Contents


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main Simulation Script for Resource Allocation Project with R Integration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
```

Step 1: Import CSV Data

(reference apolloMain_5 amd apolloMain_6 as example for data manipulation) biasData = readtable('user_choices.csv'); % Replace with the path to your data file disp('User bias data imported successfully.'); taskChoice_Data = readtable('user_choices.csv'); % Replace with the path to your data file disp('User task choice data imported successfully.');

```
robotChoice_Data = readtable('G:\My Drive\myResearch\Research Experimentation\Apollo\apollo\data\WarehouseRobot_Pairing_Data\test_pairing_data.csv');
disp('User robot choice data imported successfully.');

% Extract relevant data (modify based on your CSV structure)
% task_attributes = data{:, {'efficiency', 'speed', 'safety', 'durability', 'skill'}};

% Extract and organize robot states for all three alternatives
robot_states = struct();
attributes = {'energy','pace','safety','reliability','intelligence'};
for i = 1:3
    for attr = attributes
        robot_states.(['robot' num2str(i)]).(attr{1}) = ...
            robotChoice_Data.(['robot' num2str(i) attr{1}]);
    end
end

% Extract choice data and other metadata
choices = robotChoice_Data.choice;
participant_ids = robotChoice_Data.participantid;
trial_numbers = robotChoice_Data.trial;
stake_types = robotChoice_Data.staketype;
time_spent = robotChoice_Data.timespent;
```

```
User robot choice data imported successfully.
```

Step 2: R Bridge Implementation

```
disp('Initializing R bridge...');

% Configure paths
rscript_path = 'C:\Program Files\R\R-4.4.2\bin\x64\Rscript.exe';
r_script = 'G:\My Drive\myResearch\Research Experimentation\Apollo\apollo\example\DFT_Resource_Allocation.R';
csvFile = 'G:\My Drive\myResearch\Research Experimentation\Apollo\apollo\data\WarehouseRobot_Pairing_Data\test_pairing_data.csv';
outputDir = 'G:\My Drive\myResearch\Research Experimentation\Apollo\apollo\ResourceAllocation_Output';

% Verify installations
if ~isfile(rscript_path)
    error('Rscript.exe not found at: %s', rscript_path);
elseif ~isfile(r_script)
    error('R script not found at: %s', r_script);
elseif ~isfile(csvFile)
    error('Input CSV not found at: %s', csvFile);
elseif ~isfolder(outputDir)
    warning('Output folder does not exist, creating: %s', outputDir);
    mkdir(outputDir);
end

% Execute R with JSON output
try
    % Use proper argument formatting
    cmd = sprintf(['"%s" "%s" ', ...
                  '-i "%s" -o "%s"'], ...
                  rscript_path, r_script, csvFile, outputDir);
```

```
[status,result] = system(cmd);

    if status == 0
        % Handle output path (whether directory or file)
        if isfolder(outputDir)
            jsonFile = fullfile(outputDir, 'DFT_output.json');
        else
            jsonFile = outputDir;
        end

        % Parse JSON output
        if exist(jsonFile, 'file')
            jsonText = fileread(jsonFile);
            params = jsondecode(jsonText);

            % Extract parameters with validation
            phi1 = max(0, validateParam(params, 'phi1', 0.5)); % Ensure non-negative
            phi2 = min(max(0, validateParam(params, 'phi2', 0.8)), 1); % Constrain 0-1
            tau = 1 + exp(validateParam(params, 'timesteps', 0.5));
            error_sd = validateParam(params, 'error_sd', 0.1);

            % Extract attribute weights
            beta_weights = [
                params.b_energy;
                params.b_pace;
                params.b_safety;
                params.b_reliability;
                params.b_intelligence
            ];

            % Get initial preferences from ASCs
            initial_P = [
                params.asc_1;
                params.asc_2;
                params.asc_3;
                0  % Control alternative1 has 0 initial preference
                0  % Control alternative2 has 0 initial preference
            ];

            disp('Estimated Parameters:');
            disp(['phi1: ', num2str(phi1)]);
            disp(['phi2: ', num2str(phi2)]);
            disp(['tau: ', num2str(tau)]);
            disp(['error_sd: ', num2str(error_sd)]);
            disp('Initial Preferences (from ASCs):');
            disp(initial_P');
        else
            error('R output file not found');
        end
    else
        error('R execution failed: %s', result);
    end
catch ME
    disp('Error during R execution:');
    disp(getReport(ME, 'extended'));
    [phi1, phi2, tau, error_sd] = getFallbackParams();
    beta_weights = [0.3; 0.2; 0.4; 0.1; 0.5]; % Default weights
    initial_P = zeros(5,1); % Neutral initial preferences
end
```

```
Initializing R bridge...
```

Step 3: MDFT Formulation to Calculate Preference Dynamics

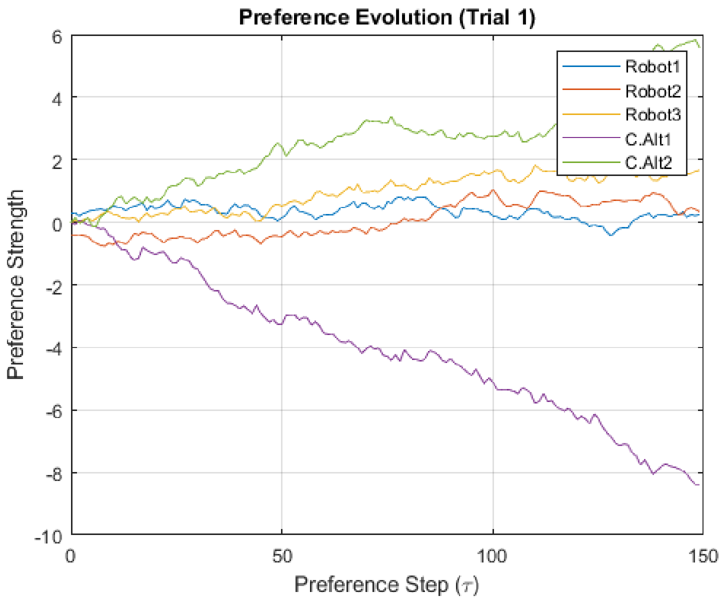
(MDFT calculations based on estimated parameters)

```
current_trial = 1; % Analyze first trial (can be looped later)

% Create M matrix from current trial's attributes
M = [
    robotChoice_Data.robot1energy(current_trial), ...
    robotChoice_Data.robot1pace(current_trial), ...
    robotChoice_Data.robot1safety(current_trial), ...
    robotChoice_Data.robot1reliability(current_trial), ...
    robotChoice_Data.robot1intelligence(current_trial);

    robotChoice_Data.robot2energy(current_trial), ...
```

```matlab
    robotChoice_Data.robot2pace(current_trial), ...
    robotChoice_Data.robot2safety(current_trial), ...
    robotChoice_Data.robot2reliability(current_trial), ...
    robotChoice_Data.robot2intelligence(current_trial);

    robotChoice_Data.robot3energy(current_trial), ...
    robotChoice_Data.robot3pace(current_trial), ...
    robotChoice_Data.robot3safety(current_trial), ...
    robotChoice_Data.robot3reliability(current_trial), ...
    robotChoice_Data.robot3intelligence(current_trial);

    0.1*ones(1,5) % Control alternative1

    0.9*ones(1,5) % Control alternative2
];

% Normalize beta weights
beta = beta_weights ./ sum(abs(beta_weights));

% Calculate DFT dynamics with initial preferences
[E_P, V_P, probs, P_tau] = calculateDFTdynamics(...
    phi1, phi2, tau, error_sd, beta, M, initial_P);

% Display results
disp('=== Current Trial Analysis ===');
disp(['Trial: ', num2str(current_trial)]);
disp(['Participant: ', num2str(participant_ids(current_trial))]);
disp(['Actual Choice: Robot ', num2str(choices(current_trial))]);

disp('M matrix (alternatives × attributes):');
disp(array2table(M, ...
    'RowNames', {'Robot1','Robot2','Robot3','Control Alt1','Control Alt2'}, ...
    'VariableNames', attributes));

% Create comparison table
result_table = table(probs, 'VariableNames', {'Probability'}, ...
                    'RowNames', {'Robot1','Robot2','Robot3','Control1','Control2'});
disp('Choice Probability Distribution:');
disp(result_table);

% Display DFT results with prediction
disp('DFT Results:');
disp(['E_P: ', num2str(E_P', '%.2f  ')]);
disp(['Choice probabilities: ', num2str(probs', '%.3f  ')]);
[~, predicted_choice] = max(probs); % Get index of highest probability
disp(['Predicted choice: Robot ', num2str(predicted_choice)]);
disp(['Actual choice: Robot ', num2str(choices(current_trial))]);
disp(' ');

% Display match/mismatch
if predicted_choice == choices(current_trial)
    disp('✓ Prediction matches actual choice');
else
    disp('✗ Prediction differs from actual choice');
end


% Plot preference evolution
figure;
plot(0:tau, P_tau);
xlabel('Preference Step (\tau)');
ylabel('Preference Strength');
legend({'Robot1','Robot2','Robot3','C.Alt1', 'C.Alt2'});
title(sprintf('Preference Evolution (Trial %d)', current_trial));
grid on;
%{
%% Step 4: Solve Equilibrium Function
% Ensure DFT outputs match expected dimensions
assert(length(E_P) == 5, 'DFT must return 5 alternatives');
assert(isequal(size(V_P), [5 5]), 'DFT covariance must be 5x5');

% Use DFT outputs for equilibrium calculation
Ep_mins = E_P;       % 4×1 expected preferences from DFT
Varp_mins = V_P;     % 4×4 preference covariance from DFT
% using robot attribute to map to product the robot can build type A vs
% type B. this is made using each robot attribute.
% 5 robot produce 2 things. each trial outputs x_mins as 10x1
x_mins = robot_production_capacity(M); % 10×1 robot state vector could recursive for each neighboring robot
```

```
% Call equilibrium solver
solutions = solve_equilibrium(Ep_mins, Varp_mins, x_mins);

% Extract solutions (works with both old and new versions)
if isfield(solutions, 'x')
    % Old version field names
    P_final = solutions.x;
    E_P_eq = solutions.lambda;
    V_P_eq = solutions.mu_vec;
else
    % New version field names
    P_final = solutions.P_final;
    E_P_eq = solutions.E_P_eq;
    V_P_eq = solutions.V_P_eq;
end

% Display results
disp('=== Equilibrium Results ===');
disp(['Final Preferences (P_final): ', num2str(P_final')]);
disp(['Expected Preferences (E_P_eq): ', num2str(E_P_eq')]);
disp(['Preference Variance (V_P_eq diagonal): ', num2str(V_P_eq')]);

%% Step 5: Output Results
disp('Saving results to CSV...');
output_table = table(E_P, V_P, P_final, ...
                     'VariableNames', {'ExpectedPreference', 'VariancePreference', 'FinalPreferences'});
writetable(output_table, 'results.csv');
disp('Results saved successfully!');
%}
```

```
=== Current Trial Analysis ===
Trial: 1
Participant: 1001
Actual Choice: Robot 2
M matrix (alternatives × attributes):
                    energy    pace    safety    reliability    intelligence
                    ______    ____    ______    ___________    ____________

    Robot1           0.65     0.55      0.7          0.6            0.58
    Robot2           0.75     0.45      0.8         0.65            0.62
    Robot3            0.5      0.6     0.65         0.55             0.5
    Control Alt1      0.1      0.1      0.1          0.1             0.1
    Control Alt2      0.9      0.9      0.9          0.9             0.9

Choice Probability Distribution:
                Probability
                ___________

    Robot1        0.11063
    Robot2        0.04412
    Robot3        0.07568
    Control1            0
    Control2      0.76957

DFT Results:
E_P: 0.92   0.24   0.66  -4.65   2.72
Choice probabilities: 0.111  0.044  0.076  0.000  0.770
Predicted choice: Robot 5
Actual choice: Robot 2

✗ Prediction differs from actual choice
```



Helper Functions

```
function param = validateParam(params, name, default)
    if isfield(params, name) && isnumeric(params.(name))
        param = params.(name);
    else
        warning('Using default for %s', name);
        param = default;
    end
end

function [phi1, phi2, tau, error_sd] = getFallbackParams()
    phi1 = 0.5;
    phi2 = 0.8;
    tau = 10;
    error_sd = 0.1;
    warning('Using default parameters');
end
```

```
Estimated Parameters:
phi1: 0.2602
phi2: 0
tau: 149.3093
error_sd: 0.0973
Initial Preferences (from ASCs):
    0.2804   -0.3938         0         0         0
```

Published with MATLAB® R2022b