



prifysgol cymru trinity saint devid

University of Wales Trinity St David

Student name: Anandu Unnikrishanan Nair

Student ID number : 2411630

Programme : MSc – Automotive Engineering

Module : Alternative fuels and Combustion Modelling

Module code: ACHA7002

Task 3

Contents

1	Task 3a	1
1.1	Table of own functions results compared with given function.....	1
2	Task 3b	2
2.1	Table of research of MATLAB/Online functions	2
2.1.1	Additional researched functions findings	4
2.2	Table of tried MATLAB/Online functions results	4
3	Referencing	5
4	Appendices.....	6
4.1	Own code and functions task 3a	6
4.2	Own code task 3b	10

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

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

numerical root-solving techniques were applied to compute the dissociation oxygen parameter n_5 .

The original function mydissfuniii.m uses a long step-based algorithm to reduce the error:

$$F(x) = x - \frac{n_5(x)}{n_{\text{total}}(x)}$$

This behaviour was reproduced by constructing a cleaner root function n5_equation.m, which allowed the use of:

- **Bisection Method**
- **Secant Method**
- **Newton–Raphson Method**

The goal was to compare each technique based on:

- Computational time
- Number of iterations
- Accuracy versus the original solver

Command Window				
Method	n5	Iter	Time(s)	Diff vs orig
Original	6.37459520e-04	-	0.028149	-
Bisection	6.37459452e-04	30	0.020720	6.7698e-11
Secant	6.37459513e-04	8	0.010022	6.6839e-12
Newton	6.37459514e-04	6	0.003909	5.8792e-12

fx >>

The results indicate that all three numerical techniques match the original dissociation code extremely well, with differences between:

$$10^{-11} \text{ and } 10^{-12}$$

This demonstrates that the created root function accurately reproduces the physical behaviour of the original function. Bisection needed the maximum number of iterations (30) and provided a reasonable calculation time. Secant converged considerably faster with only 8 iterations. Newton–Raphson gave the fastest convergence (6 iterations) and the shortest calculation time (0.0039s).

The Newton technique is the most efficient since it uses derivative information, whereas Bisection is guaranteed to converge but remains the slowest.

Because of its speed and accuracy, Newton is the best approach for this problem overall.

2 Task 3b

2.1 Table of research of MATLAB/Online functions

Research MATLAB functions and mathematical techniques to calculate n_5 , there are many chemistry linear algebra solvers, symbolic maths solvers and engineering root solvers. You should create a table of methods ranking them based on criteria such as ease of use, efficiency, and accuracy. This table should be referenced

Method	Type	Ease	Efficiency	Accuracy	Short comments
fzero	1D root solver (MATLAB)	5	4	4	Built-in scalar root finder for $f(x)=0$. Uses a combination of bisection, secant and inverse quadratic interpolation. Very easy to call with a function handle and an initial guess.

fsolve	Multivariate nonlinear solver	3	4	4	Solves systems ($F(x)=0$). Suitable for coupled equations or larger thermodynamic models. Requires Optimization Toolbox and a reasonable initial guess. Often more than needed for a single unknown.
vpasolve	Symbolic numeric solver	3	3	5	Part of the Symbolic Math Toolbox. Works with symbolic equations and returns high-precision numeric solutions. Very accurate but slower and requires extra toolbox.
Cantera / CEA	External combustion equilibrium codes	2	5	5	Specialist chemistry packages that compute full equilibrium compositions and thermodynamic properties. Highly accurate and efficient for complex mixtures but require separate installation and more complex setup than basic MATLAB functions.

Fzero is the most logical solution for the current issue among the approaches examined since appears as a single scalar unknown in the dissociation model. Adding more species or associated temperature/pressure equations makes multivariate solvers like fsolve or external equilibrium codes (Cantera/CEA) more appealing. Symbolic approaches like vpasolve can reach very high accuracy, but double-precision numerical methods are already sufficient here, as they match the original routine to roughly 10–12.

2.1.1 Additional researched functions findings

Task goal is to investigate several MATLAB and mathematical methods that can calculate the dissociation oxygen term n_5 . Due to the fact that a nonlinear implicit equation can be used to explain the problem:

$$F(x) = x - \frac{n_5(x)}{n_{\text{total}}(x)} = 0$$

Many built-in solvers or scientific methods can be applied.

Five representative methods were researched, evaluated, and ranked based on:

- Ease of use
- Efficiency
- Accuracy
- Availability within MATLAB
- Suitability for combustion/dissociation problems

2.2 Table of tried MATLAB/Online functions results

Compare original functions TIME, ITERATIONS, RESULT DIFERENCE, LINES of CODE with your results for researched code functions

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

Command Window

```
Results for fzero method:
Original n5 = 6.3745951999e-04
fzero n5    = 6.3745951837e-04
f(x)        = 8.554e-17
Iterations   = 10
Time orig    = 0.026972
Time fzero   = 0.140778
Difference   = 1.620e-12
fx >> |
```

The MATLAB function `fzero` successfully replicated the original n_5 value with a very modest inaccuracy of 1.62×10^{-12} . This shows that the precision of the lengthy, manually-programmed step-based procedure in `mydissfuniii.m` can be matched by MATLAB's built-in root-solving algorithms. Because the `fzero` routine executes extra internal logic for robustness, it took ten iterations to converge and ran a little more slowly. However, it took fewer than 10 lines of code compared to the nearly 250-line original function, making it considerably simpler and cleaner to utilize in future projects.

3 Referencing

- MathWorks (2023) *fzero* — *Root Finding*.
<https://uk.mathworks.com/help/matlab/ref/fzero.html>
- MathWorks (2023) *fsolve* — *Nonlinear Equation Solver*.
<https://uk.mathworks.com/help/optim/ug/fsolve.html>
- MathWorks (2023) *trapz* — *Trapezoidal Numerical Integration*.
<https://uk.mathworks.com/help/matlab/ref/trapz.html>
- Goodwin, D.G. et al. (2022) *Cantera: Chemical Kinetics Toolkit*. Available at:
<https://cantera.org>
- Chapra, S.C. and Canale, R.P. (2015) *Numerical Methods for Engineers*. 7th edn. McGraw-Hill.
- NASA Glenn Research Center (2014) *Chemical Equilibrium with Applications (CEA)*. NASA Reference Publication 1311.

4 Appendices

4.1 Own code and functions task 3a

```

n5_equation.m * +
1 function F = n5_equation(n5ntvalue, carbon, hydrogen,
2
3 % ---- COPY YOUR GIBBS TABLE FROM mydissfuniii HERE -
4 gibbseng=[ ...
5     0   -393.151   -238.921   -113.805
6     100 -393.683   -236.584   -120.239
7     200 -394.085   -232.766   -128.526
8     298.15 -394.389   -228.582   -137.163
9     300 -394.394   -228.5   -137.328
10    400 -394.675   -223.901   -146.338
11    500 -394.939   -219.051   -155.414
12    600 -395.182   -214.007   -164.486
13    700 -395.398   -208.812   -173.518
14    800 -395.586   -203.496   -182.497
15    900 -395.748   -198.083   -191.416
16   1000 -395.886   -192.59   -200.275
17   1100 -396.001   -187.033   -209.075
18   1200 -396.098   -181.425   -217.819
19   1300 -396.177   -175.774   -226.509
20   1400 -396.24   -170.089   -235.149
21   1500 -396.288   -164.376   -243.74
22   1600 -396.323   -158.639   -252.284
23   1700 -396.344   -152.883   -260.784
24   1800 -396.353   -147.111   -269.242
25   1900 -396.349   -141.325   -277.658
26   2000 -396.333   -135.528   -286.034
27   2100 -396.304   -129.721   -294.372
28   2200 -396.262   -123.905   -302.672
29   2300 -396.209   -118.082   -310.936
30   2400 -396.142   -112.252   -319.165
31   2500 -396.062   -106.416   -327.358
32

```

```

n5_equation.m
62      5600      -386.439      76.777      -566.818
63      5700      -385.89      82.769      -574.133
64      5800      -385.324      88.767      -581.424
65      5900      -384.745      94.77      -588.694
66      6000      -384.148      100.78      -595.939];
67
68      % Gibbs free energies (J/mol)
69      gco2 = interp1(gibbseng(:,1), gibbseng(:,2), temperature)*1e3;
70      gh2o = interp1(gibbseng(:,1), gibbseng(:,3), temperature)*1e3;
71      gco = interp1(gibbseng(:,1), gibbseng(:,4), temperature)*1e3;
72
73      % Equilibrium constants
74      kco2 = exp((-gco2-gco)*1e3/(8314.46*temperature));
75      kh2o = exp((-gh2o)*1e3/(8314.46*temperature));
76
77      % Species calculations (same as in your mydissfuniii)
78      a = sqrt(n5ntvalue*pressure);
79      n3 = (carbon*fuelequiv)/(kco2*a + 1);
80      n4 = ((hydrogen/2)*fuelequiv)/(kh2o*a + 1);
81      n1 = carbon*fuelequiv - n3;
82      n2 = (hydrogen/2)*fuelequiv - n4;
83      n5 = (molar_air_stoich*2 + oxygen*fuelequiv - 2*n1 - n2 - n3)/2;
84      n6 = molar_air_stoich*3.76;
85
86      ntsum = n1 + n2 + n3 + n4 + n5 + n6;
87
88      % Root function: we want F = 0
89      F = n5ntvalue - n5/ntsum;
90      end
91

```

```
bisection_n5.m
function [root, iters] = bisection_n5(fun, a, b, tol, imax)
% Bisection method for scalar fun(x)=0

fa = fun(a);
fb = fun(b);

if fa*fb > 0
    error('Bisection: no sign change in [a,b]');
end

for k = 1:imax
    c = (a + b)/2;
    fc = fun(c);

    % store iteration count
    iters = k;

    if abs(fc) < tol || (b - a)/2 < tol
        root = c;
        return
    end

    if fa*fc < 0
        b = c;
        fb = fc;
    else
        a = c;
        fa = fc;
    end
end

root = (a + b)/2;
warning('Bisection: reached max iterations');
```

```
bisection_n5.m
26     else
27         a = c;
28         fa = fc;
29     end
30 end
31
32 root = (a + b)/2;
33 warning('Bisection: reached max iterations');
34 end
35
```

```
newton_n5.m x +
1 function [root, iters] = newton_n5(fun, x0, tol, imax)
2
3     h = 1e-6;
4
5     for k = 1:imax
6         f0 = fun(x0);
7
8         % numerical derivative (central difference)
9         df = (fun(x0 + h) - fun(x0 - h)) / (2*h);
10
11        if df == 0
12            error('Newton: zero derivative');
13        end
14
15        x1 = x0 - f0/df;
16        iters = k;
17
18        if abs(fun(x1)) < tol
19            root = x1;
20            return
21        end
22
23        x0 = x1;
24    end
25
26    root = x1;
27    warning('Newton: reached max iterations');
28    end
29
```

```
secant_n5.m x +
1 function [root, iters] = secant_n5(fun, x0, x1, tol, imax)
2
3     f0 = fun(x0);
4     f1 = fun(x1);
5
6     for k = 1:imax
7         if f1 - f0 == 0
8             error('Secant: zero denominator');
9         end
10
11        x2 = x1 - f1*(x1 - x0)/(f1 - f0);
12        f2 = fun(x2);
13
14        iters = k;
15
16        if abs(f2) < tol
17            root = x2;
18            return
19        end
20
21        x0 = x1; f0 = f1;
22        x1 = x2; f1 = f2;
23    end
24
25    root = x2;
26    warning('Secant: reached max iterations');
27    end
28
```

4.2 Own code task 3b

```
secant_n5.m  x  +
1  function [root, iters] = secant_n5(fun, x0, x1, tol, imax)
2
3      f0 = fun(x0);
4      f1 = fun(x1);
5
6  for k = 1:imax
7      if f1 - f0 == 0
8          error('Secant: zero denominator');
9      end
10
11      x2 = x1 - f1*(x1 - x0)/(f1 - f0);
12      f2 = fun(x2);
13
14      iters = k;
15
16      if abs(f2) < tol
17          root = x2;
18          return
19      end
20
21      x0 = x1; f0 = f1;
22      x1 = x2; f1 = f2;
23  end
24
25  root = x2;
26  warning('Secant: reached max iterations');
27  end
28
```