```
%Change the following to workspace of your cureent folder
cd('C:\Users\hanib\Desktop\validationSetup\Validation Version 1');

% Load experiment and coefficient tables
load('experiment.mat');
load('allCoeff.mat');

% Surface areas
S_aria = 7.932558e-02;
S_settle = 1.011951e-02;
S_counter = 1.504496e-02;
t_step = 0.08;
T_air = 70;

% Physical constants
C_mold = 1 / (2278.4 * 7300.0 * 2.196412e-03); % Mold heat capacity
C_gob = 1 / (1460.0 * 2350.0 * 9.288814e-05); % Gob heat capacity

% Preallocate variables for storing data in a table
data = [];  % This will hold all the data for CSV output

% Open a text file to write the simulation results
fileID = fopen('simulation_results.txt', 'w');

% Iterate over each unique realIndex in experiments
unique_real_index = unique(experiment.realIndex);  % Get unique real indices (1 to
75)

for case_num = 1:length(unique_real_index)
    % Get the coefficient index for the current case using realIndex
    case_index = unique_real_index(case_num);

    % Extract the coefficients for this case from the allCoeff table
    coeff_row = allCoeff(case_index, :);
    a1_temp = [coeff_row.a1_1, coeff_row.a1_2, coeff_row.a1_3, coeff_row.a1_4];
    a2_temp = [coeff_row.a2_1, coeff_row.a2_2, coeff_row.a2_3, coeff_row.a2_4];
    b1_temp = [coeff_row.b1_1, coeff_row.b1_2, coeff_row.b1_3, coeff_row.b1_4];
    b2_temp = [coeff_row.b2_1, coeff_row.b2_2, coeff_row.b2_3, coeff_row.b2_4];
    ks_temp = [coeff_row.ks_1, coeff_row.ks_2, coeff_row.ks_3, coeff_row.ks_4];
    kc_temp = [coeff_row.kc_1, coeff_row.kc_2, coeff_row.kc_3, coeff_row.kc_4];

    a1 = fliplr(a1_temp);
    a2 = fliplr(a2_temp);
    ks = fliplr(ks_temp);
    b1 = fliplr(b1_temp);
    b2 = fliplr(b2_temp);
    kc = fliplr(kc_temp);

    % Extract the experiment parameters for this case (18 cycles per case)
    exp_data = experiment(experiment.realIndex == case_index, :);
```

```
    % Write case number in the text file
    fprintf(fileID, 'Case %d Results:\n', case_num);

    % Loop through 18 cycles for each case
    for cycle_num = 1:18
        % Extract the timing data for this cycle
        t_settle_start = exp_data.timingssettlestart(cycle_num);
        t_settle = exp_data.timingssettleend(cycle_num) - t_settle_start;
        t_counter_start = exp_data.timingscounterstart(cycle_num);
        t_counter = exp_data.timingscounterend(cycle_num) - t_counter_start;
        t_cooling_start = exp_data.timingscoolingstart(cycle_num);
        t_cooling = exp_data.timingscoolingend(cycle_num) - t_cooling_start;
        t_cycle = exp_data.cycle_duration(cycle_num);

        % Set initial gob and mold temperatures
        T_g_init = exp_data.gobstart(cycle_num);
        T_m_init = exp_data.moldstart(cycle_num);

        % Get the real final values from the experiment data
        T_g_real = exp_data.gobend(cycle_num);
        T_m_real = exp_data.moldend(cycle_num);

        % Run the simulation for this cycle
        [T_g_final, T_m_final] = simulate_cycle(T_g_init, T_m_init, ...
            t_settle_start, t_settle, t_counter_start, t_counter, ...
            t_cooling_start, t_cooling, t_cycle, t_step, a1, a2, b1, b2, ks, kc, ...
            S_settle, S_counter, S_aria, C_mold, C_gob, T_air);

        % Calculate errors between simulated and real values
        gob_error = T_g_final - T_g_real;
        mold_error = T_m_final - T_m_real;

        % Append data for this cycle to the data array (for CSV)
        data = [data; case_num, cycle_num, T_m_init, T_m_final, T_m_real,
mold_error, ...
                T_g_init, T_g_final, T_g_real, gob_error];

        % Write the data for this cycle to the text file
        fprintf(fileID, 'Cycle %d\n', cycle_num);
        fprintf(fileID, 'temp_mold_init = %.2f\n', T_m_init);
        fprintf(fileID, 'temp_mold_final (simulated) = %.2f\n', T_m_final);
        fprintf(fileID, 'temp_mold_final (real) = %.2f\n', T_m_real);
        fprintf(fileID, 'Error (mold) = %.2f\n', mold_error);
        fprintf(fileID, 'temp_gob_init = %.2f\n', T_g_init);
        fprintf(fileID, 'temp_gob_final (simulated) = %.2f\n', T_g_final);
        fprintf(fileID, 'temp_gob_final (real) = %.2f\n', T_g_real);
        fprintf(fileID, 'Error (gob) = %.2f\n', gob_error);
        fprintf(fileID, '========================\n');
    end
```

```
end

% Close the text file
fclose(fileID);

% Step 2: Create a table with headers and save as CSV
headers = {'Case', 'Cycle', 'temp_mold_init', 'temp_mold_final_sim',
'temp_mold_final_real', ...
           'Error_mold', 'temp_gob_init', 'temp_gob_final_sim',
'temp_gob_final_real', 'Error_gob'};

% Convert the data array to a table
results_table = array2table(data, 'VariableNames', headers);

% Write the table to a CSV file
writetable(results_table, 'simulation_results.csv');
```

```
function [T_g_final, T_m_final] = simulate_cycle(T_gob_initial, T_mold_initial, ...
    t_settle_start, t_settle_duration, t_counter_start, t_counter_duration, ...
    t_cooling_start, t_cooling_duration, cycle_end, t_step, a1, a2, b1, b2, ks, kc,
...
    S_settle, S_counter, S_aria, C_mold, C_gob, T_air)

    % Initialize temperatures and times
    T_gob = T_gob_initial;
    T_mold = T_mold_initial;
    time = 0.0;

    % Loop over the cycle time
    while time < cycle_end
        hf_gob = 0.0;
        hf_cool = 0.0;

        if time >= t_settle_start && time < t_settle_start + t_settle_duration
            [T_gob, T_mold, hf_gob] = simulate_settle(a1, a2, ks, T_gob, T_mold,
t_step, S_settle, C_mold, C_gob);
        end

        if time >= t_counter_start && time < t_counter_start + t_counter_duration
            [T_gob, T_mold, hf_gob] = simulate_counter(b1, b2, kc, T_gob, T_mold,
t_step, S_counter, C_mold, C_gob);
        end

        if time >= t_cooling_start && time < t_cooling_start + t_cooling_duration
            [T_mold, hf_cool] = simulate_cooling(T_mold, t_step, S_aria, T_air,
C_mold);
        end
```

```
        if time >= t_counter_start + t_counter_duration && time < cycle_end
            T_mold = simulate_passive_cooling(T_mold, t_step, T_air, C_mold);
        end

        % Increment time
        time = time + t_step;
    end

    % Return the final values
    T_g_final = T_gob;
    T_m_final = T_mold;
end
```

Simulate_settle

```
function [T_g_s, T_m_s, hf_settle] = simulate_settle(a1, a2, ks, T_g, T_m, t_step,
S_settle, C_mold, C_gob)
    T_g_surface = polyval(a2, T_g - T_m); % Evaluate polynomial for T_g_surface
    T_m_surface = polyval(a1, T_g - T_m); % Evaluate polynomial for T_m_surface

    hf_settle = S_settle * polyval(ks, T_g_surface - T_m_surface); % Heat flow
during settle

    T_m_s = T_m - (hf_settle * C_mold * t_step); % Update mold temperature
    T_g_s = T_g + (hf_settle * C_gob  * t_step); % Update gob temperature
end
```

Simulate Counter

```
% Simulate counter phase
function [T_g_c, T_m_c, hf_counter] = simulate_counter(b1, b2, kc, T_g, T_m,
t_step, S_counter, C_mold, C_gob)
    T_g_surface = polyval(b2, T_g - T_m); % Evaluate polynomial for T_g_surface
    T_m_surface = polyval(b1, T_g - T_m); % Evaluate polynomial for T_m_surface

    hf_counter = S_counter * polyval(kc, T_g_surface - T_m_surface); % Heat flow
during counter

    T_m_c = T_m - (hf_counter * C_mold * t_step); % Update mold temperature
    T_g_c = T_g + (hf_counter * C_gob  * t_step); % Update gob temperature
end
```

Simulate Cooling Active

```
% Simulate cooling phase
function [T_m_c, hf_cooling] = simulate_cooling(T_m, t_step, S_aria, T_air, C_mold)
    hf_cooling = S_aria * 360 * (T_air - T_m); % Cooling heat flow
    T_m_c = T_m + (hf_cooling * C_mold * t_step); % Update mold temperature
end
```

Simulate Cooling passive

```
% Simulate passive cooling phase
function [T_m_c] = simulate_passive_cooling(T_m, t_step, T_air, C_mold)
    hf_passive = (T_m - T_air) * 1.053 - 40.99; % Passive cooling heat flow
    T_m_c = T_m - (hf_passive * C_mold * t_step); % Update mold temperature
end
```