

DS 288 Numerical Methods

Homework 2 Solution

Q1. The results are presented below. The scripts used can be found in Appendix 1.

(a) $f(x) = x + e^{-x^2} \cos x$

Method	Root	# of iterations	Computation Time
Newton's Method	-0.5884017765	5	0.241364
Secant Method	-0.5884017765	7	0.158435
Modified Newton's Method	-0.5884017765	5	0.364328

(b) $f(x) = (x + e^{-x^2} \cos x)^2$

Method	Root	# of iterations	Computation Time
Newton's Method	-0.5884011103	18	0.767558
Secant Method	-0.5884006786	33	0.740896
Modified Newton's Method	-0.5884017765	5	1.083236

Comments on Convergence Rates

(a) The Secant Method requires more number of iterations to achieve the required accuracy, but lesser computation time. Newton's Method and Modified Newton's Method require the same number of iterations but Modified Newton's Method finishes later. Secant Method converges slower but finishes first.

(b) The distinction between the convergence rates of the three methods can be clearly seen here. Secant Method converges slowest and takes up 6 times more iterations than Modified Newton's Method and almost twice the number of iterations compared to Newton's Method. On the other hand Secant Method still finishes earlier than the other two methods.

We can conclude that Modified Newton's Method does converge better than Newton's Method, which converges better than Secant Method. This is in accordance with the analysis done theoretically in class. But in terms of real-time computation time, which is the real deciding factor for computation costs, Modified Newton's Method is the worst. Secant Method and Newton's Method perform much better.

NOTE: The methods have been implemented in MATLAB and hence are valid only for this platform. Also, the derivatives were computed implicitly by using symbolic mathematics, which can be a contributor to an increase in the computation time. Had they been explicitly provided, it is possible that Newton's Method could have finished earlier than Secant Method.

Q2. We have

$$g(x) = x - \phi(x)f(x) - \psi(x)f^2(x)$$

For $g(p_n)$ to be a cubically convergent fixed point iteration function, we must have

$$g'(p) = 0 \text{ and } g''(p) = 0 \text{ when } f(p) = 0$$

Now

$$g'(x) = 1 - \phi'(x)f(x) - \phi(x)f'(x) - \psi'(x)f^2(x) - 2\psi(x)f(x)f'(x)$$

and

$$g''(x) = -\phi''(x)f(x) - 2\phi'(x)f'(x) - \phi(x)f''(x) - \psi''(x)f^2(x) - 4\psi'(x)f(x)f'(x) - 2\psi(x)[(f'(x))^2 + f(x)f''(x)]$$

Since $f(p) = 0$, we have $g'(p) = 1 - \phi(p)f'(p)$, and $g'(p) = 0$ if and only if

$$\phi(p) = \frac{1}{f'(p)}$$

Again, since $f(p) = 0$, we have $g''(p) = -2\phi'(p)f'(p) - \phi(p)f''(p) - 2\psi(p)(f'(p))^2$, hence $g''(p) = 0$ if and only if

$$\psi(p) = \frac{-\phi'(p)}{f'(p)} - \frac{\phi(p)f''(p)}{2(f'(p))^2}$$

Now $\phi(p) = \frac{1}{f'(p)}$, hence $\phi'(p) = \frac{-f''(p)}{(f'(p))^2}$, and thus

$$\psi(p) = \frac{f''(p)}{2(f'(p))^3}$$

Thus,

$$\boxed{\phi(x) = \frac{1}{f'(x)}} \text{ and } \boxed{\psi(x) = \frac{f''(x)}{2(f'(x))^3}}$$

Convergence Analysis

We have by definition

$$p = g(p) \quad \text{and} \quad p_{n+1} = g(p_n)$$

$$\text{Then} \quad |p_{n+1} - p| = |g(p_n) - g(p)|$$

Using Taylor series expansion of $g(p_n)$ near p , we have

$$g(p_n) = g(p) + g'(p)(p_n - p) + \frac{g''(p)}{2!}(p_n - p)^2 + \frac{g'''(p)}{3!}(p_n - p)^3 + \dots$$

Since $g'(p) = 0$ and $g''(p) = 0$, it follows that

$$|p_{n+1} - p| = \left| \frac{g'''(p)}{3!}(p_n - p)^3 + \dots \right|$$

Then, we have the following limit

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^3} = \left| \frac{g'''(p)}{3!} \right|$$

We have

$$g(x) = x - \phi(x)f(x) - \psi(x)f^2(x)$$

Then

$$g'(x) = 1 - \phi'(x)f(x) - \phi(x)f'(x) - \psi'(x)f^2(x) - 2\psi(x)f(x)f'(x)$$

For simplification, we can write

$$g' = 1 - \phi'f - \phi f' - \psi'f^2 - 2\psi ff'$$

Differentiating,

$$g'' = -\phi''f - \phi'f' - \phi'f' - \phi f'' - \psi''f^2 - \psi'2ff' - 2\psi'ff' - 2\psi(ff'' + f'^2)$$

$$\text{or} \quad g'' = -\phi''f - 2\phi'f' - \phi f'' - \psi''f^2 - 4\psi'ff' - 2\psi ff'' - 2\psi f'^2$$

Differentiating again,

$$g''' = -\phi'''f - \phi''f' - 2(\phi''f' + \phi'f'') - \phi'f'' - \phi f''' - \psi'''f^2 - 2\psi''ff' - 4(\psi''ff' + \psi'f'^2 + \psi'ff'') - 2(\psi'ff'' + \psi f'f'' + \psi ff''') - 2(\psi'f'^2 + 2\psi f'f'')$$

Since $f(p)=0$, we have at $x=p$

$$g''' = -3(\phi''f' + \phi'f'') - \phi f''' - 6(\psi'f'^2 + \psi f'f'')$$

Further, since

$$\phi(x) = \frac{1}{f'(x)}$$

$$\phi' = \frac{-f''}{f'^2} \quad \text{and} \quad \phi'' = \frac{-f'''f'^2 + 2f'f''^2}{f'^4} = \frac{-f'''}{f'^2} + \frac{2f''^2}{f'^3}$$

Also,

$$\psi(x) = \frac{f''(x)}{2(f'(x))^3}$$

and
$$\psi' = \frac{2f'''f'^3 - 6f'^2f''^2}{4(f'(x))^6} = \frac{f'''}{2f'^3} - \frac{3f''^2}{2f'^4}$$

Then at $x=p$

$$g''' = -3(\phi''f' + \phi'f'') - \phi f''' - 6(\psi'f'^2 + \psi f'f'')$$

becomes

$$g''' = -3\left(\frac{-f'''}{f'^2} + \frac{2f''^2}{f'^2} - \frac{f''^2}{f'^2}\right) - \frac{f'''}{f'} - 6\left(\frac{f'''}{2f'} - \frac{3f''^2}{2f'^2} + \frac{f'f''^2}{2f'^3}\right)$$

Simplifying

$$g''' = \frac{3f''^2}{f'^2} - \frac{f'''}{f'}$$

Thus, we can re-write our earlier limit as

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^3} = \left| \frac{f''^2}{2f'^2} - \frac{f'''}{6f'} \right|$$

Therefore,

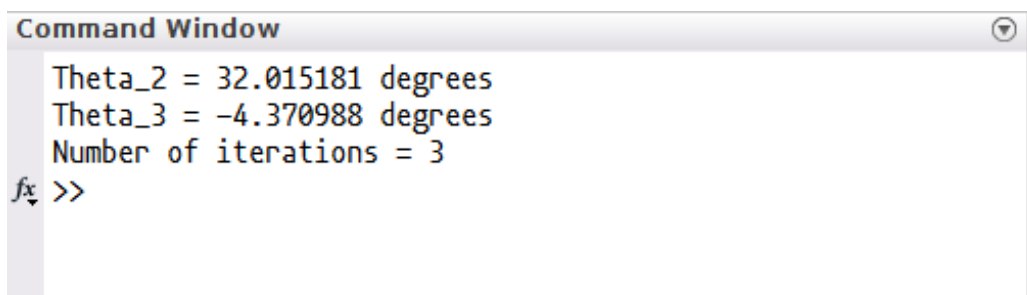
- the asymptotic order of convergence is $\boxed{\alpha=3}$
- the asymptotic error constant is $\boxed{\lambda = \left| \frac{f''^2}{2f'^2} - \frac{f'''}{6f'} \right|}$

Q3. As per the output of MATLAB script used to solve the system of non-linear equations in

θ_2 and θ_3 ,

- No. of iterations required to achieve a tolerance of $10^{-4} = 3$
- $\theta_2 = 32.015181$ degrees
- $\theta_3 = -4.370988$ degrees (i.e. clockwise angle)

The command window output is shown below.

A screenshot of the MATLAB Command Window. The window has a title bar that says "Command Window" with a close button on the right. The text inside the window is as follows:

```
Theta_2 = 32.015181 degrees  
Theta_3 = -4.370988 degrees  
Number of iterations = 3  
fx >>
```

The MATLAB script used to compute the above follows.

NOTE: Angles were computed in radians and converted back to degrees when the desired accuracy had been achieved.

Script – hw2q3.m

```
clc; clear;

% known lengths of links
r1 = 10;
r2 = 6;
r3 = 8;
r4 = 4;

% known angles (rad)
theta4 = 220*pi/180;

% initial guess for theta2 and theta3
init_theta2 = pi/6;
init_theta3 = 0;

% tolerance
tol = 1e-4;

% iteration counter
iter = 1;

while (1 > 0)
    f = -[r2*cos(init_theta2) + r3*cos(init_theta3) + r4*cos(theta4) - r1; ...
          r2*sin(init_theta2) + r3*sin(init_theta3) + r4*sin(theta4)];
    % compute Jacobian
    J = [-r2*sin(init_theta2) -r3*sin(init_theta3); ...
          r2*cos(init_theta2)  r3*cos(init_theta3)];
    % solve for new theta2 and theta3
    guess = J\f;
    theta2 = init_theta2 + guess(1);
    theta3 = init_theta3 + guess(2);
    % check if guess is as accurate as required
    if (abs(guess(1)) < tol) && (abs(guess(2)) < tol)
        break
    else
        iter = iter + 1;      % update iteration counter
        init_theta2 = theta2; % reset initial guess for theta2
        init_theta3 = theta3; % reset initial guess for theta3
    end
end

fprintf('Theta_2 = %.6f degrees\n', theta2*180/pi)
fprintf('Theta_3 = %.6f degrees\n', theta3*180/pi)
fprintf('Number of iterations = %d\n', iter)

% uncomment to check
%[r2*cos(theta2) + r3*cos(theta3) + r4*cos(theta4) - r1; ...
%r2*sin(theta2) + r3*sin(theta3) + r4*sin(theta4)]
```

Appendix 1 – MATLAB Code and CW Outputs for Q1Script 1 – hw2q1a.m (for Q1a)

```
clc; clear;

syms x

% function whose root is to be found
f = x + exp(-x^2)*cos(x);

%% Newton's Method
tic
% initial guess and tolerance
init_guess = 0;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;

% counter for number of iterations
iter = 1;

guess = init_guess;

while (1 > 0)
    a = double(subs(f, init_guess)); % f(init_guess)
    df = diff(f, x); % f'(x)
    b = double(subs(df, init_guess)); % f'(init_guess)
    guess = init_guess - a/b; % new guess for root

    if abs(guess - init_guess) < tol
        break % exit if root found
    else
        iter = iter + 1; % update iteration counter
        init_guess = guess; % set obtained root to initial guess
    end
end

% output obtained root
fprintf('NEWTON'S METHOD\nRoot = %.10f.\nNo. of iterations required = %d.\n',
guess, iter)
toc
fprintf('\n')
%% Secant Method
tic
% initial interval and tolerance
init_left = 0;
init_right = 1;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;
```



```

% counter for number of iterations
iter = 1;

while (1 > 0)
    a = double(subs(f, init_left)); % f(init_left)
    b = double(subs(f, init_right)); % f(init_right)
    % new guess for root
    guess = init_right - b*(init_right - init_left)/(b - a);

    if abs(guess - init_right) < tol % exit if root found
        break
    else
        iter = iter + 1; % update iteration counter
        init_left = init_right;
        init_right = guess;
    end
end

% output obtained root
fprintf('SECANT METHOD\nRoot = %.10f.\nNo. of iterations required = %d.\n',
guess, iter)
toc
fprintf('\n')
%% Modified Newton's Method
tic
% initial guess and tolerance
init_guess = 0;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;

% counter for number of iterations
iter = 1;

guess = init_guess;

while (1 > 0)
    a = double(subs(f, init_guess)); % f(init_guess)
    df = diff(f, x); % f'(x)
    b = double(subs(df, init_guess)); % f'(init_guess)
    ddf = diff(df, x); % f''(x)
    c = double(subs(ddf, init_guess)); % f''(init_guess)
    guess = init_guess - (a*b)/(b^2 - a*c); % new guess for root

    if abs(guess - init_guess) < tol
        break % exit if root found
    else
        iter = iter + 1; % update iteration counter
        init_guess = guess; % set obtained root to initial guess
    end
end

```

```
end
```

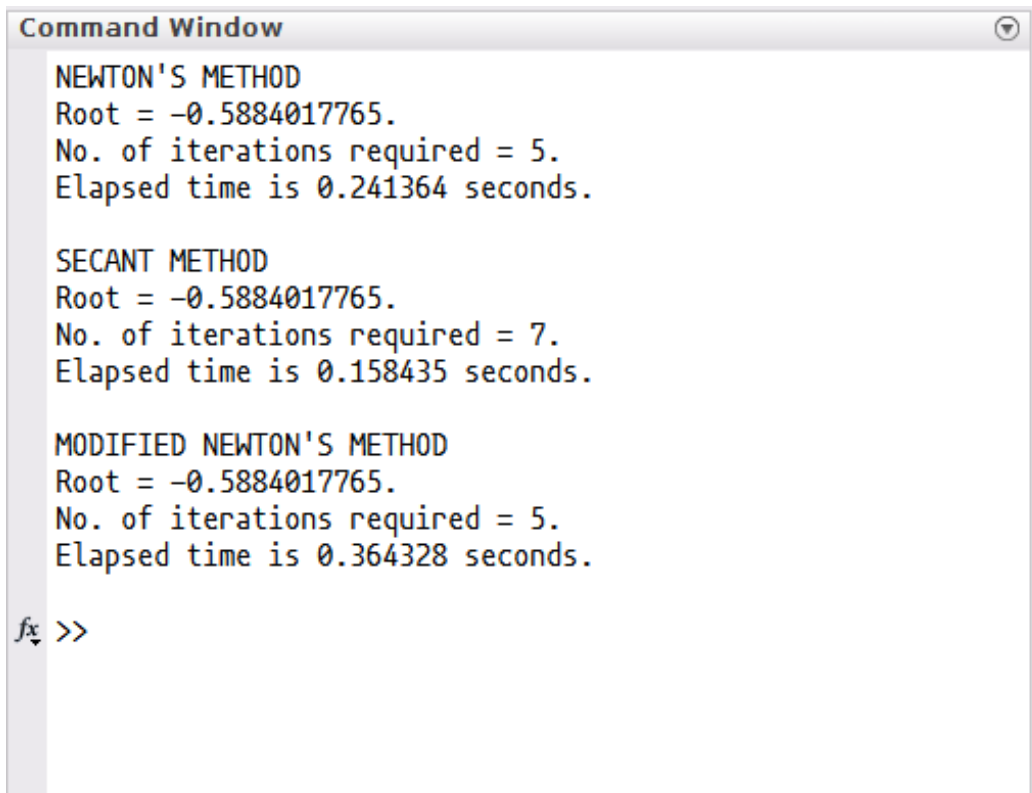
```
% output obtained root
```

```
fprintf('MODIFIED NEWTON'S METHOD\nRoot = %.10f.\nNo. of iterations required =  
%d.\n', guess, iter)
```

```
toc
```

```
fprintf('\n')
```

Command Window Output



```
Command Window

NEWTON'S METHOD
Root = -0.5884017765.
No. of iterations required = 5.
Elapsed time is 0.241364 seconds.

SECANT METHOD
Root = -0.5884017765.
No. of iterations required = 7.
Elapsed time is 0.158435 seconds.

MODIFIED NEWTON'S METHOD
Root = -0.5884017765.
No. of iterations required = 5.
Elapsed time is 0.364328 seconds.

fx >>
```

Script 2 – hw2q1b.m (for Q1b)

```
clc; clear;

syms x

% function whose root is to be found
f = (x + exp(-x^2)*cos(x))^2;

%% Newton's Method
tic
% initial guess and tolerance
init_guess = 0;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;

% counter for number of iterations
iter = 1;

guess = init_guess;

while (1 > 0)
    a = double(subs(f, init_guess)); % f(init_guess)
    df = diff(f, x); % f'(x)
    b = double(subs(df, init_guess)); % f'(init_guess)
    guess = init_guess - a/b; % new guess for root

    if abs(guess - init_guess) < tol
        break % exit if root found
    else
        iter = iter + 1; % update iteration counter
        init_guess = guess; % set obtained root to initial guess
    end
end

% output obtained root
fprintf('NEWTON'S METHOD\nRoot = %.10f.\nNo. of iterations required = %d.\n',
guess, iter)
toc
fprintf('\n')
%% Secant Method
tic
% initial interval and tolerance
init_left = 0;
init_right = 1;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;
```

```

% counter for number of iterations
iter = 1;

while (1 > 0)
    a = double(subs(f, init_left)); % f(init_left)
    b = double(subs(f, init_right)); % f(init_right)
    % new guess for root
    guess = init_right - b*(init_right - init_left)/(b - a);

    if abs(guess - init_right) < tol % exit if root found
        break
    else
        iter = iter + 1; % update iteration counter
        init_left = init_right;
        init_right = guess;
    end
end

% output obtained root
fprintf('SECANT METHOD\nRoot = %.10f.\nNo. of iterations required = %d.\n',
guess, iter)
toc
fprintf('\n')
%% Modified Newton's Method

% initial guess and tolerance
init_guess = 0;
tol = 1e-6;

% use in case looping happens
%maxiter = 100;

% counter for number of iterations
iter = 1;

guess = init_guess;

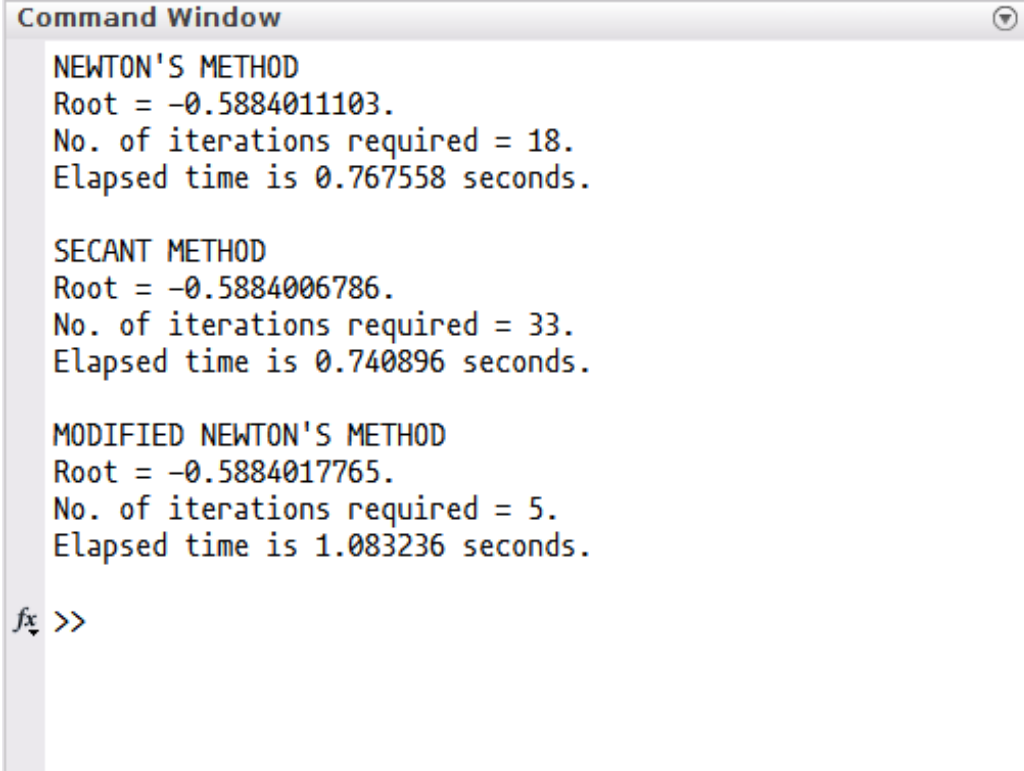
while (1 > 0)
    a = double(subs(f, init_guess)); % f(init_guess)
    df = diff(f, x); % f'(x)
    b = double(subs(df, init_guess)); % f'(init_guess)
    ddf = diff(df, x); % f''(x)
    c = double(subs(ddf, init_guess)); % f''(init_guess)
    guess = init_guess - (a*b)/(b^2 - a*c); % new guess for root

    if abs(guess - init_guess) < tol
        break % exit if root found
    else
        iter = iter + 1; % update iteration counter
        init_guess = guess; % set obtained root to initial guess
    end
end

```

```
% output obtained root  
fprintf('MODIFIED NEWTON'S METHOD\nRoot = %.10f.\nNo. of iterations required =  
%d.\n', guess, iter)  
toc  
fprintf('\n')
```

Command Window Output



```
Command Window  
  
NEWTON'S METHOD  
Root = -0.5884011103.  
No. of iterations required = 18.  
Elapsed time is 0.767558 seconds.  
  
SECANT METHOD  
Root = -0.5884006786.  
No. of iterations required = 33.  
Elapsed time is 0.740896 seconds.  
  
MODIFIED NEWTON'S METHOD  
Root = -0.5884017765.  
No. of iterations required = 5.  
Elapsed time is 1.083236 seconds.  
  
fx >>
```