

UWTSD

Module

Task 3

Module code: ACHA7002

Student number: 2200899

7/12/2025

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.....	2
4	Appendices.....	3
4.1	Own code and functions task 3a	3
	Own code task 3b	8

1 Task 3a

1.1 Table of own functions results compared with given function

Table 1 – Comparison of varying root solving functions provided

	Original	Bisection	Secant	Newton Raphson
Run time (s)	1.748	0.337	0.353	0.479
Iterations	2101	30	56	31
Result difference	1e-9	9.313e-10	9.3949e-10	9.313e-10
Lines of code	216	117	113	173

All runs were done with a tolerance of 1e-9, as well as this it should be stated that the starting values

2 Task 3b

2.1 Table of research of MATLAB/Online functions

Table 2 - Alternative methods for root solving non-linear functions

Method	Ease of use	Efficiency	Accuracy
Trust-region Methods (Frontiers in Computing and Intelligent Systems, 2024)	Can be complex to implement	Can be efficient if the trust region and guess work is done correctly, otherwise it can reduce accuracy	Good assuming the local region around the guess is accurate
MATLAB fsolve function (MATLAB, date unknown)	Easy as it's a built in MATLAB function	Highly efficient in terms of amount of code, however due to it needing to recall a MATLAB function the overall time is slower	Very accurate as it can solve where a value crosses exactly
Brents Method (WOLFRAM, 2025)	Can be complex to set up however it often converges on a root	Can be very quick as it uses one of a multiple of 3 methods to find the solution (Bisection method, Secant method,	High level of accuracy as it can choose altering methods for which

		Inverse quadratic interpolation). However, it does need to decide between the three which solution is the best	one will give the best result.
--	--	--	--------------------------------

2.1.1 Additional researched functions findings

Other methods such as Gaussian method and Cramer's rule are alternative methods that can be applied as they work for linear methods (Nicholas J. Higham 2011).

However, due to the likelihood of a significant amount of the analysis working better through the use of non-linear functions.

2.2 Table of tried MATLAB/Online functions results

Table 3 - Alternative methods for root solving non-linear functions

	Trust region method	MATLAB fsolve function	Brent's method
Run time (s)	0.365	8.005	0.501
Iterations	36	Not applicable as built in MATLAB function	55
Result difference	1.760e-12	5.537e-16	7.451e-10
Lines of code	150	84	132

A threshold of under $1e-9$ was set for all the alternative methods apart from the MATLAB fsolve function as this uses its own convergence value.

3 Referencing

- FRONTIERS IN COMPUTING AND INTELLIGENT SYSTEMS, 2024. Overview of Trust-region Methods. **8**(3) (6/12/2025)
- MATLAB, fsolve. MATLAB (6/12/2025)
- NICHOLAS J. HIGHAM, 2011. Gaussian elimination. **3**, pp. 230–238. (6/12/2025)
- WOLFRAM, 2025. Brent's Method. (6/12/2025)

- WOLFRAM, Unconstrained Optimization: Methods for Solving Nonlinear Equations. (6/12/2025)

4 Appendices

4.1 Own code and functions task 3a

```

klc
clear

% ----- Original code ----- %

disp('Original code')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue]=mydissfuniii(c,h,o,temp(i,1),freq(b,1),airs,press(j,1));
        end
    end
end
disp('Original code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc

```

Figure 1 - Original code main script

```

function val = Task3_Function2(x, C, H, O, freq, airs, press, kco2, kh2o)

    a = sqrt(x*press);
    n3 = (C*freq)/(kco2*a + 1);
    n4 = ((H/2)*freq)/(kh2o*a + 1);
    n1 = C*freq - n3;
    n2 = (H/2)*freq - n4;
    n5 = (airs*2 + O*freq - 2*n1 - n2 - n3)/2;
    n6 = airs*3.76;
    nt = n1+n2+n3+n4+n5+n6;
    val = x - n5/nt;

end

```

Figure 2 - Task 3 function 2 (used in non-original sections)

```

% ----- Bisection code ----- %

disp('Bisection method')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue] = mydissfuniii_Bisection(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('Bisection code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc

```

Figure 3 - Bisection code main script

```

% ----- Secant ----- %

disp('Secant method')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue] = mydissfuniii_Secant(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('Secant code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc

```

Figure 4 - Secant code main script

```
% ----- Newton Rapshon ----- %

disp('Newton Rapshon method')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue] = mydissfuniii_Newton(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('Newton Rapshon code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc
```

Figure 5 - Newton Raphson main code

```
x1 = 0; % guess 1
x3 = 1; %guess 3

f_1 = Task3_Function2(x1, C, H, O, phi, airs, press, kco2, kh2o);
f_3 = Task3_Function2(x3, C, H, O, phi, airs, press, kco2, kh2o);

% Check for sign change
if f_1 * f_3 > 0
    warning('Bisection does not contain a root');
    iteration = 0;
    return;
end

% ---- Bisection loop ----
while abs(x3 - x1) > tolerance
    iteration = iteration + 1;

    x_2 = (x1 + x3)/2;
    f_2 = Task3_Function2(x_2, C, H, O, phi, airs, press, kco2, kh2o);

    if abs(f_2) < tolerance
        break;
    end
    if abs(x3 - x1) < tolerance
        break;
    end

    if f_1 * f_2 < 0
        x3 = x_2;
        f_3 = f_2;
    else
        x1 = x_2;
        f_1 = f_2;
    end
end

error = f_2;
n5ntvalue = x_2;

end
```

Figure 6 - Bisection method function 1

```
x1 = 1;
x2 = 0;

f1 = Task3_Function2(x1, C,H,O,feq,airs,press,kco2,kh2o);
f2 = Task3_Function2(x2, C,H,O,feq,airs,press,kco2,kh2o);

%Needs to use x2-x1 instead of f2 as f2 struggles to break out of the while
%loop without a max iteration condition at the end as well as the value
%being way below tolerance
while abs(x2-x1) > tolerance
    iteration = iteration + 1;
    x3 = x2 - f2 * (x2 - x1) / (f2 - f1);

    % Need to set a limit to the x3 value as it ends up being greater than
    % 1 or smaller than 0
    if x3<0
        x3=0;
    elseif x3>1
        x3=1;
    end

    x1 = x2;
    f1 = f2;

    x2 = x3;
    f2 = Task3_Function2(x2, C,H,O,feq,airs,press,kco2,kh2o);

if iteration > 200
    break;
end
end

error = f2;
n5ntvalue = x2;

end
```

Figure 7 - Secant method function 1

```
x1 = 1;
step = 1e-6;
tolerance = 1e-9;
iteration = 0;

while iteration < 200

    iteration = iteration + 1;

    f1 = Task3_Function2(x1, C,H,O,feq,airs,press,kco2,kh2o);

    if abs(x1) < tolerance
        break;
    end

    if x1 > step
        f_pos = Task3_Function2(x1 + step, C,H,O,feq,airs,press,kco2,kh2o);
        f_neg = Task3_Function2(x1 - step, C,H,O,feq,airs,press,kco2,kh2o);
        df1 = (f_pos - f_neg) / (2*step);
    else
        f_pos = Task3_Function2(x1 + step, C,H,O,feq,airs,press,kco2,kh2o);
        df1 = (f_pos - f1) / step;
    end

    % step o new value
    x_new = x1 - f1/df1;

    % New value
    if x_new < 0
        x_new = x1 / 2;
    end

    x1 = x_new;

end

n5ntvalue = x1;
```

Figure 8 - Newton method function 1

Own code task 3b

```

% PART B - Using new methods

disp('Part B using own methods')

% Trust region method

disp('Trust region method')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
nSnt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [nSnt(i,j,b),e(i,j,b),iteration,nSntvalue] = mydissfuniii_Trust(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('Trust region code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',nSntvalue)
toc

```

Figure 9 - Trust region method main code

```

x1_old = 1;          % starting guess
Delta = 0.05;        % trust region radius
Delta_max = 0.5;     % maximum radius (needs to be >= Delta)
region_threshold = 1e-10; % acceptance threshold (eta)
h = 1e-10;          % derivative step

while true

    iteration = iteration + 1;

    f1 = Task3_Function2(x1_old, C,H,O,feq,airs,press,kco2,kh2o);

    if abs(f1) < tolerance
        break
    end

    %Solves the max iteration issue
    if Delta < tolerance
        % warning('Delta collapsed');
        break
    end

    f_pos = Task3_Function2(x1_old + h, C,H,O,feq,airs,press,kco2,kh2o);
    f_neg = Task3_Function2(max(x1_old - h, 0), C,H,O,feq,airs,press,kco2,kh2o);
    df1 = (f_pos - f_neg) / ( (x1_old + h) - max(x1_old - h, 0) );

    %To control the step size which relates to how much x1_old and f change
    if abs(df1) < tolerance
        % small slope: take a small scaled step towards decreasing f (fallback)
        step_Newton = -f1 * Delta;
    else
        step_Newton = -f1 / df1;
    end

    % Apply trust region restriction and ensure trial x >= 0
    step = max(min(step_Newton, Delta), -Delta);
    x1 = max(0, x1_old + step); % clamp to physical domain

    f2 = Task3_Function2(x1, C,H,O,feq,airs,press,kco2,kh2o);

    % predicted and actual reduction (use absolute values)
    predicted_reduct = abs(f1) - abs(f1 + df1*(x1 - x1_old));
    actual_Reduct    = abs(f1) - abs(f2);

    % protect against zero predicted reduction
    if predicted_reduct <= 0
        ratio = -Inf;
    else
        ratio = actual_Reduct / predicted_reduct;
    end

    % ---- ACCEPT/REJECT STEP ----
    if ratio > region_threshold
        x1_old = x1; % accept step
    end

    % ---- UPDATE TRUST REGION ----
    if ratio > 0.75
        Delta = min(2 * Delta, Delta_max);
    elseif ratio < 0.25
        Delta = Delta / 2;
    end

    % extra termination: if step is tiny and f1 is small enough
    if abs(x1_old - x1) < tolerance && abs(f1) < tolerance
        break
    end

end

n5ntvalue = x1_old;

end

```

Figure 10 - Trust region method function 1

```
% MATLAB fsolve function

disp('fsolve function')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue] = mydissfuniii_fsolve(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('fsolve function code values')
disp('Number of iterations - Not applicable as built in matlab function')
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc
```

Figure 11 - fsolve MATLAB function main script

```
x = 1;

options = optimset('TolX', tolerance, 'Display', 'off'); % optional: control tolerance and suppress output

x_value = fsolve(@(x) Task3_Function2(x, C, H, O, freq, airs, press, kco2, kh2o), x, options);

n5ntvalue = x_value;

end
```

Figure 12 - fsolve MATLAB function 1

```
% Brents method

disp('Brent's method')

tic
c=8;
h=18;
o=0;
airs=12.5;
temp=(100:25:3000)';
press=(20:5:120)';
freq=(0.8:0.02:1.3)';
n5nt=zeros(49,4,26);
for i=1:49
    for j=1:21
        for b=1:26
            [n5nt(i,j,b),e(i,j,b),iteration,n5ntvalue] = mydissfuniii_Brents(c,h,o,temp(i),freq(b),airs,press(j));
        end
    end
end
disp('Brent's method code values')
fprintf('Number of iterations %.5e\n',iteration)
fprintf('Convergence/root value %.5e\n',n5ntvalue)
toc
```

Figure 13 - Brent's method main script

```

x1 = 0.15;
x2=0.05;
x3=-0.05;

while true

    f1 = Task3_Function2(x1,C,H,O,freq,airs,press,kco2,kh2o);
    f2 = Task3_Function2(x2,C,H,O,freq,airs,press,kco2,kh2o);
    f3 = Task3_Function2(x3,C,H,O,freq,airs,press,kco2,kh2o);

    if abs(f2) < tolerance
        break;
    end

    if abs(x3 - x1) < tolerance * max(1, abs(x2))
        break;
    end

    R = f2 / f3;
    S = f2 / f1;
    T = f1 / f3;

    P = S*(T*(R-T)*(x3-x2) - (1-R)*(x2-x1));
    Q = (T-1)*(R-1)*(S-1);

    %Checks to see if interpolation is valid
    safe = (Q ~= 0) && isfinite(P/Q);

    if safe
        x_new = x2 + P/Q;
        % Check if x_new is inside bracket
        if (x_new > x1) && (x_new < x3)

            else
                % if others are not applicable use bisection to get new x
                x_new = 0.5*(x1 + x3);
            end
        else
            % if interpolation is not applicable at all
            x_new = 0.5*(x1 + x3);
        end

        if f1 * f2 > 0
            x1 = x2;
        else
            x3 = x2;
        end

        x2 = x_new;

        iteration=iteration+1;
    end

    n5ntvalue = x2;
end

```

Figure 14 – Brent's method function 1