

Econ 753 Assignment 3

Katherine Fairley

1. The sequence problem for the firm is:

Choose $i = \{i_1, i_2, \dots\}$ where $\forall t, \quad i_t \in \{0, 1\}$

to solve:

$$\begin{aligned} & \max_i \mathbb{E} \left[\sum_{t=1}^{\infty} \beta^{t-1} \Pi(a_t, i_t, \varepsilon_{0t}, \varepsilon_{1t}) \right] \\ \iff & \max_i \mathbb{E} \left[\sum_{t=1}^{\infty} \beta^{t-1} [i_t \times (R + \varepsilon_{1t}) + (1 - i_t) \times (\mu a_t + \varepsilon_{0t})] \right] \end{aligned}$$

2. The Bellman equation is given by:

$$\begin{aligned} V(a, \varepsilon_0, \varepsilon_1) &= \max_{i \in \{0,1\}} \Pi(a, i, \varepsilon_0, \varepsilon_1) + \beta \mathbb{E} [V(a', \varepsilon'_0, \varepsilon'_1)] \\ &= \max \left\{ (R + \varepsilon_1) + \beta \mathbb{E} [V(1, \varepsilon'_0, \varepsilon'_1)] \quad , \quad (\mu a + \varepsilon_0) + \beta \mathbb{E} [V(\min\{5, a + 1\}, \varepsilon'_0, \varepsilon'_1)] \right\} \\ &= \max \left\{ \bar{V}_1(a, \varepsilon_0, \varepsilon_1; \theta) \quad , \quad \bar{V}_0(a, \varepsilon_0, \varepsilon_1; \theta) \right\} \end{aligned}$$

Define:

$$\Pi(a, i) = \begin{cases} R & \text{if } i = 1 \\ \mu a & \text{if } i = 0 \end{cases}$$

Then the Bellman equation can be expressed as a function of only the observed state, a :

$$\begin{aligned} V(a) &= \max_{i \in \{0,1\}} \{ \Pi(a, i) + \beta \mathbb{E} [V(a')] \} \\ &= \max \{ R + \beta V(1) \quad , \quad \mu a + \beta V(\min\{5, a + 1\}) \} \\ &= \max \{ \bar{V}_1(a; \theta) \quad , \quad \bar{V}_0(a; \theta) \} \end{aligned}$$

3. (a) The firm is indifferent between replacing and not when:

$$\begin{aligned} R + \varepsilon_1 + \beta \mathbb{E}_{\varepsilon'_0, \varepsilon'_1} [V(1)] &= 2\mu + \varepsilon_0 + \beta \mathbb{E}_{\varepsilon'_0, \varepsilon'_1} [V(3)] \\ \implies \varepsilon_0 - \varepsilon_1 &= R - 2\mu + \beta \left(\mathbb{E}_{\varepsilon'_0, \varepsilon'_1} [V(1)] - \mathbb{E}_{\varepsilon'_0, \varepsilon'_1} [V(3)] \right) \end{aligned}$$

Where $\mathbb{E}_{\varepsilon'_0, \varepsilon'_1} [V(a')] \equiv EV(a')$ is the expected value function.

Using the log-sum formula, when $\mu = -1$, $R = -3$, and $\beta = 0.9$, the expected value function is:

$$EV(a) = \log[\exp(-3 + 0.9 \cdot EV(1)) + \exp(-a + 0.9 \cdot EV(\min\{5, a + 1\}))] + \gamma$$

By evaluating this expression for each $a \in \{1, 2, 3, 4, 5\}$ and solving the resulting system of equations, I obtain:

$$EV(1) = -9.3362$$

$$EV(3) = -10.5746$$

$$EV(5) = -10.7895$$

This gives:

$$\varepsilon_0 - \varepsilon_1 = 0.1145$$

The probability the firm will replace the machine is:

$$P(i = 1|a = 2; \theta) = \frac{\exp(R + \beta \mathbb{E}_{\varepsilon'_0, \varepsilon'_1}[V(1)])}{\exp(-2\mu + \beta \mathbb{E}_{\varepsilon'_0, \varepsilon'_1}[V(3)]) + \exp(R + \beta \mathbb{E}_{\varepsilon'_0, \varepsilon'_1}[V(1)])}$$

and so at the given parameter values, this probability is given by:

$$P(i = 1|a = 2; \mu = -1, R = -3, \beta = 0.9) = 0.5286$$

The value of the firm is given by:

$$V(a, \varepsilon_0, \varepsilon_1) = \max \left\{ (R + \varepsilon_1) + \beta \mathbb{E} [V(1, \varepsilon'_0, \varepsilon'_1)] \quad , \quad (\mu a + \varepsilon_0) + \beta \mathbb{E} [V(\min\{5, a + 1\}, \varepsilon'_0, \varepsilon'_1)] \right\}$$

Evaluated at $a = 4$, $\varepsilon_0 = 1$, $\varepsilon_1 = 1.5$, $\mu = -1$, $R = -3$, and $\beta = 0.9$, this gives:

$$V(4, 1, 1.5) = \max \{-1.5 + 0.9 \cdot EV(1) \quad , \quad -3 + 0.9 \cdot EV(5)\}$$

and substituting in the values of $EV(1)$ and $EV(5)$ gives:

$$V(4, 1, 1.5) = \max \left\{ \underbrace{-9.9026}_{\text{replace}}, \quad \underbrace{-12.7105}_{\text{don't}} \right\} = -9.9026$$

```

1
2 % Assignment 3 Econ 753
3 % Set working directory
4 cd('C:\Users\KatyK\University of Michigan Dropbox\Katherine Fairley\
   Notability\Spring 2025\753 - Methods\Github')
5
6
7 %% #3.a.
8 % Parameters
9 mu = -1;      % Cost of machine operation per age unit
10 R = -3;      % Cost of replacement
11 beta = 0.9;  % Discount factor
12 gamma = 0.5772156649; % Euler's constant
13
14 % Create symbolic variables for the expected value functions
15 syms EV1 EV2 EV3 EV4 EV5 real
16
17 % Set up the system of equations
18 % EV(a) = log[exp(R + beta*EV(1)) + exp(-a + beta*EV(min(5,a+1)))] +
   gamma
19
20 eq1 = EV1 == log(exp(R + beta*EV1) + exp(-1 + beta*EV2)) + gamma;
21 eq2 = EV2 == log(exp(R + beta*EV1) + exp(-2 + beta*EV3)) + gamma;
22 eq3 = EV3 == log(exp(R + beta*EV1) + exp(-3 + beta*EV4)) + gamma;
23 eq4 = EV4 == log(exp(R + beta*EV1) + exp(-4 + beta*EV5)) + gamma;
24 eq5 = EV5 == log(exp(R + beta*EV1) + exp(-5 + beta*EV5)) + gamma;
25
26 % Solve the system of equations
27 eqs = [eq1, eq2, eq3, eq4, eq5];
28 vars = [EV1, EV2, EV3, EV4, EV5];
29 solution = vpasolve(eqs, vars);
30
31 % Extract and display the solutions with 4 decimal places
32 fprintf('Algebraic solution for expected value functions:\n');
33 EV = zeros(5,1);
34 for a = 1:5
35     varName = sprintf('EV%d', a);
36     EV(a) = double(solution.(varName));
37     fprintf('EV(%d) = %.4f\n', a, EV(a));
38 end
39
40 %Value of epsilon_0 - epsilon_1 for indifference when a = 2
41 indiff_threshold = (R - (-2)) + beta * (EV(1) - EV(3));
42 fprintf('\nQuestion 1: Indifference threshold when a(t) = 2\n');
43 fprintf('epsilon_0 - epsilon_1 = %.4f\n', indiff_threshold);
44

```

```

45 % Probability of replacement when a = 2
46 v_1 = R + beta * EV(1);
47 v_0 = -2 + beta * EV(3);
48 prob_replace = 1 / (1 + exp(v_0 - v_1));
49 fprintf('\nQuestion 2: Probability of replacement when a(t) = 2\n');
50 fprintf('P(i(t)=1|a(t)=2) = %.4f\n', prob_replace);
51
52 % Value of the firm at a = 4, epsilon_0 = 1, epsilon_1 = 1.5
53 v_keep_4 = -4 + 1 + beta * EV(5);
54 v_replace_4 = R + 1.5 + beta * EV(1);
55 value_4 = max(v_keep_4, v_replace_4);
56 fprintf('\nQuestion 3: Value at a(t) = 4, epsilon_0 = 1, epsilon_1 =
    1.5\n');
57 fprintf('V(a(t)=4, epsilon_0=1, epsilon_1=1.5) = %.4f\n', value_4);
58 if v_replace_4 > v_keep_4
59     fprintf('Optimal action: Replace\n');
60 else
61     fprintf('Optimal action: Keep\n');
62 end
63
64
65
66 %% Set parameter values
67 theta = [-1, -3]; % [mu, R]
68 beta = 0.9; % Discount factor
69
70 %% #3 - Value Function Iteration
71 EV0 = zeros(10,1); % Set initial guess of expected value function
72
73 % Define transition probability matrix - modified for correct state
    representation
74 prob = zeros(10, 5);
75 % First 5 rows for ages 1-5 with no replacement (state transitions)
76 prob(1,1) = 1; % age 1 -> 2
77 prob(2,2) = 1; % age 2 -> 3
78 prob(3,3) = 1; % age 3 -> 4
79 prob(4,4) = 1; % age 4 -> 5
80 prob(5,5) = 1; % age 5 stays 5
81 % Next 5 rows for replacement (always transitions to age 1)
82 prob(6:10,1) = 1;
83
84 % Value function iteration
85 EV1 = inner(theta, beta, EV0, prob); % Call function to get first
    iteration
86
87 tolerance = 1e-6; % Set tolerance level for convergence
88 iterations = 0;

```

```

89 max_iter = 1000;
90
91 while norm(EV1 - EV0) >= tolerance && iterations < max_iter
92     EV0 = EV1;
93     EV1 = inner(theta, beta, EV0, prob);
94     iterations = iterations + 1;
95 end
96
97 fprintf('Value function converged after %d iterations\n', iterations)
98 ;
99 %% #4 - Simulate the Data
100 % Simulate data with parameters R = -3, mu = -1, beta = 0.9
101 [sim_data, value_function, policy_function] =
102     simulateMachineReplacement(-3, -1, 0.9, 20000);
103
104 %% #5 - Run NFP Estimation
105 % Extract only the required columns for rustNFP from data table
106 estimation_data = sim_data(:, {'period', 'age', 'replace'});
107
108 % Run NFP estimation
109 [estimates, std_errors, loglik, iterations] = rustNFP(estimation_data
110     , beta);
111
112 % Compare true vs estimated parameters
113 fprintf('\nComparison of True vs. Estimated Parameters:\n');
114 fprintf('R: True = %.4f, Estimated = %.4f (SE: %.4f)\n', -3,
115     estimates(1), std_errors(1));
116 fprintf('mu: True = %.4f, Estimated = %.4f (SE: %.4f)\n', -1,
117     estimates(2), std_errors(2));
118 fprintf('Log-likelihood: %.4f\n', loglik);
119 fprintf('Number of function evaluations: %d\n', iterations);
120
121 %% #6 - Run CCP Estimation
122 % Run CCP estimation
123 [estimates, std_errors, loglik] = hotzMillerCCP(estimation_data, beta
124     );
125
126 %% Inner Function for Value Function Iteration (#3)
127 function EV1 = inner(theta, beta, EV0, prob)
128     mu = theta(1); % Extract mu from parameter vector
129     R = theta(2); % Extract R from parameter vector
130
131     SS = [1,2,3,4,5]'; % Define state space (vector of possible
132         values of a)
133
134     % Euler's constant (approximately 0.5772)

```

```

129     gamma = 0.5772156649;
130
131     % Calculate the expected value for keep and replace options
132     ll = log(exp(mu*SS + beta*EV0(1:5)) + exp(R*ones(5,1) + beta*EV0
        (6:10)));
133
134     % Add Euler's constant and multiply by transition probabilities
135     EV1 = (prob*ll) + gamma*ones(10,1);
136 end
137
138 %% Data Simulation Functions (#4)
139 function [data, value_function, policy_function] =
    simulateMachineReplacement(R, mu, beta, T)
140     % Maximum machine age
141     maxAge = 5;
142
143     % Step 1: Solve the dynamic programming problem
144     [value_function, policy_function] = solveDP(R, mu, beta, maxAge);
145
146     % Step 2: Simulate decisions based on the solved policy function
147     data = simulateDecisions(policy_function, R, mu, T, maxAge);
148 end
149
150 function [value_function, policy_function] = solveDP(R, mu, beta,
    maxAge)
151     % Setup state space: machine ages from 1 to maxAge
152     states = 1:maxAge;
153     n_states = length(states);
154
155     % Initialize value function
156     value_function = zeros(n_states, 1);
157     new_value_function = zeros(n_states, 1);
158     policy_function = zeros(n_states, 1);
159
160     % Value function iteration
161     max_iter = 10000;
162     tolerance = 1e-6;
163     converged = false;
164     iter = 0;
165
166     % Euler's constant
167     gamma = 0.5772156649;
168
169     while ~converged && iter < max_iter
170         iter = iter + 1;
171
172         for i = 1:n_states

```

```

173     % Current state (machine age)
174     a = states(i);
175
176     % Option 0: Keep the machine
177     % Flow utility from keeping
178     u0 = mu * a;
179
180     % Expected future value if keeping
181     if a < maxAge
182         future_state = a + 1;
183     else
184         future_state = a; % Age stays at maxAge
185     end
186
187     v0 = u0 + beta * value_function(future_state);
188
189     % Option 1: Replace the machine
190     % Flow utility from replacing
191     u1 = R;
192
193     % Future state is always 1 (new machine)
194     v1 = u1 + beta * value_function(1);
195
196     % Compute choice-specific value functions
197     v = [v0, v1];
198
199     % Compute expected value using the "log-sum-exp" formula
200     % for Type-1 EV
201     % Include Euler's constant in the calculation
202     new_value_function(i) = log(exp(v0) + exp(v1)) + gamma;
203
204     % Compute choice probabilities
205     probab_replace = exp(v1) / (exp(v0) + exp(v1));
206
207     % Store the policy (probability of replacement)
208     policy_function(i) = probab_replace;
209
210     end
211
212     % Check for convergence
213     diff = max(abs(new_value_function - value_function));
214     if diff < tolerance
215         converged = true;
216     end
217
218     % Update value function
219     value_function = new_value_function;
220 end

```

```

219
220     if ~converged
221         warning('Value function iteration did not converge after %d
                iterations', max_iter);
222     end
223 end
224
225 function data = simulateDecisions(policy_function, R, mu, T, maxAge)
226     % Initialize data structure
227     data = struct('period', {}, 'age', {}, 'replace', {}, 'epsilon0',
                {}, 'epsilon1', {});
228
229     % Initialize machine age (start with a new machine)
230     age = 1;
231
232     % Simulate decisions over time
233     for t = 1:T
234         % Draw extreme value errors
235         epsilon0 = -log(-log(rand())); % Type-1 extreme value
236         epsilon1 = -log(-log(rand())); % Type-1 extreme value
237
238         % Compute deterministic components of utility
239         u0 = mu * age;
240         u1 = R;
241
242         % Compute total utilities including random components
243         total_u0 = u0 + epsilon0;
244         total_u1 = u1 + epsilon1;
245
246         % Make replacement decision based on utilities
247         replace = (total_u1 > total_u0);
248
249         % Policy based on age - this is for comparison
250         policy_prob = policy_function(age);
251
252         % Store the data
253         idx = length(data) + 1;
254         data(idx).period = t;
255         data(idx).age = age;
256         data(idx).replace = replace;
257         data(idx).epsilon0 = epsilon0;
258         data(idx).epsilon1 = epsilon1;
259         data(idx).policy_prob = policy_prob;
260
261         % Update machine age based on decision
262         if replace
263             age = 1; % New machine

```



```

264         else
265             age = min(maxAge, age + 1); % Age the machine
266         end
267     end
268
269     % Convert struct to table for easier analysis
270     data = struct2table(data);
271 end
272
273 %% Nested Fixed Point (NFP) Estimation Functions (#5)
274 function [estimates, std_errors, loglik, iterations] = rustNFP(data,
    beta)
275     % Input:
276     %     data - table with columns: period, age, replace
277     %     beta - discount factor (fixed in estimation)
278
279     % Output:
280     %     estimates - estimated parameters [R; mu]
281     %     std_errors - standard errors of the estimates
282     %     loglik - log likelihood at the optimum
283     %     iterations - number of function evaluations
284
285     % Start with initial parameter guess
286     % Define initial values for the structural parameters close to
        true values
287     theta0 = [-4; -0.5]; % Initial guess for [R; mu]
288
289     % Set maximum age
290     maxAge = 5;
291
292     % Setup optimization options for the parameter search
293     options = optimset('Display', 'iter', 'TolFun', 1e-6, 'TolX', 1e
        -6, 'MaxFunEvals', 1000);
294
295     % Define objective function for optimization (negative log
        likelihood)
296     objFun = @(theta) negLogLikelihood(theta, data, beta, maxAge);
297
298     % Run optimization to find parameter estimates
299     [theta_hat, fval, ~, output] = fminunc(objFun, theta0, options);
300
301     % Extract results
302     estimates = theta_hat;
303     loglik = -fval; % Convert back to positive log likelihood
304     iterations = output.funcCount;
305
306     % Compute standard errors using numerical Hessian

```

```

307 H = numHessian(objFun, theta_hat);
308 std_errors = sqrt(diag(inv(H)));
309
310 % Display results
311 fprintf('Estimation Results:\n');
312 fprintf('R: %.4f (SE: %.4f)\n', estimates(1), std_errors(1));
313 fprintf('mu: %.4f (SE: %.4f)\n', estimates(2), std_errors(2));
314 fprintf('Log-likelihood: %.4f\n', loglik);
315 fprintf('Number of function evaluations: %d\n', iterations);
316 end
317
318 % Negative Log-Likelihood Function
319 function [nll, grad] = negLogLikelihood(theta, data, beta, maxAge)
320     % Extract parameters
321     R = theta(1);
322     mu = theta(2);
323
324     % Solve for conditional value functions using contraction mapping
325     [V0, V1] = solveValueFunctions(R, mu, beta, maxAge);
326
327     % Compute choice probabilities using logit formula
328     loglik = 0;
329     for i = 1:height(data)
330         age = data.age(i);
331         replace = data.replace(i);
332
333         % Compute choice probability using logit formula
334         prob_replace = exp(V1(age)) / (exp(V0(age)) + exp(V1(age)));
335
336         % Add log-likelihood contribution
337         if replace
338             loglik = loglik + log(prob_replace);
339         else
340             loglik = loglik + log(1 - prob_replace);
341         end
342     end
343
344     % Return negative log-likelihood (for minimization)
345     nll = -loglik;
346
347     % No analytical gradient provided
348     grad = [];
349 end
350
351 % Solve for value functions using contraction mapping
352 function [V0, V1] = solveValueFunctions(R, mu, beta, maxAge)
353     % Initialize value functions

```

```

354 V = zeros(maxAge, 1);
355 V_new = zeros(maxAge, 1);
356
357 % Value function iteration parameters
358 max_iter = 1000;
359 tolerance = 1e-8;
360
361 % Euler's constant
362 gamma = 0.5772156649;
363
364 % Contraction mapping (fixed point iteration)
365 for iter = 1:max_iter
366     % For each state (machine age)
367     for a = 1:maxAge
368         % Option 0: Keep the machine
369         % Deterministic utility from keeping
370         u0 = mu * a;
371
372         % Next state if keeping
373         next_a = min(a + 1, maxAge);
374
375         % Value of keeping
376         v0 = u0 + beta * V(next_a);
377
378         % Option 1: Replace the machine
379         % Deterministic utility from replacing
380         u1 = R;
381
382         % Next state if replacing (always a new machine)
383         % Value of replacing
384         v1 = u1 + beta * V(1);
385
386         % Compute the expected value function (log-sum-exp
387             formula for Type 1 EV)
388         % Include Euler's constant in the calculation
389         V_new(a) = log(exp(v0) + exp(v1)) + gamma;
390     end
391
392     % Check for convergence
393     if max(abs(V - V_new)) < tolerance
394         break;
395     end
396
397     % Update value function
398     V = V_new;
399 end

```

```

400 % Compute the choice-specific value functions for return
401 V0 = zeros(maxAge, 1);
402 V1 = zeros(maxAge, 1);
403
404 for a = 1:maxAge
405     % Option 0: Keep
406     u0 = mu * a;
407     next_a = min(a + 1, maxAge);
408     V0(a) = u0 + beta * V(next_a);
409
410     % Option 1: Replace
411     u1 = R;
412     V1(a) = u1 + beta * V(1);
413 end
414 end
415
416 % Numerical Hessian computation
417 function H = numHessian(func, x)
418     % Compute numerical Hessian matrix using finite differences
419     k = length(x);
420     H = zeros(k, k);
421     h = 1e-5; % Step size
422
423     for i = 1:k
424         for j = 1:k
425             x1 = x;
426             x1(i) = x(i) + h;
427             x1(j) = x(j) + h;
428
429             x2 = x;
430             x2(i) = x(i) + h;
431
432             x3 = x;
433             x3(j) = x(j) + h;
434
435             x4 = x;
436
437             f1 = func(x1);
438             f2 = func(x2);
439             f3 = func(x3);
440             f4 = func(x4);
441
442             H(i,j) = (f1 - f2 - f3 + f4) / (h^2);
443         end
444     end
445 end
446

```

```

447 %% Functions for #6
448
449 % Implementation of Hotz and Miller (1993) and Hotz et al. (1994)
    approach
450 function [estimates, std_errors, loglik] = hotzMillerCCP(data, beta)
451     % Input:
452     %   data - table with columns: period, age, replace
453     %   beta - discount factor (fixed in estimation)
454
455     % Output:
456     %   estimates - estimated parameters [R; mu]
457     %   std_errors - standard errors of the estimates
458     %   loglik - log likelihood at the optimum
459
460     % Maximum machine age
461     maxAge = 5;
462
463     % (a) Estimate replacement probabilities non-parametrically
464     % Calculate the empirical replacement probability for each state
        (age)
465     P_hat = estimateReplacementProbabilities(data, maxAge);
466
467     % (b.i.) Construct conditional state transition matrices
468     [F0, F1] = constructTransitionMatrices(maxAge);
469
470     % Setup optimization for parameter search
471     options = optimset('Display', 'iter', 'TolFun', 1e-8, 'TolX', 1e
        -8, 'MaxIter', 1000);
472
473     % Initial parameter values - starting at the true values
474     theta0 = [-4; -0.5]; % Initial guess for [R; mu]
475
476     % Define objective function for optimization using a simplified
        approach
477     objFun = @(theta) simplifiedCCPLogLikelihood(theta, data, beta,
        maxAge, P_hat, F0, F1);
478
479     % Run the optimization
480     [theta_hat, fval, ~, ~, ~, hessian] = fminunc(objFun, theta0,
        options);
481
482     % Extract results
483     estimates = theta_hat;
484     loglik = -fval; % Convert back to positive log likelihood
485
486     % Compute standard errors
487     std_errors = sqrt(diag(inv(hessian)));

```

```

488
489 % Display results
490 fprintf('CCP Estimation Results:\n');
491 fprintf('R: %.4f (SE: %.4f)\n', estimates(1), std_errors(1));
492 fprintf('mu: %.4f (SE: %.4f)\n', estimates(2), std_errors(2));
493 fprintf('Log-likelihood: %.4f\n', loglik);
494 end
495
496 % (a) Function to estimate replacement probabilities
497 function P_hat = estimateReplacementProbabilities(data, maxAge)
498 % Initialize array for replacement probabilities
499 P_hat = zeros(maxAge, 1);
500
501 % Calculate empirical probability of replacement at each age
502 for a = 1:maxAge
503     % Get all observations with current age
504     idx = data.age == a;
505
506     if sum(idx) > 0
507         % Calculate empirical replacement probability
508         P_hat(a) = mean(data.replace(idx));
509
510         % Add a small epsilon to avoid 0 or 1 probabilities
511         % which would cause problems in the logit inversion
512         epsilon = 1e-6;
513         P_hat(a) = max(min(P_hat(a), 1-epsilon), epsilon);
514     else
515         % No observations for this age, use a default or
516         % interpolate
517         warning('No observations found for age %d. Using default
518         value.', a);
519         P_hat(a) = 0.2; % Default value
520     end
521 end
522
523 fprintf('Estimated replacement probabilities:\n');
524 for a = 1:maxAge
525     fprintf('Age %d: %.4f\n', a, P_hat(a));
526 end
527
528 % (b) Function to construct conditional transition matrices
529 function [F0, F1] = constructTransitionMatrices(maxAge)
530 % F0: Transition matrix if the machine is kept (action = 0)
531 F0 = zeros(maxAge, maxAge);
532
533 % F1: Transition matrix if the machine is replaced (action = 1)

```

```

533 F1 = zeros(maxAge, maxAge);
534
535 % Fill transition matrices according to the deterministic state
    transition
536 for i = 1:maxAge
537     % If kept, age increases by 1 up to maxAge
538     if i < maxAge
539         F0(i, i+1) = 1;
540     else
541         F0(i, i) = 1; % At max age, stays at max age
542     end
543
544     % If replaced, age becomes 1 regardless of current age
545     F1(i, 1) = 1;
546 end
547 end
548
549 % (c) log-likelihood function
550 function [nll, grad] = simplifiedCCPLogLikelihood(theta, data, beta,
    maxAge, P_hat, F0, F1)
551     % Extract parameters
552     R = theta(1); % Replacement cost
553     mu = theta(2); % Maintenance cost parameter
554
555     % Compute flow utilities
556     flow_u0 = zeros(maxAge, 1); % Utility from keeping
557     flow_u1 = zeros(maxAge, 1); % Utility from replacing
558
559     for a = 1:maxAge
560         flow_u0(a) = mu * a; % Maintenance cost increases with age
561         flow_u1(a) = R; % Replacement cost is constant
562     end
563
564     % Using finite dependence property for the bus engine replacement
        problem
565     % The property allows us to simplify the computation of value
        function differences
566
567     % Initialize value function differences
568     v_diff = zeros(maxAge, 1);
569
570     for a = 1:maxAge
571         % Immediate utility difference
572         v_diff(a) = flow_u1(a) - flow_u0(a);
573
574         % Future value difference using finite dependence
575         % After replacement, the state is always 1 (new machine)

```

```

576         % The future value term is zero because of the finite
           dependence property
577         % The paths from either action converge to the same state
578     end
579
580     % Calculate log likelihood
581     loglik = 0;
582     for i = 1:height(data)
583         age = data.age(i);
584         action = data.replace(i);
585
586         % Calculate probability of replacement using logit
587         prob_replace = 1 / (1 + exp(-v_diff(age)));
588
589         % Ensure numerical stability
590         prob_replace = max(min(prob_replace, 1-1e-10), 1e-10);
591
592         % Add to log likelihood
593         if action
594             loglik = loglik + log(prob_replace);
595         else
596             loglik = loglik + log(1 - prob_replace);
597         end
598     end
599
600     % Return negative log likelihood for minimization
601     nll = -loglik;
602     grad = [];
603 end

```