MATH 446/OR 481
Project 3

September 15, 2022

Project Summary

Newton's Method works well and converges quadratically for functions with roots whose multiplicity equals 1. For roots with multiplicity > 1, the method converges linearly.

Q1

The functions given are

$$f(x) = (400x^6 + (5x^2 - 1)^4 + 60x^2 + 8)\exp(1) - 24\exp(5x^2) \quad (1)$$
$$f(x) = (500x^6 + (5x^2 - 1)^4 + 60x^2 + 8)\exp(1) - 24\exp(5x^2) \quad (2)$$
$$f(x) = (600x^6 + (5x^2 - 1)^4 + 60x^2 + 8)\exp(1) - 24\exp(5x^2) \quad (3)$$

Using Newton's Method, the roots and corresponding number of iterations required are shown in the table below. A tolerance level of 10^{-9} is used.

version	root	iterations
1	0.2528577	8
2	0.4475174	32
3	0.8432553	64

Q2

Using MATLAB, we see that Newton's method converges quadratically for (1) and (3), whereas it exhibits linear convergence for (2). The results are tabulated below.

version	$f'(r)$	ratio
1	-2.9786	3.3334
2	-4.1553×10^{-11}	0.7928
3	-1.1062×10^{3}	11.7954

From the ratio computed for function (2), the multiplicity of the root can be estimated to be

$$m = \frac{1}{1 - 0.7928} = 4.83 \implies \boxed{m = 5}$$

The theoretical values will be computed now.

We know that Newton's method exhibits quadratic convergence when $f'(r) \neq 0$ and we have

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left|\frac{f''(r)}{2f'(r)}\right|$$

If $f'(r) = 0$, the method is linearly convergent with

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = \frac{m-1}{m}$$

where m is the multiplicity of the root.

The values of first and second derivatives for the functions are tabulated below. We use the obtained roots as the true roots for computations. For (2), the true root is $\frac{1}{\sqrt{5}}$ which is used to evaluate the derivatives.

version	root	$f'(r)$	$f''(r)$
1	0.2528577	-2.9786	-19.828
2	$\frac{1}{\sqrt{5}} = 0.4471236$	0	0
3	0.8432553	-1106.2	-26106.8

For function (1),

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left|\frac{f''(r)}{2f'(r)}\right| = \frac{19.828}{2(2.9786)} = \boxed{3.3284}$$

For function (3),

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left|\frac{f''(r)}{2f'(r)}\right| = \frac{26106.8}{2(1106.2)} = \boxed{11.8002}$$

For function (2), we observe that $f'(r) = f''(r) = 0$. Moving on to higher derivatives, we see that $f'''(r) = f^{(iv)}(r) = 0$. But $f^{(v)}(r) = -116702.6 \neq 0$. Thus, the multiplicity of the root is 5, i.e. $\boxed{\text{m} = 5}$ and hence,

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i} = \frac{m-1}{m} = \boxed{0.8}$$

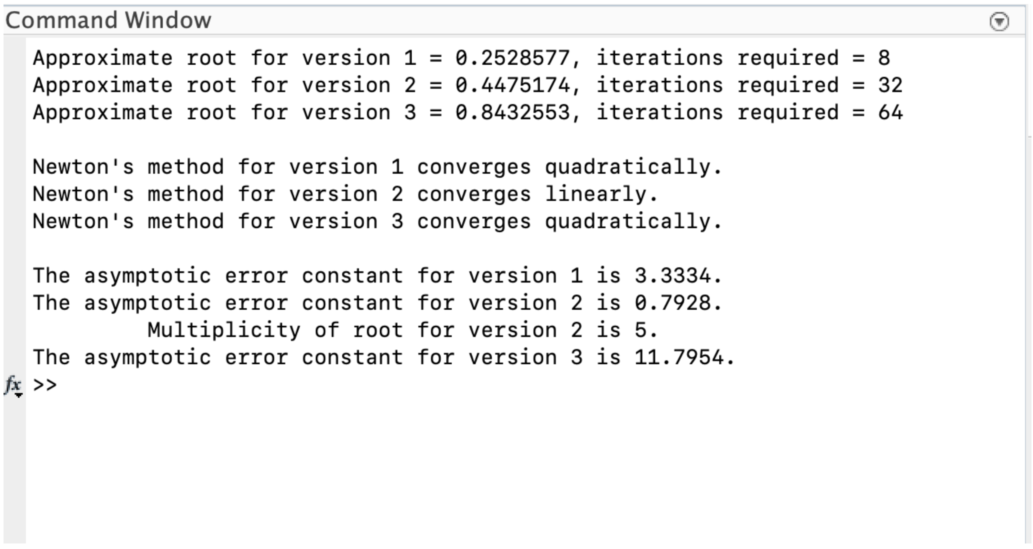
A comparison of theoretical and computed error constants is shown below.

version	theoretical	computed
1	3.3284	3.3334
2	0.8	0.7928
3	11.8002	11.7954

Q3

We can see from our results that the computed ratios agree fairly well with the expected theoretical values. We also see the worse performance of Netwon's method when applied to a function with a root of high multiplicity like function (2). Errors don't scale down fast enough, and the convergence is slow. The method can also perform worse when the root has low multiplicity but the error constant is high. The function (3) requires the highest number of iterations, even when $f'(r) \neq 0$.

Appendix - Command Window Output



```
Approximate root for version 1 = 0.2528577, iterations required = 8
Approximate root for version 2 = 0.4475174, iterations required = 32
Approximate root for version 3 = 0.8432553, iterations required = 64

Newton's method for version 1 converges quadratically.
Newton's method for version 2 converges linearly.
Newton's method for version 3 converges quadratically.

The asymptotic error constant for version 1 is 3.3334.
The asymptotic error constant for version 2 is 0.7928.
         Multiplicity of root for version 2 is 5.
The asymptotic error constant for version 3 is 11.7954.
>>
```

Appendix - MATLAB Code

```
clc, clear, close all

% define functions
f1 = @(x) (400*x^6 + (5*x^2 - 1)^4 + 60*x^2 + 8)*exp(1) - 24*exp(5*x^2);
f2 = @(x) (500*x^6 + (5*x^2 - 1)^4 + 60*x^2 + 8)*exp(1) - 24*exp(5*x^2);
f3 = @(x) (600*x^6 + (5*x^2 - 1)^4 + 60*x^2 + 8)*exp(1) - 24*exp(5*x^2);

% define derivatives
df1 = @(x) (6*400*x^5 + (10*x)*4*(5*x^2 - 1)^3 + 2*60*x)*exp(1) ...
- 24*exp(5*x^2)*10*x;
df2 = @(x) (6*500*x^5 + (10*x)*4*(5*x^2 - 1)^3 + 2*60*x)*exp(1) ...
- 24*exp(5*x^2)*10*x;
df3 = @(x) (6*600*x^5 + (10*x)*4*(5*x^2 - 1)^3 + 2*60*x)*exp(1) ...
- 24*exp(5*x^2)*10*x;


%% 1. find root using Newton's Method

initialGuess = (0+1)/2; % initial guess
tolerance = 1e-9;       % convergence threshold

oldGuess = initialGuess;
newGuess = 0;
niter = 0;

approxRoots = zeros(1, 3); % final approximate roots
maxiters = zeros(1, 3);    % interations required for convergence

% iterate over all 3 versions
for i = 1:3
% choose function and derivative
if i == 1
f = f1; df = df1;
else
if i == 2
f = f2; df = df2;
else
f = f3; df = df3;
end
```

```
end

oldGuess = initialGuess;
while(true)
% update iteration counter
niter = niter + 1;

% newton iteration
newGuess = oldGuess - f(oldGuess)/df(oldGuess);

% check for convergence
if (abs(newGuess - oldGuess)) < tolerance
break
else
oldGuess = newGuess;
end
end

approxRoots(i) = newGuess;
maxiters(i) = niter;

fprintf("Approximate root for version %d = %.7f, " + ...
"iterations required = %d\n", i, approxRoots(i), niter);
end


%% 2. find order of convergence and multiplicity of roots

fprintf('\n')

% find order of convergence
convergence_order = zeros(1, 3);

for i = 1:3
% choose derivative
if i == 1
df = df1;
else
if i == 2
df = df2;
else
```

```
df = df3;
end
end

% check if f'(r) not 0
if (abs(df(approxRoots(i)))) > 1e-6
convergence_order(i) = 2;
fprintf('Newton''s method for version %d converges quadratically.\n', i)
else
convergence_order(i) = 1;
fprintf('Newton''s method for version %d converges linearly.\n', i)
end
end

% find asymttotic error constant
fprintf('\n');
for i = 1:3
root = approxRoots(i);
if (i == 2)
root = 1/sqrt(5);
end
tolerance = 1e-6;

errors = zeros(1, maxiters(i));
niter = 0;

% choose function and derivative
if i == 1
f = f1; df = df1;
else
if i == 2
f = f2; df = df2;
else
f = f3; df = df3;
end
end

oldGuess = initialGuess;

while(true)
niter = niter + 1;
```

```
% newton iteration
newGuess = oldGuess - f(oldGuess)/df(oldGuess);

% compute error
errors(niter) = abs(newGuess - root);

% check for convergence
if (abs(newGuess - oldGuess)) < tolerance
break
else
oldGuess = newGuess;
end
end

% computer asymptotic error constant (lim i -> Inf)
if (i ~= 2)     % quadratic convergence
errorConstant = errors(niter - 1)/errors(niter - 2)^2;
else            % linear convergence
errorConstant = errors(niter - 3)/errors(niter - 4 );
end
fprintf('The asymptotic error constant for version %d is %.4f.\n', ...
i, errorConstant);


% for (B), compute root multiplicity
if (i == 2)
m = 1/(1 - errorConstant);
fprintf('\t Multiplicity of root for version 2 is %d.\n', round(m))
end
end
```