

UWTSD

Alternative Fuels Combustion Modelling

Task 3

ACHA 7002

2204298

07/12/2025

Contents

1	Task 3a	1
1.1	Table of own functions results compared with given function.....	1
2	Task 3b	1
2.1	Table of research of MATLAB/Online functions	1
2.1.1	Additional researched functions findings	2
2.2	Table of tried MATLAB/Online functions results	2
3	Referencing.....	Error! Bookmark not defined.
4	Appendices.....	3
4.1	Own code and functions task 3a	4
4.2	Own code task 3b	41

1 Task 3a

1.1 Table of own functions results compared with given function

Method	n5_Value	Time_s	Iterations	Result_Difference	Lines_of_Code
{'Original' }	1.5639e-05	0.0075016	1.6404e+06	0	147
{'Newton-Raphson'}	1.5639e-05	0.0003514	15	1.0267e-12	121
{'Bisection' }	1.5736e-05	0.000431	21	9.6348e-08	125
{'Secant' }	1.5681e-05	0.0002864	7	4.1776e-08	111
{'Fixed-Point' }	1.537e-05	0.0002352	10	2.6972e-07	66

2 Task 3b

2.1 Table of research of MATLAB/Online functions

Table 1 - method ranking for linear equation balance problems

Rank	Method	Type	Ease of Use (x/5)	Efficiency (x/5)	Accuracy (x/5)	Situation to use
1	A\b	Linear algebra solver	5	5	5	Solve full linear element- balance systems. Fastest and most suitable for linier systems (Mathworks, n.d.)
2	Linsolve	Linear algebra solver	3	3	5	Solve linear equations in the form $A(x) = b$. It works best when the coefficient matrix has a simple structure. (Mathworks, n.d.)
3	Fsolve	Non-linear root solver	3	4	4	A non-linear root solver. Good for

						nonlinear equilibrium calculations. (Mathworks, n.d.)
--	--	--	--	--	--	--

-0p;//

2.1.1 Additional researched functions findings

The `a\b` operator automatically chooses the most efficient function for any linear equation in the form $A(x) = b$ based on the properties of your input matrix. This makes it fast, reliable and efficient for non-linear problems.

`Linsolve` also solves linear equations in the same format, but it allows the user to specify matrix characteristics which allows provides MATLAB with more information that helps it to choose the most appropriate function.

`Fsolve` is a non-linear equation solver that uses iterative numerical techniques that can handle complex models including exponentials and equilibrium equations. `Fsolve` requires an initial guesses or starting points that have a direct impact on its convergence behaviour and time.

2.2 Table of tried MATLAB/Online functions results

3 References

Mathworks, n.d. *Matlab help center*. [Online]

Available at:

<https://uk.mathworks.com/support/search.html?c%5B%5D=support&q=non%20linear%20solver&page=1>

[Accessed 07 12 2025].

Mathworks, n.d. *mathworks*. [Online]

Available at:

https://uk.mathworks.com/help/matlab/ref/double.mldivide.html?s_tid=srchtitle_support_results_1_mldivide

[Accessed 07 12 2025].

Mathworks, n.d. *Matlab help center*. [Online]

Available at:

https://uk.mathworks.com/help/matlab/ref/linsolve.html?s_tid=srchtitle_support_results_2_linear+root+solver

[Accessed 07 12 2025].

4 Appendices

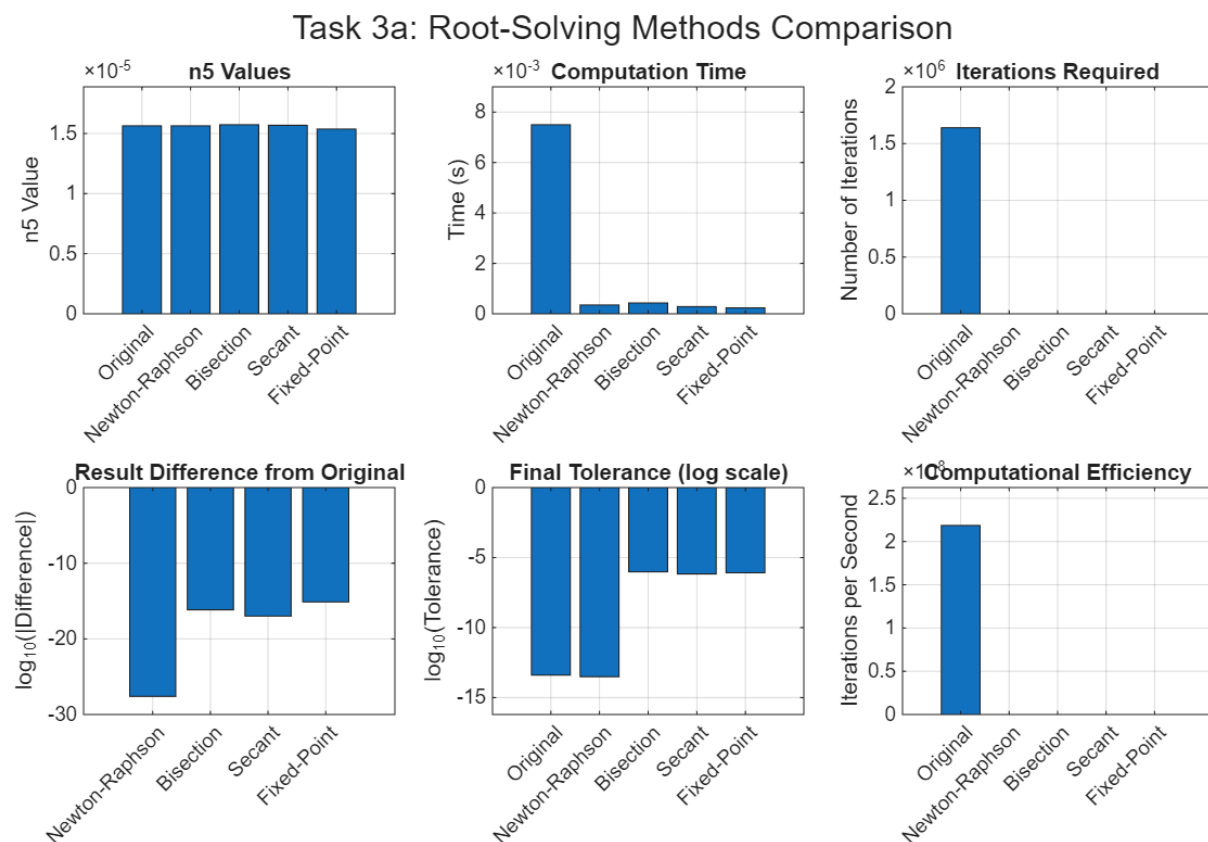


Figure 1 - root solving methods comparison

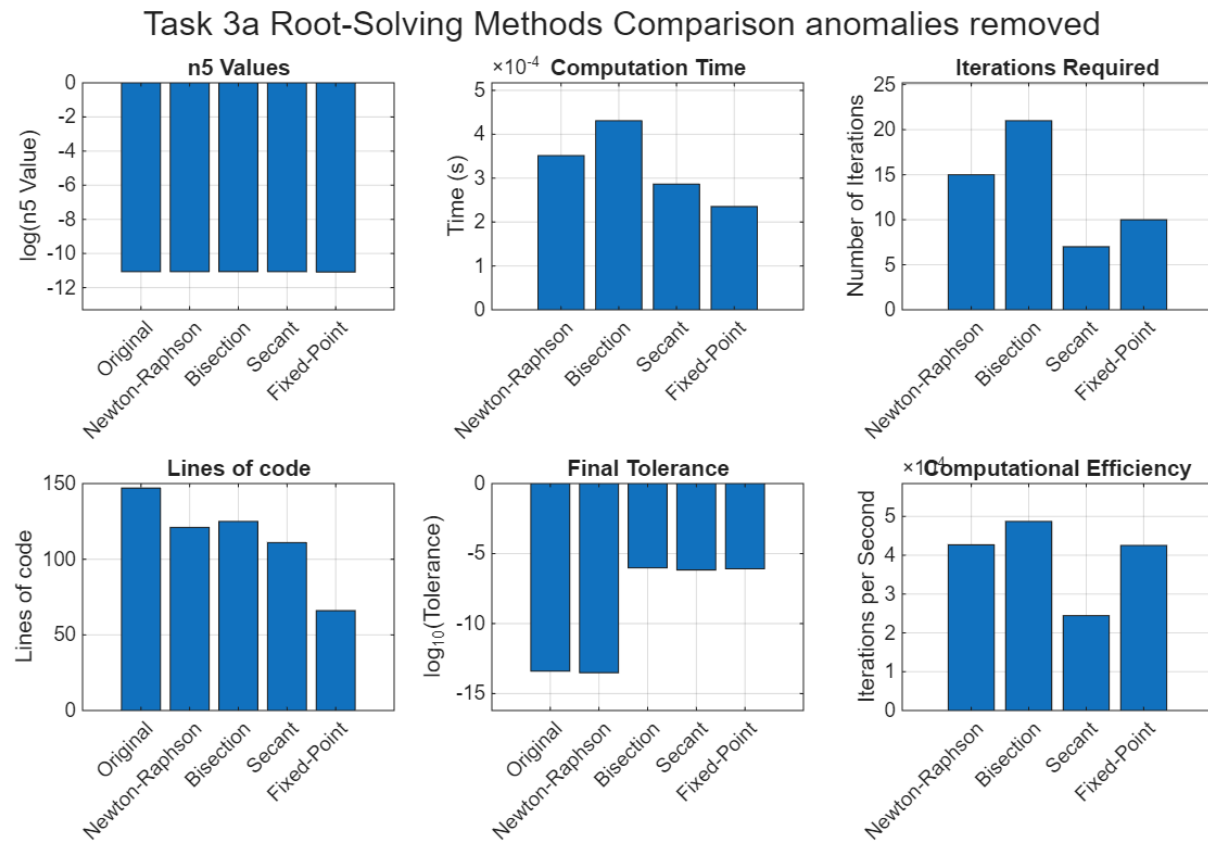


Figure 2 - Root solving methods comparisons with anomalies removed

4.1 Own code and functions task 3a

%% Task 3a - Comparison of Root-Solving Methods

clc; clear; close all;

%% Test Parameters

c = 8;

h = 18;

o = 0;

airs = 12.5;

temp_test = 1500;

```
press_test = 50;

feq_test = 1.0;


fprintf('\n----- Root-Solving Methods Comparison ----- \n\n');


fprintf('Test conditions:\n');

fprintf(' Fuel composition: C%dH%d\n', c, h);

fprintf(' Temperature: %.0f K\n', temp_test);

fprintf(' Pressure: %.0f bar\n', press_test);

fprintf(' Equivalence ratio: %.2f\n', feq_test);

fprintf(' Stoichiometric air: %.2f\n', airs);


%% Benchmark Original Method


fprintf('\n----- Benchmark Original Method (mydissfuniii) ----- \n\n');


tic;

[n5nt_original, error_original, iter_original] = iterations_3(c, h, o, temp_test, feq_test,
airs, press_test);

time_original = toc;

fprintf('Result: n5/nt = %.8e\n', n5nt_original);

fprintf('Iterations: %d\n', iter_original);

fprintf('Final error: %.2e\n', error_original);

fprintf('Computation time: %.6f s\n', time_original);
```

%% Method 1 Newton-Raphson

fprintf('\n----- Method 1 Newton-Raphson ----- \n\n');

tic;

[n5_NR, iter_NR, tol_NR] = newton_3(c, h, o, temp_test, feq_test, airs, press_test);

time_NR = toc;

fprintf('Result: n5 = %.8e\n', n5_NR);

fprintf('Iterations: %d\n', iter_NR);

fprintf('Computation time: %.6f s\n', time_NR);

fprintf('Final tolerance: %.2e\n\n', tol_NR);

%% Method 2 Bisection

fprintf('\n----- Method 2 Bisection ----- \n\n');

tic;

[n5_bisect, iter_bisect, tol_bisect] = bisect_3(c, h, o, temp_test, feq_test, airs, press_test);

time_bisect = toc;

fprintf('Result: n5 = %.8e\n', n5_bisect);

fprintf('Iterations: %d\n', iter_bisect);

fprintf('Computation time: %.6f s\n', time_bisect);

fprintf('Final tolerance: %.2e\n\n', tol_bisect);

%% Method 3 Secant Method

```

fprintf('\n----- Method 3 Secant ----- \n\n');

tic;

[n5_secant, iter_secant, tol_secant] = secant_3(c, h, o, temp_test, freq_test, airs,
press_test);

time_secant = toc;

fprintf('Result: n5 = %.8e\n', n5_secant);

fprintf('Iterations: %d\n', iter_secant);

fprintf('Computation time: %.6f s\n', time_secant);

fprintf('Final tolerance: %.2e\n\n', tol_secant);

%% Method 4 Fixed-Point Iteration

fprintf('\n----- Method 4 Fixed-Point ----- \n\n');

tic;

[n5_fixed, iter_fixed, tol_fixed] = fixedpoint_3(c, h, o, temp_test, freq_test, airs,
press_test);

time_fixed = toc;

fprintf('Result: n5 = %.8e\n', n5_fixed);

fprintf('Iterations: %d\n', iter_fixed);

fprintf('Computation time: %.6f s\n', time_fixed);

fprintf('Final tolerance: %.2e\n\n', tol_fixed);

%% Comparison Table

```

```

fprintf('\n----- Summary Table ----- \n\n');

methods = {'Original'; 'Newton-Raphson'; 'Bisection'; 'Secant'; 'Fixed-Point'};
n5_values = [n5nt_original; n5_NR; n5_bisect; n5_secant; n5_fixed];
iterations = [iter_original; iter_NR; iter_bisect; iter_secant; iter_fixed];
comp_time = [time_original; time_NR; time_bisect; time_secant; time_fixed];
tolerance = [abs(error_original); tol_NR; tol_bisect; tol_secant; tol_fixed];

% Calculate result difference from original method (absolute difference)
result_diff = abs(n5_values - n5nt_original);

lines_of_code = [147; 121; 125; 111; 66];    % script and functions not including spaces
or comments

% Create table
comparison_table = table(methods, n5_values, comp_time, iterations, result_diff,
lines_of_code, ...

    'VariableNames', {'Method', 'n5_Value', 'Time_s' 'Iterations', 'Result_Difference',
'Lines_of_Code'});

disp(comparison_table);

%% Plotting
figure();

```

```
% n5 values

subplot(2,3,1);

bar(n5_values);

ylim([0 max(n5_values)*1.2]);

set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);

ylabel('n5 Value');

title('n5 Values');

grid on;
```

```
% Computation time

subplot(2,3,2);

bar(comp_time);

set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);

ylabel('Time (s)');

ylim([0 max(comp_time)*1.2])

title('Computation Time');

grid on;
```

```
% Iterations

subplot(2,3,3);

bar(iterations);

set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);

ylabel('Number of Iterations');
```

```
title('Iterations Required');
```

```
grid on;
```

```
% Result difference from original w/ log scale
```

```
subplot(2,3,4);
```

```
bar(log(result_diff([2 3 4 5])+1e-15)); % remove original to avoid log(0)
```

```
set(gca, 'XTickLabel', methods([2 3 4 5]), 'XTickLabelRotation', 45);
```

```
ylabel('log10(| Difference |)');
```

```
title('Result Difference from Original');
```

```
grid on;
```

```
% Tolerance achieved w/ log scale
```

```
subplot(2,3,5);
```

```
bar(log10(tolerance));
```

```
ylim([(min(log10(tolerance))*1.2) 0]);
```

```
set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);
```

```
ylabel('log10(Tolerance)');
```

```
title('Final Tolerance (log scale)');
```

```
grid on;
```

```
% Efficiency
```

```
subplot(2,3,6);
```

```
efficiency = iterations ./ comp_time;

bar(efficiency);

set(gca, 'XTickLabel', methods(1:end), 'XTickLabelRotation', 45);

ylabel('Iterations per Second');

ylim([0 max(efficiency)*1.2])

title('Computational Efficiency');

grid on;

sgtitle('Task 3a: Root-Solving Methods Comparison');
```

```
%% Plotting 2
```

```
figure();

% n5 values w/log

subplot(2,3,1);

bar(log(n5_values));

set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);

ylabel('log(n5 Value)');

ylim([min(log(n5_values))*1.2 0]);

title('n5 Values');

grid on;
```

```
% Computation time

subplot(2,3,2);

bar(comp_time([2 3 4 5]));
```

```
set(gca, 'XTickLabel', methods([2 3 4 5]), 'XTickLabelRotation', 45);  
ylabel('Time (s)');  
ylim([0 max(comp_time([2 3 4 5]))*1.2])  
title('Computation Time');  
grid on;
```

```
% Iterations
```

```
subplot(2,3,3);  
bar(iterations([2 3 4 5]));  
set(gca, 'XTickLabel', methods([2 3 4 5]), 'XTickLabelRotation', 45);  
ylabel('Number of Iterations');  
ylim([0 max(iterations([2 3 4 5]))*1.2]);  
title('Iterations Required');  
grid on;
```

```
% lines of code
```

```
subplot(2,3,4);  
bar(lines_of_code); % remove original to avoid log(0)  
set(gca, 'XTickLabel', methods([1 2 3 4 5]), 'XTickLabelRotation', 45);  
ylabel('Lines of code');  
title('Lines of code');  
grid on;
```

```

% Tolerance achieved

subplot(2,3,5);

bar(log10(tolerance));

ylim([(min(log10(tolerance))*1.2) 0]);

set(gca, 'XTickLabel', methods, 'XTickLabelRotation', 45);

ylabel('log_{10}(Tolerance)');

title('Final Tolerance');

grid on;


% Efficiency

subplot(2,3,6);

efficiency = iterations ./ comp_time;

bar(efficiency([2 3 4 5]));

set(gca, 'XTickLabel', methods([2 3 4 5]), 'XTickLabelRotation', 45);

ylabel('Iterations per Second');

ylim([0 max(efficiency([2 3 4 5]))*1.2])

title('Computational Efficiency');

grid on;


sgtitle('Task 3a Root-Solving Methods Comparison anomalies removed');


function [out, error, total_iterations] = iterations_3(c, h, o, temp_test, feq_test, airs,
press_test)

```

```
n5ntvalue = 0.1; % Initial guess for n5/nt ratio
```

```
error = 1;
```

```
total_iterations = 0;
```

```
%TEMP CO2    H2O      CO
```

```
gibbseng = [
```

```

0      -393.151 -238.921 -113.805
100    -393.683 -236.584 -120.239
200    -394.085 -232.766 -128.526
298.15 -394.389 -228.582 -137.163
300    -394.394 -228.5   -137.328
400    -394.675 -223.901 -146.338
500    -394.939 -219.051 -155.414
600    -395.182 -214.007 -164.486
700    -395.398 -208.812 -173.518
800    -395.586 -203.496 -182.497
900    -395.748 -198.083 -191.416
1000   -395.886 -192.59  -200.275
1100   -396.001 -187.033 -209.075
1200   -396.098 -181.425 -217.819
1300   -396.177 -175.774 -226.509
1400   -396.24  -170.089 -235.149
1500   -396.288 -164.376 -243.74
1600   -396.323 -158.639 -252.284
1700   -396.344 -152.883 -260.784
```

```
1800    -396.353 -147.111 -269.242
```

```
1900    -396.349 -141.325 -277.658
```

```
2000    -396.333 -135.528 -286.034
```

```
];
```

```
% Gibbs free energy
```

```
gco2 = interp1(gibbseng(:,1), gibbseng(:,2), temp_test) * 1e3;
```

```
gh2o = interp1(gibbseng(:,1), gibbseng(:,3), temp_test) * 1e3;
```

```
gco = interp1(gibbseng(:,1), gibbseng(:,4), temp_test) * 1e3;
```

```
% Equilibrium constants
```

```
kco2 = exp(-(gco2-gco) * 1e3 / (8314.46 * temp_test));
```

```
kh2o = exp(-(gh2o) * 1e3 / (8314.46 * temp_test));
```

```
%Goodger method for calculation of dissociation
```

```
%Iterate through n5/nt guess until error goes negative or guess reaches 0
```

```
while error > 0
```

```
    a = sqrt(n5ntvalue * press_test);
```

```
    n3 = (c * feq_test) / (kco2 * a + 1);
```

```
    n4 = ((h/2) * feq_test) / (kh2o * a + 1);
```

```
    n1 = c * feq_test - n3;
```

```
    n2 = (h/2) * feq_test - n4;
```

```
    n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;
```

```
    n6 = airs * 3.76;
```

```
    ntsum = n1 + n2 + n3 + n4 + n5 + n6;
```

```

error = n5ntvalue - n5/ntsum;

n5ntvalue = n5ntvalue - 0.001;

total_iterations = total_iterations + 1;

if n5ntvalue < 0, break; end

end

%-----

%If guess reached 0, reset guess to 0.001 and iterate down in 10^-3 steps

if n5ntvalue < 0

    n5ntvalue = 0.001;

    while error > 0

        a = sqrt(n5ntvalue * press_test);

        n3 = (c * feq_test) / (kco2 * a + 1);

        n4 = ((h/2) * feq_test) / (kh2o * a + 1);

        n1 = c * feq_test - n3;

        n2 = (h/2) * feq_test - n4;

        n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

        n6 = airs * 3.76;

        ntsum = n1 + n2 + n3 + n4 + n5 + n6;

        error = n5ntvalue - n5/ntsum;

    end

    n5ntvalue = n5ntvalue - 0.000001;

    total_iterations = total_iterations + 1;

    if n5ntvalue < 0, break; end

```

```

    end

    %if error went negative, iterate guess in +10^-6 increments until error is

    %positive

    else

        while error < 0

            a = sqrt(n5ntvalue * pressure);

            n3 = (c * feq_test) / (kco2 * a + 1);

            n4 = ((h/2) * feq_test) / (kh2o * a + 1);

            n1 = c * feq_test - n3;

            n2 = (h/2) * feq_test - n4;

            n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

            n6 = airs * 3.76;

            ntsum = n1 + n2 + n3 + n4 + n5 + n6;

            error = n5ntvalue - n5/ntsum;

            n5ntvalue = n5ntvalue + 0.000001;

            total_iterations = total_iterations + 1;

        end

    end

    %-----

    %repeat with smaller increments

    %now positive error

    if n5ntvalue < 0

        n5ntvalue = 0.000001;

        while error > 0

```

```

a = sqrt(n5ntvalue * presstest);

n3 = (c * feq_test) / (kco2 * a + 1);

n4 = ((h/2) * feq_test) / (kh2o * a + 1);

n1 = c * feq_test - n3;

n2 = (h/2) * feq_test - n4;

n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

n6 = airs * 3.76;

ntsum = n1 + n2 + n3 + n4 + n5 + n6;

error = n5ntvalue - n5/ntsum;

n5ntvalue = n5ntvalue - 0.000000001;

total_iterations = total_iterations + 1;

if n5ntvalue < 0, break; end

end

else

while error > 0

a = sqrt(n5ntvalue * press_test);

n3 = (c * feq_test) / (kco2 * a + 1);

n4 = ((h/2) * feq_test) / (kh2o * a + 1);

n1 = c * feq_test - n3;

n2 = (h/2) * feq_test - n4;

n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

n6 = airs * 3.76;

ntsum = n1 + n2 + n3 + n4 + n5 + n6;

error = n5ntvalue - n5/ntsum;

```

```

        n5ntvalue = n5ntvalue - 0.000000001;

        total_iterations = total_iterations + 1;

    end

end

%-----

%error now negative

if n5ntvalue < 0

    n5ntvalue = 0.000000001;

    while error > 0

        a = sqrt(n5ntvalue * press_test);

        n3 = (c * feq_test) / (kco2 * a + 1);

        n4 = ((h/2) * feq_test) / (kh2o * a + 1);

        n1 = c * feq_test - n3;

        n2 = (h/2) * feq_test - n4;

        n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

        n6 = airs * 3.76;

        ntsum = n1 + n2 + n3 + n4 + n5 + n6;

        error = n5ntvalue - n5/ntsum;

        n5ntvalue = n5ntvalue - 0.000000000001;

        total_iterations = total_iterations + 1;

        if n5ntvalue < 0, break; end

    end

else

```

```

while error < 0

    a = sqrt(n5ntvalue * press_test);

    n3 = (c * feq_test) / (kco2 * a + 1);

    n4 = ((h/2) * feq_test) / (kh2o * a + 1);

    n1 = c * feq_test - n3;

    n2 = (h/2) * feq_test - n4;

    n5 = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

    n6 = airs * 3.76;

    ntsum = n1 + n2 + n3 + n4 + n5 + n6;

    error = n5ntvalue - n5/ntsum;

    n5ntvalue = n5ntvalue + 0.0000000000001;

    total_iterations = total_iterations + 1;

end

end

% Return final value

out = n5ntvalue;

end

function [n5, iterations, final_tol] = newton_3(c, h, o, temp_test, feq_test, airs,
press_test)

    tol = 1e-10;

    max_iter = 100;

```

```
n5 = 1e-10; % initial guess
```

```
for iterations = 1:max_iter
```

```
    [f, df] = mydissfun_derivative(n5, c, h, o, temp_test, feq_test, airs, press_test);
```

```
    % Check for zero derivative
```

```
    if abs(df) < 1e-15
```

```
        warning('Derivative too small at iteration %d', iterations);
```

```
        break;
```

```
    end
```

```
    n5_new = n5 - f/df;
```

```
    % Ensure n5 stays positive
```

```
    if n5_new < 0
```

```
        n5_new = n5 / 2; % Reduce step if going negative
```

```
    end
```

```
    final_tol = abs(n5_new - n5);
```

```
    if final_tol < tol
```

```
        n5 = n5_new;
```

```
        return;
```

```
    end
```

```

    n5 = n5_new;

end

end

function [f, df] = mydissfun_derivative(n5, carbon, hydrogen, oxygen, temperature,
fuelequiv, molar_air_stoich, pressure)

% Inputs:

% n5          - Guessed value of n5/nt ratio (oxygen mole fraction)
% carbon      - Carbon atoms in fuel molecule
% hydrogen    - Hydrogen atoms in fuel molecule
% oxygen      - Oxygen atoms in fuel molecule
% temperature - Combustion temperature [K]
% fuelequiv   - Fuel equivalence ratio
% molar_air_stoich - Stoichiometric air requirement
% pressure    - Combustion pressure [bar]

% Gibbs Free Energy Data

gibbseng = [0   -393.151 -238.921 -113.805

100 -393.683 -236.584 -120.239

200 -394.085 -232.766 -128.526

298.15 -394.389 -228.582 -137.163

300 -394.394 -228.5   -137.328

400 -394.675 -223.901 -146.338

500 -394.939 -219.051 -155.414

```

600 -395.182 -214.007 -164.486
700 -395.398 -208.812 -173.518
800 -395.586 -203.496 -182.497
900 -395.748 -198.083 -191.416
1000 -395.886 -192.59 -200.275
1100 -396.001 -187.033 -209.075
1200 -396.098 -181.425 -217.819
1300 -396.177 -175.774 -226.509
1400 -396.24 -170.089 -235.149
1500 -396.288 -164.376 -243.74
1600 -396.323 -158.639 -252.284
1700 -396.344 -152.883 -260.784
1800 -396.353 -147.111 -269.242
1900 -396.349 -141.325 -277.658
2000 -396.333 -135.528 -286.034
2100 -396.304 -129.721 -294.372
2200 -396.262 -123.905 -302.672
2300 -396.209 -118.082 -310.936
2400 -396.142 -112.252 -319.165
2500 -396.062 -106.416 -327.358
2600 -395.969 -100.575 -335.517
2700 -395.862 -94.729 -343.643
2800 -395.742 -88.878 -351.736
2900 -395.609 -83.023 -359.797
3000 -395.461 -77.163 -367.826

3100	-395.298	-71.298	-375.824
3200	-395.122	-65.43	-383.792
3300	-394.932	-59.558	-391.73
3400	-394.726	-53.681	-399.638
3500	-394.506	-47.801	-407.517
3600	-394.271	-41.916	-415.366
3700	-394.022	-36.027	-423.188
3800	-393.756	-30.133	-430.981
3900	-393.477	-24.236	-438.746
4000	-393.183	-18.334	-446.485
4100	-392.874	-12.427	-454.196
4200	-392.55	-6.516	-461.881
4300	-392.21	-0.6	-469.539
4400	-391.857	5.32	-477.171
4500	-391.488	11.245	-484.778
4600	-391.105	17.175	-492.359
4700	-390.706	23.111	-499.915
4800	-390.292	29.052	-507.446
4900	-389.862	34.998	-514.951
5000	-389.419	40.949	-522.433
5100	-388.959	46.906	-529.89
5200	-388.486	52.869	-537.323
5300	-387.996	58.838	-544.732
5400	-387.493	64.811	-552.118
5500	-386.974	70.791	-559.48

```

5600 -386.439 76.777 -566.818
5700 -385.89 82.769 -574.133
5800 -385.324 88.767 -581.424
5900 -384.745 94.77 -588.694
6000 -384.148 100.78 -595.939];

```

```
% gibbs free energy (KJ/mol)
```

```
gco2 = interp1(gibbseng(:,1), gibbseng(:,2), temperature);
```

```
gh2o = interp1(gibbseng(:,1), gibbseng(:,3), temperature);
```

```
gco = interp1(gibbseng(:,1), gibbseng(:,4), temperature);
```

```
% Universal gas constant
```

```
R = 8.31446e-3; % kJ/(mol ·K)
```

```
kco2 = exp(-(gco2 - gco) / (R * temperature));
```

```
kh2o = exp(-gh2o / (R * temperature));
```

```
% Moles using Current n5 Guess
```

```
a = sqrt(n5 * pressure);
```

```
% Calculate CO and CO2 moles
```

```
n3 = (carbon * fuelequiv) / (kco2 * a + 1); % CO2 moles
```

```
n1 = carbon * fuelequiv - n3; % CO moles
```

```
% Calculate H2 and H2O moles
```

```
n4 = ((hydrogen/2) * fuelequiv) / (kh2o * a + 1); % H2 moles
```

```
n2 = (hydrogen/2) * fuelequiv - n4; % H2O moles
```

```
% Calculate O2 moles from oxygen balance
```

```
n5_calc = (molar_air_stoich * 2 + oxygen * fuelequiv - 2*n1 - n2 - n3) / 2;
```

```
% Calculate N2 moles (inert)
```

```
n6 = molar_air_stoich * 3.76;
```

```
% Total moles
```

```
ntsum = n1 + n2 + n3 + n4 + n5_calc + n6;
```

```
% Residual
```

```
f = n5 - (n5_calc / ntsum);
```

```
%finding the derivative df/dn5 using Chain Rule
```

```
%f = n5 - n5_calc/ntsum;
```

```
%df/dn5 = 1 - d(n5_calc/ntsum)/dn5;
```

```
if n5 > 0
```

```
da_dn5 = 0.5 * sqrt(pressure / n5);
```

```
else
```

```
da_dn5 = 0; % Avoid division by zero
```

```
end
```

```
% dn3/dn5: n3 = (carbon*fuelequiv) / (kco2*a + 1)
```

```
% Using chain rule: dn3/dn5 = -(carbon*fuelequiv) * kco2 * da_dn5 / (kco2*a + 1)^2
```

```
dn3_dn5 = -(carbon * fuelequiv) * kco2 * da_dn5 / ((kco2 * a + 1)^2);
```

```
% dn1/dn5: n1 = carbon*fuelequiv - n3
```

```
dn1_dn5 = -dn3_dn5;
```

```
% dn4/dn5: n4 = ((hydrogen/2)*fuelequiv) / (kh2o*a + 1)
```

```
dn4_dn5 = -((hydrogen/2) * fuelequiv) * kh2o * da_dn5 / ((kh2o * a + 1)^2);
```

```
% dn2/dn5: n2 = (hydrogen/2)*fuelequiv - n4
```

```
dn2_dn5 = -dn4_dn5;
```

```
% dn5_calc/dn5: n5_calc = (molar_air_stoich*2 + oxygen*fuelequiv - 2*n1 - n2 - n3) / 2
```

```
dn5_calc_dn5 = (-2*dn1_dn5 - dn2_dn5 - dn3_dn5) / 2;
```

```
dntsum_dn5 = dn1_dn5 + dn2_dn5 + dn3_dn5 + dn4_dn5 + dn5_calc_dn5;    % n6 is  
constant (nitrogen doesn't depend on n5)
```

```
% Now apply quotient rule for d(n5_calc/ntsum)/dn5
```

```
d_ratio_dn5 = (ntsum * dn5_calc_dn5 - n5_calc * dntsum_dn5) / (ntsum^2);
```

```
% Final derivative
```

```
df = 1 - d_ratio_dn5;
```

end

function f =

mydissfun_residual(n5,carbon,hydrogen,oxygen,temperature,fuelequiv,molar_air_stoich,
pressure)

% Inputs:

% n5_guess - Guessed value of n_5/n_t (oxygen mole fraction * total moles)

% carbon - Carbon atoms in fuel

% hydrogen - Hydrogen atoms in fuel

% oxygen - Oxygen atoms in fuel

% temperature - Combustion temperature [K]

% fueleviv - Fuel equivalence ratio

% molar_air_stoich - Stoichiometric air requirement

% pressure - Combustion pressure [bar]

% Gibbs Free Energy Data

gibbseng = [0 -393.151 -238.921 -113.805

100 -393.683 -236.584 -120.239

200 -394.085 -232.766 -128.526

298.15 -394.389 -228.582 -137.163

300 -394.394 -228.5 -137.328

400 -394.675 -223.901 -146.338

500 -394.939 -219.051 -155.414

600 -395.182 -214.007 -164.486

700 -395.398 -208.812 -173.518
800 -395.586 -203.496 -182.497
900 -395.748 -198.083 -191.416
1000 -395.886 -192.59 -200.275
1100 -396.001 -187.033 -209.075
1200 -396.098 -181.425 -217.819
1300 -396.177 -175.774 -226.509
1400 -396.24 -170.089 -235.149
1500 -396.288 -164.376 -243.74
1600 -396.323 -158.639 -252.284
1700 -396.344 -152.883 -260.784
1800 -396.353 -147.111 -269.242
1900 -396.349 -141.325 -277.658
2000 -396.333 -135.528 -286.034
2100 -396.304 -129.721 -294.372
2200 -396.262 -123.905 -302.672
2300 -396.209 -118.082 -310.936
2400 -396.142 -112.252 -319.165
2500 -396.062 -106.416 -327.358
2600 -395.969 -100.575 -335.517
2700 -395.862 -94.729 -343.643
2800 -395.742 -88.878 -351.736
2900 -395.609 -83.023 -359.797
3000 -395.461 -77.163 -367.826
3100 -395.298 -71.298 -375.824

3200	-395.122	-65.43	-383.792
3300	-394.932	-59.558	-391.73
3400	-394.726	-53.681	-399.638
3500	-394.506	-47.801	-407.517
3600	-394.271	-41.916	-415.366
3700	-394.022	-36.027	-423.188
3800	-393.756	-30.133	-430.981
3900	-393.477	-24.236	-438.746
4000	-393.183	-18.334	-446.485
4100	-392.874	-12.427	-454.196
4200	-392.55	-6.516	-461.881
4300	-392.21	-0.6	-469.539
4400	-391.857	5.32	-477.171
4500	-391.488	11.245	-484.778
4600	-391.105	17.175	-492.359
4700	-390.706	23.111	-499.915
4800	-390.292	29.052	-507.446
4900	-389.862	34.998	-514.951
5000	-389.419	40.949	-522.433
5100	-388.959	46.906	-529.89
5200	-388.486	52.869	-537.323
5300	-387.996	58.838	-544.732
5400	-387.493	64.811	-552.118
5500	-386.974	70.791	-559.48
5600	-386.439	76.777	-566.818

5700 -385.89 82.769 -574.133

5800 -385.324 88.767 -581.424

5900 -384.745 94.77 -588.694

6000 -384.148 100.78 -595.939];

% gibbs free energy

gco2 = interp1(gibbseng(:,1), gibbseng(:,2), temperature) * 1e3; % Convert kJ to J

gh2o = interp1(gibbseng(:,1), gibbseng(:,3), temperature) * 1e3;

gco = interp1(gibbseng(:,1), gibbseng(:,4), temperature) * 1e3;

R = 8314.46; % Universal gas constant [J/(kmol ·K)]

%Equilibrium Constants

kco2 = exp(-(gco2 - gco) * 1e3 / (R * temperature));

kh2o = exp(-gh2o * 1e3 / (R * temperature));

% Moles using Current n5 Guess

a = sqrt(n5 * pressure);

% CO2 moles (n1) considering dissociation

n3 = (carbon * fuelequiv) / (kco2 * a + 1); % CO moles from dissociation

n1 = carbon * fuelequiv - n3; % CO2 moles remaining

% H2O moles (n2) considering dissociation

n4 = ((hydrogen/2) * fuelequiv) / (kh2o * a + 1); % H2 moles from dissociation

```

n2 = (hydrogen/2) * fuelequiv - n4;           % H2O moles remaining

% O2 moles (n5) from oxygen balance

n5_calc = (molar_air_stoich * 2 + oxygen * fuelequiv - 2*n1 - n2 - n3) / 2; % At the
correct solution, this should be zero

% N2 moles (n6)

n6 = molar_air_stoich * 3.76;

% Total Moles

ntsum = n1 + n2 + n3 + n4 + n5_calc + n6;

% Residual Error

f = n5 - (n5_calc / ntsum); % At the correct solution, this should be zero

end

function [root, iter, final_tol] = bisect_3(c, h, o, temp_test, feq_test, airs, press_test)

    tol = 1e-6;

    max_iter = 100;

    iter = 0;

    % Define initial bracket [a, b]

    a = 0;           % Lower bound

```

```
b = 1;          % Upper bound

% Verify bracket contains a root (f(a) and f(b) have opposite signs)

fa = mydissfun_residual(a, c, h, o, temp_test, feq_test, airs, press_test);

fb = mydissfun_residual(b, c, h, o, temp_test, feq_test, airs, press_test);

if fa * fb > 0

    warning('Bisection: Initial bracket does not contain root! Adjusting...');

    a = 1e-8;

    b = 1e-2;

    fa = mydissfun_residual(a, c, h, o, temp_test, feq_test, airs, press_test);

    fb = mydissfun_residual(b, c, h, o, temp_test, feq_test, airs, press_test);

    fprintf(' New bracket: [%.2e, %.2e]\n', a, b);

    fprintf(' f(a) = %.2e, f(b) = %.2e\n', fa, fb);

    if fa * fb > 0

        error('Bisection: Cannot find valid bracket!');

    end

end

% Bisection loop

while iter < max_iter

    iter = iter + 1;

    n5 = (a + b) / 2;

    f = mydissfun_residual(n5, c, h, o, temp_test, feq_test, airs, press_test);
```

```
    final_tol = abs(b - a);

    % Check convergence

    if final_tol < tol

        root = n5;

        return;

    end

    % Determine which half contains the root

    if f * fa < 0

        % Root is in left half

        b = n5;

        fb = f;

    else

        % Root is in right half

        a = n5;

        fa = f;

    end

end

% If we get here, max iterations reached

warning('Bisection: Maximum iterations (%d) reached', max_iter);

root = (a+b)/2;

end
```

```

function [n5, iterations, final_tol] = secant_3(c, h, o, temp_test, feq_test, airs, press_test)

    tol = 1e-6;      % Convergence tolerance

    max_iter = 100;  % Maximum iterations

    % Initial guesses (two starting points near expected value)

    n5_old = 1e-10;  % First guess

    n5 = 1;          % Second guess

    for iterations = 1:max_iter

        % Evaluate function at both points

        f = mydissfun_residual(n5, c, h, o, temp_test, feq_test, airs, press_test); %
        residual error at second guess

        f_old = mydissfun_residual(n5_old, c, h, o, temp_test, feq_test, airs, press_test);
        % residual error at first guess

        % Check for division by zero

        if abs(f - f_old) < 1e-15

            warning('Secant: Denominator too small at iteration %d', iterations);

            break;

        end

        % Secant method update:  $x_{\text{new}} = x - f(x) * (x - x_{\text{old}}) / (f(x) - f(x_{\text{old}}))$ 

        n5_new = n5 - f * (n5 - n5_old) / (f - f_old);

```

```
% Ensure n5 stays positive (physical constraint)

if n5_new < 0

    n5_new = abs(n5_new); % Take absolute value

end

% Calculate tolerance

final_tol = abs(n5_new - n5);

% Check convergence

if final_tol < tol

    n5 = n5_new;

    return;

end

% Update for next iteration

n5_old = n5;

n5 = n5_new;

end

% If we get here, max iterations reached

warning('Secant: Maximum iterations (%d) reached without convergence', max_iter);

end

function [n5, iterations, final_tol] = fixedpoint_3(c, h, o, temp_test, feq_test, airs,
press_test)
```

```
tol = 1e-6;      % Convergence tolerance

max_iter = 100;  % Maximum iterations


% Initial guess

n5 = 0; % Start near expected value


for iterations = 1:max_iter

    % Apply fixed-point function: n5_new = g(n5)

    n5_new = fixedpoint_function_3(n5, c, h, o, temp_test, feq_test, airs, press_test);


    % Ensure n5 stays positive and bounded (physical constraints)

    if n5_new < 0

        n5_new = abs(n5_new);

    end

    if n5_new > 1 % Unrealistic value (mole fraction > 1)

        n5_new = n5 * 0.9; % Reduce instead

    end


    % Calculate tolerance

    final_tol = abs(n5_new - n5);


    % Check convergence

    if final_tol < tol

        n5 = n5_new;

        return;
```

```

end

% Update for next iteration

n5 = n5_new;

end

% If we get here, max iterations reached

warning('Fixed-point: Maximum iterations (%d) reached without convergence',
max_iter);

end

function n5_new = fixedpoint_function_3(n5, c, h, o, temp_test, feq_test, airs, press_test)

% Gibbs Free Energy Data [K, CO2, H2O, CO] in kJ/kmol

gibbseng = [

0      -393.151 -238.921 -113.805

100    -393.683 -236.584 -120.239

200    -394.085 -232.766 -128.526

298.15 -394.389 -228.582 -137.163

300    -394.394 -228.5   -137.328

400    -394.675 -223.901 -146.338

500    -394.939 -219.051 -155.414

600    -395.182 -214.007 -164.486

700    -395.398 -208.812 -173.518

800    -395.586 -203.496 -182.497

```

```

900    -395.748 -198.083 -191.416
1000   -395.886 -192.59  -200.275
1100   -396.001 -187.033 -209.075
1200   -396.098 -181.425 -217.819
1300   -396.177 -175.774 -226.509
1400   -396.24  -170.089 -235.149
1500   -396.288 -164.376 -243.74
1600   -396.323 -158.639 -252.284
1700   -396.344 -152.883 -260.784
1800   -396.353 -147.111 -269.242
1900   -396.349 -141.325 -277.658
2000   -396.333 -135.528 -286.034

```

```
];
```

```
% Interpolate Gibbs free energy at given temperature
```

```
gco2 = interp1(gibbseng(:,1), gibbseng(:,2), temp_test) * 1e3; % Convert kJ to J
```

```
gh2o = interp1(gibbseng(:,1), gibbseng(:,3), temp_test) * 1e3;
```

```
gco = interp1(gibbseng(:,1), gibbseng(:,4), temp_test) * 1e3;
```

```
% Universal gas constant [J/(kmol ·K)]
```

```
R = 8314.46;
```

```
% Calculate equilibrium constants
```

```
kco2 = exp(-(gco2 - gco) * 1e3 / (R * temp_test)); %  $\text{CO}_2 \rightleftharpoons \text{CO} + 0.5\text{O}_2$ 
```

```
kh2o = exp(-gh2o * 1e3 / (R * temp_test)); %  $\text{H}_2 + 0.5\text{O}_2 \rightleftharpoons \text{H}_2\text{O}$ 
```

```
% Calculate intermediate variable

a = sqrt(n5 * press_test);

% Calculate species moles considering dissociation

n3 = (c * feq_test) / (kco2 * a + 1);          % CO moles

n1 = c * feq_test - n3;                        % CO2 moles

n4 = ((h/2) * feq_test) / (kh2o * a + 1);      % H2 moles

n2 = (h/2) * feq_test - n4;                    % H2O moles

% Calculate O2 moles from oxygen balance

n5_calc = (airs * 2 + o * feq_test - 2*n1 - n2 - n3) / 2;

% N2 moles (inert, doesn't participate in reactions)

n6 = airs * 3.76;

% Total moles

ntsum = n1 + n2 + n3 + n4 + n5_calc + n6;

% Fixed-point iteration: n5_new = n5_calc/ntsum

% This is the reformulation of  $f(n5) = n5 - n5\_calc/ntsum = 0$ 

n5_new = n5_calc / ntsum;

end
```

4.2 Own code task 3b