimated_demand,
622                                           engine = rep('Engine_1', n_periods))
623     ### Collate temp dataframe to full dataframe
624     estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
        temp)
625
626 }
627
628 ### Reset theta_1 to the "new engine", redo the exercise above.
629 theta_1 = .02
630
631 ### Loop through RCs
632 for (j in RC_range) {
633     RC = j
634
635     ### Set a critical value for to measure the deviation between iterative
        updates of EV. The distance between the two EV matrices
636     ### is the infinity norm of the difference

```

```

637 cri <- 10^(-8)
638
639 ### Set an initial value for the EV matrix (all 0s, EV), and another EV
640     object to hold the updated estimates, EV2.
641 EV <- matrix(100,33,2)
642 EV2 <- matrix(0,33,2)
643
644 ## While the infinity norm is less than the threshold, iterate
645 while(max(abs(EV-EV2))>cri){
646     ### Set the current EV to the previous updated EV
647     EV <- EV2
648     ### Compute a new updated EV by iterating on the current EV
649     EV2 <- value.Iterate(EV)
650 }
651
652 ### Do one last update to set EV equal to the last EV2
653 EV <- EV2
654
655 # get EV(x,i) for x=0,1,2,...,30
656 ### EV contains extra states, which we needed to compute the above
657     computation. Throw them away.
658 EV <- EV[1:31,]
659
660 ### Get probability estimates based on EV
661 p_ix_hat <- choice.prob.Estimate()
662 ### Estimate demand
663 estimated_demand <- estimate_demand(n_sims, n_buses, n_periods)
664
665 ### Add to temp dataframe
666 estimated_demand_df_temp <- data.frame(time_period = seq(1, n_periods, 1)
667     ,
668     RC = rep(RC, n_periods),
669     demand = estimated_demand,
670     engine = rep('Engine_2', n_periods)
671 )
672
673 ### Collate to full dataframe
674 estimated_demand_df <- rbind(estimated_demand_df, estimated_demand_df_
675     temp)
676 }
677
678 ### For a reduced set of RCs, see the period-by-period demand
679 per_period_demand_plot <- estimated_demand_df %>%
680     filter(RC %in% c(1, 3, 5, 7, 10)) %>%
681     mutate(RC = as.factor(RC)) %>%
682     ggplot(., aes(x=time_period, y=demand, color=RC)) + geom_line() +
683     facet_wrap(~engine) + xlab('Period') + ylab('Demand_for_Engines') +
684     ggtitle('Demand_for_engines_over_time') +
685     theme(plot.title = element_text(hjust = 0.5))
686 ggsave(per_period_demand_plot, file='per_period_demand_plot.png', height=4,
687     width=6, units='in')

```



```

683 ### Aggregate over periods to get demand as a function of RC for different
    thetas.
684 aggregate_demand_plot <- estimated_demand_df %>%
685   group_by(RC, engine) %>%
686   summarise(total_demand = sum(demand)/n_periods) %>%
687   ungroup() %>% head()
688   ggplot(., aes(x=RC, y=total_demand, color=engine)) + geom_line() + xlab('
    RC') +
689   ylab('Average_per-period_demand') + ggtitle('Average_per-period_engine_
    demand_for_100_buses') +
690   ggsave(aggregate_demand_plot, file='aggregate_demand_plot.png', height=4,
    width=6, units='in')
691
692
693 #####
694 # Question 3.5 #
695 #####

```

Listing 1: ./rust.R