Econ 753 Assignment 2

Katherine Fairley

1. The sequence problem for the firm is:

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

to solve:

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

2. The Bellman equation is given by:

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

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:

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

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

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

Where $\mathbb{E}_{\varepsilon_0', \varepsilon_1'}\left[V(a')\right] \equiv EV(a')$ 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:

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

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

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

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

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

```
% Assignment 3 Econ 753
% Set working directory
cd('C:\Users\KatyK\University of Michigan Dropbox\Katherine Fairley\Notability\Spring 2025\753 - Methods\Github')


%% #3.a.
% Parameters
mu = -1;      % Cost of machine operation per age unit
R = -3;       % Cost of replacement
beta = 0.9;   % Discount factor
gamma = 0.5772156649; % Euler's constant

% Create symbolic variables for the expected value functions
syms EV1 EV2 EV3 EV4 EV5 real

% Set up the system of equations
% EV(a) = log[exp(R + beta*EV(1)) + exp(-a + beta*EV(min(5,a+1)))] + gamma

eq1 = EV1 == log(exp(R + beta*EV1) + exp(-1 + beta*EV2)) + gamma;
eq2 = EV2 == log(exp(R + beta*EV1) + exp(-2 + beta*EV3)) + gamma;
eq3 = EV3 == log(exp(R + beta*EV1) + exp(-3 + beta*EV4)) + gamma;
eq4 = EV4 == log(exp(R + beta*EV1) + exp(-4 + beta*EV5)) + gamma;
eq5 = EV5 == log(exp(R + beta*EV1) + exp(-5 + beta*EV5)) + gamma;

% Solve the system of equations
eqs = [eq1, eq2, eq3, eq4, eq5];
vars = [EV1, EV2, EV3, EV4, EV5];
solution = vpasolve(eqs, vars);

% Extract and display the solutions with 4 decimal places
fprintf('Algebraic solution for expected value functions:\n');
EV = zeros(5,1);
for a = 1:5
    varName = sprintf('EV%d', a);
    EV(a) = double(solution.(varName));
    fprintf('EV(%d) = %.4f\n', a, EV(a));
end

%Value of epsilon_0 - epsilon_1 for indifference when a = 2
indiff_threshold = (R - (-2)) + beta * (EV(1) - EV(3));
fprintf('\nQuestion 1: Indifference threshold when a(t) = 2\n');
fprintf('epsilon_0 - epsilon_1 = %.4f\n', indiff_threshold);
```

```
% Probability of replacement when a = 2
v_1 = R + beta * EV(1);
v_0 = -2 + beta * EV(3);
prob_replace = 1 / (1 + exp(v_0 - v_1));
fprintf('\nQuestion 2: Probability of replacement when a(t) = 2\n');
fprintf('P(i(t)=1|a(t)=2) = %.4f\n', prob_replace);

% Value of the firm at a = 4, epsilon_0 = 1, epsilon_1 = 1.5
v_keep_4 = -4 + 1 + beta * EV(5);
v_replace_4 = R + 1.5 + beta * EV(1);
value_4 = max(v_keep_4, v_replace_4);
fprintf('\nQuestion 3: Value at a(t) = 4, epsilon_0 = 1, epsilon_1 =
    1.5\n');
fprintf('V(a(t)=4, epsilon_0=1, epsilon_1=1.5) = %.4f\n', value_4);
if v_replace_4 > v_keep_4
    fprintf('Optimal action: Replace\n');
else
    fprintf('Optimal action: Keep\n');
end



%% Set parameter values
theta = [-1, -3];  % [mu, R]
beta = 0.9;        % Discount factor

%% #3 - Value Function Iteration
EV0 = zeros(10,1); % Set initial guess of expected value function

% Define transition probability matrix - modified for correct state
    representation
prob = zeros(10, 5);
% First 5 rows for ages 1-5 with no replacement (state transitions)
prob(1,1) = 1; % age 1 -> 2
prob(2,2) = 1; % age 2 -> 3
prob(3,3) = 1; % age 3 -> 4
prob(4,4) = 1; % age 4 -> 5
prob(5,5) = 1; % age 5 stays 5
% Next 5 rows for replacement (always transitions to age 1)
prob(6:10,1) = 1;

% Value function iteration
EV1 = inner(theta, beta, EV0, prob); % Call function to get first
    iteration

tolerance = 1e-6; % Set tolerance level for convergence
iterations = 0;
```

```
max_iter = 1000;

while norm(EV1 - EV0) >= tolerance && iterations < max_iter
    EV0 = EV1;
    EV1 = inner(theta, beta, EV0, prob);
    iterations = iterations + 1;
end

fprintf('Value function converged after %d iterations\n', iterations);

%% #4 - Simulate the Data
% Simulate data with parameters R = -3, mu = -1, beta = 0.9
[sim_data, value_function, policy_function] = simulateMachineReplacement(-3, -1, 0.9, 20000);

%% #5 - Run NFP Estimation
% Extract only the required columns for rustNFP from data table
estimation_data = sim_data(:, {'period', 'age', 'replace'});

% Run NFP estimation
[estimates, std_errors, loglik, iterations] = rustNFP(estimation_data, beta);

% Compare true vs estimated parameters
fprintf('\nComparison of True vs. Estimated Parameters:\n');
fprintf('R:  True = %.4f, Estimated = %.4f (SE: %.4f)\n', -3, estimates(1), std_errors(1));
fprintf('mu: True = %.4f, Estimated = %.4f (SE: %.4f)\n', -1, estimates(2), std_errors(2));
fprintf('Log-likelihood: %.4f\n', loglik);
fprintf('Number of function evaluations: %d\n', iterations);

%% #6 - Run CCP Estimation
% Run CCP estimation
[estimates, std_errors, loglik] = hotzMillerCCP(estimation_data, beta);

%% Inner Function for Value Function Iteration (#3)
function EV1 = inner(theta, beta, EV0, prob)
    mu = theta(1);   % Extract mu from parameter vector
    R = theta(2);    % Extract R from parameter vector

    SS = [1,2,3,4,5]'; % Define state space (vector of possible values of a)

    % Euler's constant (approximately 0.5772)
```

```
    gamma = 0.5772156649;

    % Calculate the expected value for keep and replace options
    ll = log(exp(mu*SS + beta*EV0(1:5)) + exp(R*ones(5,1) + beta*EV0(6:10)));

    % Add Euler's constant and multiply by transition probabilities
    EV1 = (prob*ll) + gamma*ones(10,1);
end

%% Data Simulation Functions (#4)
function [data, value_function, policy_function] = simulateMachineReplacement(R, mu, beta, T)
    % Maximum machine age
    maxAge = 5;

    % Step 1: Solve the dynamic programming problem
    [value_function, policy_function] = solveDP(R, mu, beta, maxAge);

    % Step 2: Simulate decisions based on the solved policy function
    data = simulateDecisions(policy_function, R, mu, T, maxAge);
end

function [value_function, policy_function] = solveDP(R, mu, beta, maxAge)
    % Setup state space: machine ages from 1 to maxAge
    states = 1:maxAge;
    n_states = length(states);

    % Initialize value function
    value_function = zeros(n_states, 1);
    new_value_function = zeros(n_states, 1);
    policy_function = zeros(n_states, 1);

    % Value function iteration
    max_iter = 10000;
    tolerance = 1e-6;
    converged = false;
    iter = 0;

    % Euler's constant
    gamma = 0.5772156649;

    while ~converged && iter < max_iter
        iter = iter + 1;

        for i = 1:n_states
```

```
            % Current state (machine age)
            a = states(i);

            % Option 0: Keep the machine
            % Flow utility from keeping
            u0 = mu * a;

            % Expected future value if keeping
            if a < maxAge
                future_state = a + 1;
            else
                future_state = a; % Age stays at maxAge
            end

            v0 = u0 + beta * value_function(future_state);

            % Option 1: Replace the machine
            % Flow utility from replacing
            u1 = R;

            % Future state is always 1 (new machine)
            v1 = u1 + beta * value_function(1);

            % Compute choice-specific value functions
            v = [v0, v1];

            % Compute expected value using the "log-sum-exp" formula for Type-1 EV
            % Include Euler's constant in the calculation
            new_value_function(i) = log(exp(v0) + exp(v1)) + gamma;

            % Compute choice probabilities
            prob_replace = exp(v1) / (exp(v0) + exp(v1));

            % Store the policy (probability of replacement)
            policy_function(i) = prob_replace;
        end

        % Check for convergence
        diff = max(abs(new_value_function - value_function));
        if diff < tolerance
            converged = true;
        end

        % Update value function
        value_function = new_value_function;
    end
```

```
    if ~converged
        warning('Value function iteration did not converge after %d
           iterations', max_iter);
    end
end

function data = simulateDecisions(policy_function, R, mu, T, maxAge)
    % Initialize data structure
    data = struct('period', {}, 'age', {}, 'replace', {}, 'epsilon0',
        {}, 'epsilon1', {});

    % Initialize machine age (start with a new machine)
    age = 1;

    % Simulate decisions over time
    for t = 1:T
        % Draw extreme value errors
        epsilon0 = -log(-log(rand()));  % Type-1 extreme value
        epsilon1 = -log(-log(rand()));  % Type-1 extreme value

        % Compute deterministic components of utility
        u0 = mu * age;
        u1 = R;

        % Compute total utilities including random components
        total_u0 = u0 + epsilon0;
        total_u1 = u1 + epsilon1;

        % Make replacement decision based on utilities
        replace = (total_u1 > total_u0);

        % Policy based on age - this is for comparison
        policy_prob = policy_function(age);

        % Store the data
        idx = length(data) + 1;
        data(idx).period = t;
        data(idx).age = age;
        data(idx).replace = replace;
        data(idx).epsilon0 = epsilon0;
        data(idx).epsilon1 = epsilon1;
        data(idx).policy_prob = policy_prob;

        % Update machine age based on decision
        if replace
            age = 1;  % New machine
```

```
        else
            age = min(maxAge, age + 1);  % Age the machine
        end
    end

    % Convert struct to table for easier analysis
    data = struct2table(data);
end

%% Nested Fixed Point (NFP) Estimation Functions (#5)
function [estimates, std_errors, loglik, iterations] = rustNFP(data, beta)
    % Input:
    %   data - table with columns: period, age, replace
    %   beta - discount factor (fixed in estimation)

    % Output:
    %   estimates - estimated parameters [R; mu]
    %   std_errors - standard errors of the estimates
    %   loglik - log likelihood at the optimum
    %   iterations - number of function evaluations

    % Start with initial parameter guess
    % Define initial values for the structural parameters close to true values
    theta0 = [-4; -0.5];  % Initial guess for [R; mu]

    % Set maximum age
    maxAge = 5;

    % Setup optimization options for the parameter search
    options = optimset('Display', 'iter', 'TolFun', 1e-6, 'TolX', 1e-6, 'MaxFunEvals', 1000);

    % Define objective function for optimization (negative log likelihood)
    objFun = @(theta) negLogLikelihood(theta, data, beta, maxAge);

    % Run optimization to find parameter estimates
    [theta_hat, fval, ~, output] = fminunc(objFun, theta0, options);

    % Extract results
    estimates = theta_hat;
    loglik = -fval;  % Convert back to positive log likelihood
    iterations = output.funcCount;

    % Compute standard errors using numerical Hessian
```

```
    H = numHessian(objFun, theta_hat);
    std_errors = sqrt(diag(inv(H)));

    % Display results
    fprintf('Estimation Results:\n');
    fprintf('R: %.4f (SE: %.4f)\n', estimates(1), std_errors(1));
    fprintf('mu: %.4f (SE: %.4f)\n', estimates(2), std_errors(2));
    fprintf('Log-likelihood: %.4f\n', loglik);
    fprintf('Number of function evaluations: %d\n', iterations);
end

% Negative Log-Likelihood Function
function [nll, grad] = negLogLikelihood(theta, data, beta, maxAge)
    % Extract parameters
    R = theta(1);
    mu = theta(2);

    % Solve for conditional value functions using contraction mapping
    [V0, V1] = solveValueFunctions(R, mu, beta, maxAge);

    % Compute choice probabilities using logit formula
    loglik = 0;
    for i = 1:height(data)
        age = data.age(i);
        replace = data.replace(i);

        % Compute choice probability using logit formula
        prob_replace = exp(V1(age)) / (exp(V0(age)) + exp(V1(age)));

        % Add log-likelihood contribution
        if replace
            loglik = loglik + log(prob_replace);
        else
            loglik = loglik + log(1 - prob_replace);
        end
    end

    % Return negative log-likelihood (for minimization)
    nll = -loglik;

    % No analytical gradient provided
    grad = [];
end

% Solve for value functions using contraction mapping
function [V0, V1] = solveValueFunctions(R, mu, beta, maxAge)
    % Initialize value functions
```

```
    V = zeros(maxAge, 1);
    V_new = zeros(maxAge, 1);

    % Value function iteration parameters
    max_iter = 1000;
    tolerance = 1e-8;

    % Euler's constant
    gamma = 0.5772156649;

    % Contraction mapping (fixed point iteration)
    for iter = 1:max_iter
        % For each state (machine age)
        for a = 1:maxAge
            % Option 0: Keep the machine
            % Deterministic utility from keeping
            u0 = mu * a;

            % Next state if keeping
            next_a = min(a + 1, maxAge);

            % Value of keeping
            v0 = u0 + beta * V(next_a);

            % Option 1: Replace the machine
            % Deterministic utility from replacing
            u1 = R;

            % Next state if replacing (always a new machine)
            % Value of replacing
            v1 = u1 + beta * V(1);

            % Compute the expected value function (log-sum-exp
               formula for Type 1 EV)
            % Include Euler's constant in the calculation
            V_new(a) = log(exp(v0) + exp(v1)) + gamma;
        end

        % Check for convergence
        if max(abs(V - V_new)) < tolerance
            break;
        end

        % Update value function
        V = V_new;
    end

```

```
    % Compute the choice-specific value functions for return
    V0 = zeros(maxAge, 1);
    V1 = zeros(maxAge, 1);

    for a = 1:maxAge
        % Option 0: Keep
        u0 = mu * a;
        next_a = min(a + 1, maxAge);
        V0(a) = u0 + beta * V(next_a);

        % Option 1: Replace
        u1 = R;
        V1(a) = u1 + beta * V(1);
    end
end

% Numerical Hessian computation
function H = numHessian(func, x)
    % Compute numerical Hessian matrix using finite differences
    k = length(x);
    H = zeros(k, k);
    h = 1e-5;  % Step size

    for i = 1:k
        for j = 1:k
            x1 = x;
            x1(i) = x(i) + h;
            x1(j) = x(j) + h;

            x2 = x;
            x2(i) = x(i) + h;

            x3 = x;
            x3(j) = x(j) + h;

            x4 = x;

            f1 = func(x1);
            f2 = func(x2);
            f3 = func(x3);
            f4 = func(x4);

            H(i,j) = (f1 - f2 - f3 + f4) / (h^2);
        end
    end
end

```

```matlab
%% Functions for #6

% Implementation of Hotz and Miller (1993) and Hotz et al. (1994) approach
function [estimates, std_errors, loglik] = hotzMillerCCP(data, beta)
    % Input:
    %   data - table with columns: period, age, replace
    %   beta - discount factor (fixed in estimation)

    % Output:
    %   estimates - estimated parameters [R; mu]
    %   std_errors - standard errors of the estimates
    %   loglik - log likelihood at the optimum

    % Maximum machine age
    maxAge = 5;

    % (a) Estimate replacement probabilities non-parametrically
    % Calculate the empirical replacement probability for each state (age)
    P_hat = estimateReplacementProbabilities(data, maxAge);

    % (b.i.) Construct conditional state transition matrices
    [F0, F1] = constructTransitionMatrices(maxAge);

    % Setup optimization for parameter search
    options = optimset('Display', 'iter', 'TolFun', 1e-8, 'TolX', 1e-8, 'MaxIter', 1000);

    % Initial parameter values - starting at the true values
    theta0 = [-4; -0.5];  % Initial guess for [R; mu]

    % Define objective function for optimization using a simplified approach
    objFun = @(theta) simplifiedCCPLogLikelihood(theta, data, beta, maxAge, P_hat, F0, F1);

    % Run the optimization
    [theta_hat, fval, ~, ~, ~, hessian] = fminunc(objFun, theta0, options);

    % Extract results
    estimates = theta_hat;
    loglik = -fval;  % Convert back to positive log likelihood

    % Compute standard errors
    std_errors = sqrt(diag(inv(hessian)));
```

```matlab

    % Display results
    fprintf('CCP Estimation Results:\n');
    fprintf('R: %.4f (SE: %.4f)\n', estimates(1), std_errors(1));
    fprintf('mu: %.4f (SE: %.4f)\n', estimates(2), std_errors(2));
    fprintf('Log-likelihood: %.4f\n', loglik);
end

% (a) Function to estimate replacement probabilities
function P_hat = estimateReplacementProbabilities(data, maxAge)
    % Initialize array for replacement probabilities
    P_hat = zeros(maxAge, 1);

    % Calculate empirical probability of replacement at each age
    for a = 1:maxAge
        % Get all observations with current age
        idx = data.age == a;

        if sum(idx) > 0
            % Calculate empirical replacement probability
            P_hat(a) = mean(data.replace(idx));

            % Add a small epsilon to avoid 0 or 1 probabilities
            % which would cause problems in the logit inversion
            epsilon = 1e-6;
            P_hat(a) = max(min(P_hat(a), 1-epsilon), epsilon);
        else
            % No observations for this age, use a default or
               interpolate
            warning('No observations found for age %d. Using default
               value.', a);
            P_hat(a) = 0.2;  % Default value
        end
    end

    fprintf('Estimated replacement probabilities:\n');
    for a = 1:maxAge
        fprintf('Age %d: %.4f\n', a, P_hat(a));
    end
end

% (b) Function to construct conditional transition matrices
function [F0, F1] = constructTransitionMatrices(maxAge)
    % F0: Transition matrix if the machine is kept (action = 0)
    F0 = zeros(maxAge, maxAge);

    % F1: Transition matrix if the machine is replaced (action = 1)
```

```
    F1 = zeros(maxAge, maxAge);

    % Fill transition matrices according to the deterministic state
       transition
    for i = 1:maxAge
        % If kept, age increases by 1 up to maxAge
        if i < maxAge
            F0(i, i+1) = 1;
        else
            F0(i, i) = 1;  % At max age, stays at max age
        end

        % If replaced, age becomes 1 regardless of current age
        F1(i, 1) = 1;
    end
end

% (c) log-likelihood function
function [nll, grad] = simplifiedCCPLogLikelihood(theta, data, beta,
   maxAge, P_hat, F0, F1)
    % Extract parameters
    R = theta(1);   % Replacement cost
    mu = theta(2);  % Maintenance cost parameter

    % Compute flow utilities
    flow_u0 = zeros(maxAge, 1);  % Utility from keeping
    flow_u1 = zeros(maxAge, 1);  % Utility from replacing

    for a = 1:maxAge
        flow_u0(a) = mu * a;     % Maintenance cost increases with age
        flow_u1(a) = R;          % Replacement cost is constant
    end

    % Using finite dependence property for the bus engine replacement
        problem
    % The property allows us to simplify the computation of value
       function differences

    % Initialize value function differences
    v_diff = zeros(maxAge, 1);

    for a = 1:maxAge
        % Immediate utility difference
        v_diff(a) = flow_u1(a) - flow_u0(a);

        % Future value difference using finite dependence
        % After replacement, the state is always 1 (new machine)
```

```
        % The future value term is zero because of the finite
           dependence property
        % The paths from either action converge to the same state
    end

    % Calculate log likelihood
    loglik = 0;
    for i = 1:height(data)
        age = data.age(i);
        action = data.replace(i);

        % Calculate probability of replacement using logit
        prob_replace = 1 / (1 + exp(-v_diff(age)));

        % Ensure numerical stability
        prob_replace = max(min(prob_replace, 1-1e-10), 1e-10);

        % Add to log likelihood
        if action
            loglik = loglik + log(prob_replace);
        else
            loglik = loglik + log(1 - prob_replace);
        end
    end

    % Return negative log likelihood for minimization
    nll = -loglik;
    grad = [];
end
```