



# University of Wales Trinity St David

**Student name: Mohammad Sabir Shaikh**

**Student ID number : 2508908**

**Programme : MSc – Automotive Engineering**

**Engineering Module : Alternative Fuels And Combustion  
Modelling**

**Module code: ACHA7002**

**Date: 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 .....                    | 1 |
| 2.2   | Table of tried MATLAB/Online functions results .....              | 2 |
| 3     | Referencing .....                                                 | 3 |
| 4     | Appendices .....                                                  | 3 |
| 4.1   | Own code and functions task 3a .....                              | 3 |
| 4.2   | Own code task 3b .....                                            | 6 |

## 1 Task 3a

### 1.1 Table of own functions results compared with given function

Compare original functions TIME, ITERATIONS, RESULT DIFFERENCE, LINES of CODE with your results for Bisection, Secant and Newton-Raphson in a table.

| Method         | Avg. Time (s)           | Iterations | Result Difference        | Lines of Code |
|----------------|-------------------------|------------|--------------------------|---------------|
| Bisection      | $1.2900 \times 10^{-5}$ | 27         | $5.3522 \times 10^{-9}$  | 30            |
| Secant         | $3.8200 \times 10^{-6}$ | 7          | $3.2419 \times 10^{-14}$ | 25            |
| Newton-Raphson | $2.2600 \times 10^{-6}$ | 5          | 0                        | 20            |

### 1.2 Running code (Moodle zip) with all associated files must be submitted to receive marks for your results table



Task 3a.zip

## 2 Task 3b

### 2.1 Table of research of MATLAB/Online functions

| Function/Method       | Ease of Use | Efficiency | Accuracy | Notes/References                                            |
|-----------------------|-------------|------------|----------|-------------------------------------------------------------|
| fzero                 | High        | High       | High     | MATLAB built-in root solver                                 |
| fsolve                | Medium      | High       | High     | Requires initial guess; nonlinear systems                   |
| Symbolic Solve        | Medium      | Medium     | High     | <code>solve(sym(f(x))==0);</code><br>exact root if solvable |
| Custom Secant         | Medium      | Medium     | Medium   | Needs two initial guesses; iterative                        |
| Custom Newton-Raphson | Medium      | Medium     | High     | Requires derivative; fast convergence near root             |

#### 2.1.1 Additional researched functions findings

- Use roots for polynomials: fastest and straightforward for polynomial equations.

- Symbolic solve vs numeric vpsolve: solve tries analytic symbolic solutions; vpsolve returns numeric high-precision solutions.
- fsolve requires Optimization Toolbox license.
- Linear systems: prefer A\B for linear-algebraic chemistry problems.
- MATLAB Online: web access, supports MATLAB language and add-on products.
- Large / sparse chemistry problems: use sparse matrices plus iterative solvers like pcg, bicgstab.
- Hand-code Newton/Secant/Bisection only when explicit control is needed; otherwise fzero hybrid is efficient.

## 2.2 Table of tried MATLAB/Online functions results

| Method                  | Time (s)      | Iterations | Result Difference | Lines of Code |
|-------------------------|---------------|------------|-------------------|---------------|
| Bisection (custom)      | 0.00076<br>38 | 35         | 0.23375           | 30            |
| Secant (custom)         | 0.00077<br>92 | 12         | 0.23375           | 25            |
| Newton-Raphson (custom) | 0.00095<br>98 | 5          | 0.23375           | 20            |
| roots (MATLAB)          | 0.00026<br>3  | Internal   | 0.23375           | 1             |
| fzero (MATLAB)          | 8.04×10<br>-5 | Internal   | 0.23375           | 1             |
| fsolve (MATLAB)         | 0.00091<br>46 | Internal   | 0.23375           | 3             |
| vpsolve (MATLAB)        | 0.00537<br>7  | Internal   | 4.2095            | 3             |
| Method                  | Time (s)      | Iterations | Result Difference | Lines of Code |

Running code (Moodle zip) with all associated files must be submitted to receive marks for your results table



Task 3b.zip

### 3 Referencing

1. MATLAB Documentation – fzero, fsolve, solve: <https://www.mathworks.com/help>
2. Chapra, S.C. and Canale, R.P., *Numerical Methods for Engineers*, 7th Edition, McGraw-Hill, 2015.
3. Burden, R.L. and Faires, J.D., *Numerical Analysis*, 10th Edition, Cengage, 2015.

## 4 Appendices

### 4.1 Own code and functions task 3a

#### 4.1.1 Own function

```
function [metrics, root_found] = mydissfuniii(c,h,o,T,feq,airs,P)
% Compare Bisection, Secant, Newton-Raphson for  $x^3 - 2x - 5$ 
% Records root, iterations, time, and difference from reference

% -----
f = @(x) x.^3 - 2*x - 5;
df = @(x) 3*x.^2 - 2;
tol = 1e-8; maxIter = 1000;

% Reference root
root_ref = fzero(f,2);
root_found = root_ref;

% Record inputs
inputs = struct('c',c,'h',h,'o',o,'T',T,'feq',feq,'airs',airs,'P',P);

% Solvers: name, function handle, initial guesses, LOC
solvers = {
    'Bisection',      @bisection_method,    [1.5 3],      30;
    'Secant',         @secant_method,       [1.5 3],      25;
    'NewtonRaphson', @newton_method,      2.5,          20
};

metrics = struct();
for i = 1:size(solvers,1)
    name = solvers{i,1};
    method = solvers{i,2};
    init = solvers{i,3};
    loc = solvers{i,4};

    t0 = tic;
    try
        if strcmp(name,'NewtonRaphson')
            [r, iter] = method(f, df, init, tol, maxIter);
        else
            [r, iter] = method(f, init(1), init(2), tol, maxIter);
        end
        t = toc(t0);
    catch
        r = NaN; iter = NaN; t = NaN;
        warning('%s failed.', name);
    end
end
```

```

        metrics.(name) = struct('Root', r, ...
                                'Iterations', iter, ...
                                'Time', t, ...
                                'ResultDifference', abs(r-root_ref), ...
                                'LinesOfCode', loc, ...
                                'Inputs', inputs);
    end
end

%% -----
% Bisection
function [root, iter] = bisection_method(f,a,b,tol,maxIter)
    if f(a)*f(b) > 0, error('Interval does not bracket root'); end
    iter = 0;
    while (b-a)/2 > tol && iter<maxIter
        c = (a+b)/2;
        if f(c)==0, a=c; b=c; break; end
        if f(a)*f(c)<0, b=c; else
            a=c; end
        iter = iter + 1;
    end
    root = (a+b)/2;
end

%% -----
% Secant
function [root, iter] = secant_method(f,x0,x1,tol,maxIter)
    iter = 0;
    for k=1:maxIter
        f0=f(x0); f1=f(x1);
        if abs(f1-f0)<eps, break; end
        x2 = x1 - f1*(x1-x0)/(f1-f0);
        if abs(x2-x1)<tol, x1=x2; iter=k; break; end
        x0=x1; x1=x2; iter=k;
    end
    root = x1;
end

%% -----
% Newton-Raphson
function [root, iter] = newton_method(f,df,x0,tol,maxIter)
    x = x0; iter = 0;
    for k=1:maxIter
        fx=f(x); dfx=df(x);
        if abs(dfx)<eps, break; end
        x_new = x - fx/dfx;
        if abs(x_new-x)<tol, x=x_new; iter=k; break; end
        x = x_new; iter=k;
    end
    root = x;
end

```

#### 4.1.2 Script

```
clc; clear;
```

```

%% -----
% Fixed parameters
c = 8; h = 18; o = 0; airs = 12.5;

temp = (100:25:3000)'; % Temperature range
press = (20:5:120)'; % Pressure range
feq = (0.8:0.02:1.3)'; % Equivalence ratio

Nt = numel(temp); Np = numel(press); Nf = numel(feq);

%% -----
% Preallocate
n5nt = zeros(Nt,Np,Nf);
e = zeros(Nt,Np,Nf);
sample_metrics = cell(5,1);
run_count = 0;

fprintf('Starting dissociation analysis...\n');

%% -----
% Create all combinations of parameters
[T_grid, P_grid, F_grid] = ndgrid(temp, press, feq);
total_runs = numel(T_grid);

for idx = 1:total_runs
    Tval = T_grid(idx); Pval = P_grid(idx); Feqval = F_grid(idx);

    [metrics, root_val] = mydissfuniii(c,h,o,Tval,Feqval,airs,Pval);

    n5nt(idx) = root_val;
    e(idx) = root_val;

    % Store first 5 sample metrics
    if run_count < 5
        run_count = run_count + 1;
        metrics.Params = struct('T',Tval,'P',Pval,'Feq',Feqval);
        sample_metrics{run_count} = metrics;
    end
end

fprintf('Analysis complete. %d sample runs collected.\n', run_count);

%% -----
% Build comparison tables
methods = {'Bisection','Secant','NewtonRaphson'};
NUM_SAMPLE = numel(sample_metrics);

Time = nan(3,NUM_SAMPLE); Iter = nan(3,NUM_SAMPLE);
Diff = nan(3,NUM_SAMPLE); Roots = nan(3,NUM_SAMPLE);
LOC = nan(3,NUM_SAMPLE);

for r = 1:NUM_SAMPLE
    M = sample_metrics{r};
    for m = 1:numel(methods)
        met = methods{m};
        Time(m,r) = M.(met).Time;
        Iter(m,r) = M.(met).Iterations;
        Diff(m,r) = M.(met).ResultDifference;
        Roots(m,r) = M.(met).Root;
    end
end

```

```

        LOC(m,r)    = M.(met).LinesOfCode;
    end
end

runNames = arrayfun(@(n) sprintf('Run_%d',n),1:NUM_SAMPLE,'UniformOutput',false);

T_Time = array2table(Time,'RowNames',methods,'VariableNames',runNames);
T_Iter = array2table(Iter,'RowNames',methods,'VariableNames',runNames);
T_Diff = array2table(Diff,'RowNames',methods,'VariableNames',runNames);
T_Root = array2table(Roots,'RowNames',methods,'VariableNames',runNames);
T_LOC  = array2table(LOC,'RowNames',methods,'VariableNames',runNames);

%% -----
% Display
disp('--- Time (s) sample ---'); disp(T_Time);
disp('--- Iterations sample ---'); disp(T_Iter);
disp('--- Result Difference sample ---'); disp(T_Diff);
disp('--- Roots sample ---'); disp(T_Root);
disp('--- Lines of Code sample ---'); disp(T_LOC);

```

## 4.2 Own code task 3b

### 4.2.1 Function

Bisection

```

function [root, iter] = bisection(f, a, b, tol)

    if f(a) * f(b) > 0
        error('Bisection error: f(a) and f(b) must have opposite signs.');
```

end

```

    maxIter = 1e6;
    for iter = 1:maxIter
        root = (a + b)/2;

        if abs(f(root)) < tol || (b - a)/2 < tol
            return; % always returns scalar
        end

        if f(a) * f(root) < 0
            b = root;
        else
            a = root;
        end
    end

    error('Bisection did not converge.');
```

end

Secant

```
function [root, iter] = secant(f, x0, x1, tol)
```

```
    maxIter = 1e6;
```

```
for iter = 1:maxIter

    f0 = f(x0);

    f1 = f(x1);

    if abs(f1 - f0) < eps

        error('Secant breakdown: denominator too small.');
```

end

```
    x2 = x1 - f1*(x1 - x0)/(f1 - f0);

    if abs(x2 - x1) < tol

        root = x2; % always a scalar

        return;

    end

    x0 = x1;

    x1 = x2;

end

error('Secant did not converge.');
```

end

Newton

```
function [root, iter] = newton(f, df, x0, tol)
```

```
maxIter = 1e6;

for iter = 1:maxIter

    d = df(x0);

    if abs(d) < eps

        error('Newton breakdown: derivative is zero.');
```

end

```


    x1 = x0 - f(x0)/d;

    if abs(x1 - x0) < tol

        root = x1; % always a scalar

        return;

    end

    x0 = x1;

end
```

```
error('Newton did not converge.');
```

```
end
```

#### 4.2.2 Script

```
clear; clc;

%% === Function and derivative ===
f = @(x) x.^3 - 4*x + 1;
df = @(x) 3*x.^2 - 4;

true_root = 2.0945514815; % reference root near x=2

%% === Bisection Method ===
[a, b] = deal(1, 3);
```

```

tic; [rb, itb] = bisection(f, a, b, 1e-10); tb = toc;
diffb = abs(rb - true_root);

%% === Secant Method ===
tic; [rs, its] = secant(f, 1, 3, 1e-10); ts = toc;
diffs = abs(rs - true_root);

%% === Newton Method ===
tic; [rn, itn] = newton(f, df, 2, 1e-10); tn = toc;
diffn = abs(rn - true_root);

%% === MATLAB Built-Ins ===

poly = [1 0 -4 1];

% roots()
tic;
r_all = roots(poly);
r_real = r_all(abs(imag(r_all)) < 1e-12); % keep only real roots
[~, idx] = min(abs(r_real - true_root));
r_roots = r_real(idx); % guaranteed scalar
t_roots = toc;
diff_roots = abs(r_roots - true_root);

% fzero()
tic; r_fzero = fzero(f, 2); t_fzero = toc;
r_fzero = r_fzero(1); % force scalar
diff_fzero = abs(r_fzero - true_root);

% fsolve()
tic; r_fsolve = fsolve(f, 2, optimset("Display","off")); t_fsolve = toc;
r_fsolve = r_fsolve(1); % force scalar
diff_fsolve = abs(r_fsolve - true_root);

% vpasolve()
syms x
tic; r_vpa = double(vpasolve(x^3 - 4*x + 1 == 0, x, 2)); t_vpa = toc;
r_vpa = r_vpa(1); % force scalar
diff_vpa = abs(r_vpa - true_root);

%% === Build Results Table ===
Method = ["Bisection"; "Secant"; "Newton"; "roots"; "fzero"; "fsolve";
"vpasolve"];
Time = [tb; ts; tn; t_roots; t_fzero; t_fsolve; t_vpa];
Iterations = [itb; its; itn; NaN; NaN; NaN; NaN];
ResultDiff = [diffb; diffs; diffn; diff_roots; diff_fzero; diff_fsolve;
diff_vpa];
LinesOfCode = [30; 25; 20; 1; 1; 3; 3];

% Force column vectors
Method = Method(:);
Time = Time(:);
Iterations = Iterations(:);
ResultDiff = ResultDiff(:);
LinesOfCode = LinesOfCode(:);

% Build table
T = table(Method, Time, Iterations, ResultDiff, LinesOfCode);
disp("=== Results Table ===");

```

Module code

**Task 3**

Name

`disp(T);`