Text

Description automatically generated

**(a)** A fixed point of a function is an element that is mapped to itself by the function. That is, c is a fixed point of a function f if c belongs to both the domain and the codomain of f, and f(c) = c  
=> . Replacing  into the function, we have:

=> is a fixed point for 

Solving the equation  to find all fixed points of the function . Condition: .   
=> => Another fixed point is   
**(b)** The Matlab code for squareroot3.m is  
%{

This function uses recursion to implemen the iteration method

We need to find the square root of 3. In other words, we should find the

root of the function x^2 = 3.

We should convert this function to the form: x = f(x) so that

it will become an iteration method

=> x = 1/2 \* (x + 3/x)

In the function, the output is a matrix of errors for n = 0,1,2,3,...

%}

function [errors] = squareroot3(x,n)

format short e

errors = inner([],x,n);

end

function [errorsLoop] = inner(errorsInput,x,n)

if n ~= 0

x\_nextiteration = 1/2 \* (x + 3/x);

error = [errorsInput, abs(x - sqrt(3))];

errorsLoop = inner(error, x\_nextiteration, n - 1);

else

errorsLoop = [errorsInput, abs(x - sqrt(3))];

end

end

The Matlab code for squareroot3\_order.m is:

clc;

% I choose n = 5 to obtain a correct rate of convergence

% The rate of converge to sqrt(3) for x = 1 to 5 are

for x = 1:5

errors = squareroot3(x,5);

disp("The errors for x = " + x)

disp(errors);

disp(" ");

disp("Rate of convergence for " + x + ": " + detectrate(errors));

end

Text

Description automatically generated

=> The rate of convergence is around=> Order of the method is 2

**(c)** Proof





As we can see, big number converges by being halved in each iteration => All x converges to smaller value when x is large. When x is near sqrt(3), it converges to sqrt(3). This can be seen in the limits  
=> The fixed point methos using  converges to for any 

**(d)** Proof

First iteration: 

Second iteration:   
We can see that as x approaches 0, the result will reach positive infinity in the second iteration. And we know that large values will converge to sqrt(3) again as proved in (c) from second iteration onwards.

is strictly positive, so we can see that after each iteration, values of x strictly increases. If x exceeds sqrt(3), it will starts to converge down as seen from (c).   
=> For all x > 0, the fixed point method converges to sqrt(3)

Text

Description automatically generated

We have:

=> LHS = RHS (proven)

Order of the method  
Suppose  is a sequence that converges to x\*, with  for all n. If positive constants and  exist with



Then  converges to p of order , with asymptotic error constant 

By definition, we know that , since the iteration method is convergent to sqrt(3) as found in (d) for all x > 0

=>   
We need to find such that the result of the limit is a constant.   
Observation: which is constant => We can make the termvanished => 

=> =. The order of the method is thus 2

Text

Description automatically generated

The Newton’s Method is given by: . Now we applied Newton’s method to the function 



=> The fixed point method is truly the Newton’s method in disguise.

Chart

Description automatically generated

Text, letter

Description automatically generated

a) The estimation for the accuracy of the difference approximationis given by the floating-point number:.

We have: fl(y) = fl(1) = 1, fl(2^-n) = 2^-n, since they are integers and can be expressed exactly in floating-point. The formula can be simplified as:



The rounding formula for fl(x), x = e^(2^-n) is given by



- For n = 25, we have:



We know that for n >= 53, 2^-n in floating-point cannot be recognized by computers and thus, it equals to 0.

=> 

=> 

=> Difference approximation for n = 25:



Similarly for n = 26:



=> 

=> 



n = 27:



=> 

=> 



n = 53:

=>  => 



=> For n = 25, 

=> For n > 25 and n < 53, 

=> For n >= 53, 

b) Accuracy of series expansion

\*For: Done in part A

\* For 

- For n = 25, => 



- For n > 26 and n < 52,   
- For >= 52, 

\* For 

- For n = 25, 

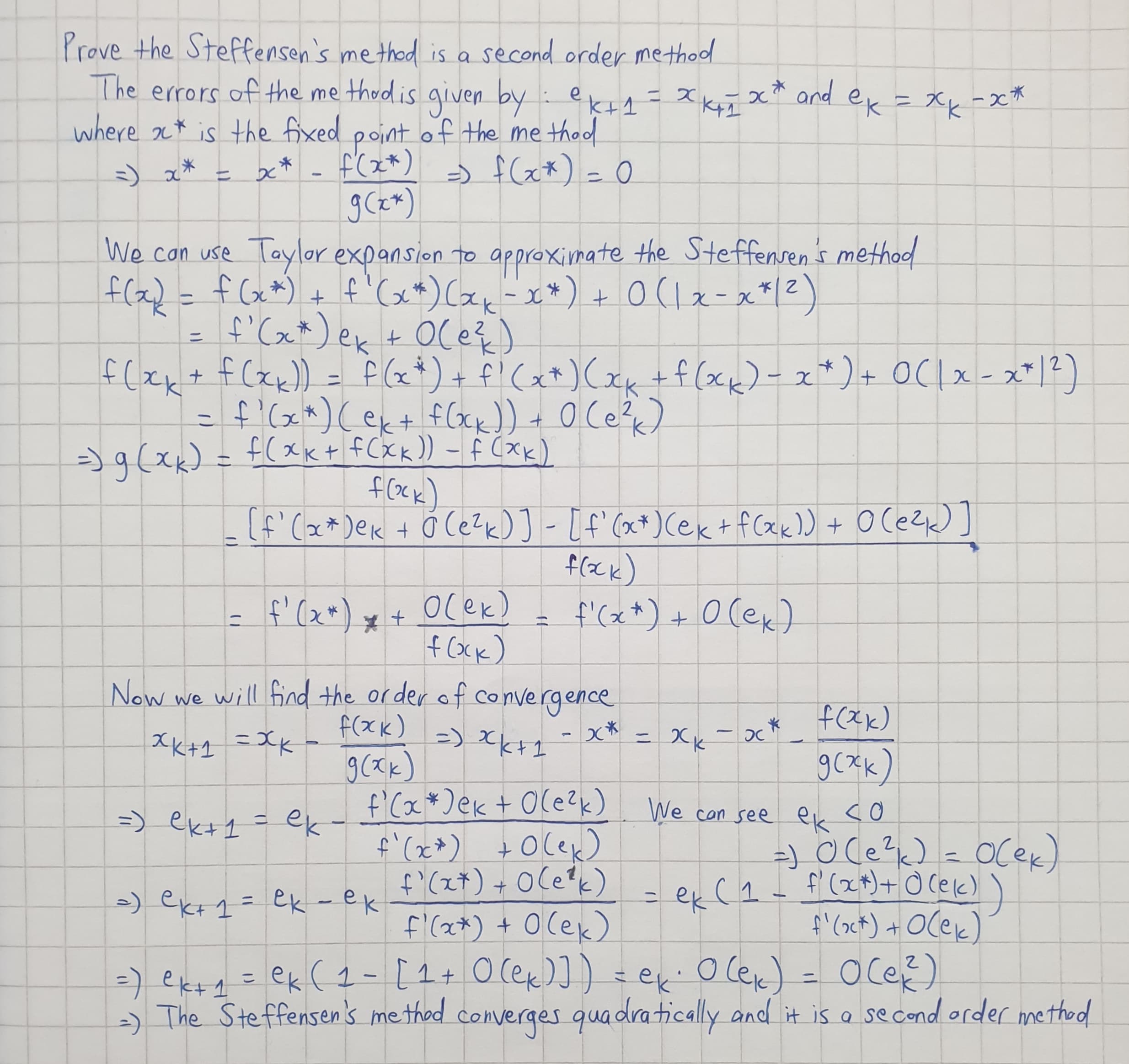
- For n > 26 and n < 52, 

- For >= 52, 

=> As n increases, the accuracy gets close to 1 => The series expansion becomes more accurate

Text, letter

Description automatically generated



Text, letter

Description automatically generated

The Matlab code for arctanappr.m

function val = arctanappr(x)

if x >= 0 && x <= 1.7e-9

val = x;

elseif x > 1.7e-9 && x <= 2e-2

val = x - (x^3)/3 + (x^5)/5 - (x^7)/7;

elseif x >= 0 && x <= 1

y = x;

a = 0;

b = 1;

if y >= 0 && y <= sqrt(2) - 1

c = pi/16;

d = tan(c);

elseif y > sqrt(2) - 1 && y <= 1

c = 3\*pi/16;

d = tan(c);

end

u = (y - d)/(1 + d \* y);

arctan\_u = u \* ((135135 + 171962.46\*u^2 + 52490.4832\*u^4 + 2218.1\*u^6)/(135135 + 217007.46\*u^2 + 97799.3033\*u^4 + 10721.3745\*u^6));

val = a + b\*(c + arctan\_u);

elseif x > 1

y = 1/x;

a = pi/2;

b = -1;

if y >= 0 && y <= sqrt(2) - 1

c = pi/16;

d = tan(c);

elseif y > sqrt(2) - 1 && y <= 1

c = 3\*pi/16;

d = tan(c);

end

u = (y - d)/(1 + d \* y);

arctan\_u = u \* ((135135 + 171962.46\*u^2 + 52490.4832\*u^4 + 2218.1\*u^6)/(135135 + 217007.46\*u^2 + 97799.3033\*u^4 + 10721.3745\*u^6));

val = a + b\*(c + arctan\_u);

end

end

The testing Matlab file for arctanappr\_test.m

%{

If x is the exact value, x0 is the approximation value

=> Absolute error: Abs\_e = x - x0

=> Relative error: Rel\_e = Abs\_e/x.

%}

clc;

t=linspace(0,2\*pi,10000);

rt = atan(t);

for i=1:length(t)

at(i)=arctanappr(t(i));

end

figure(1);

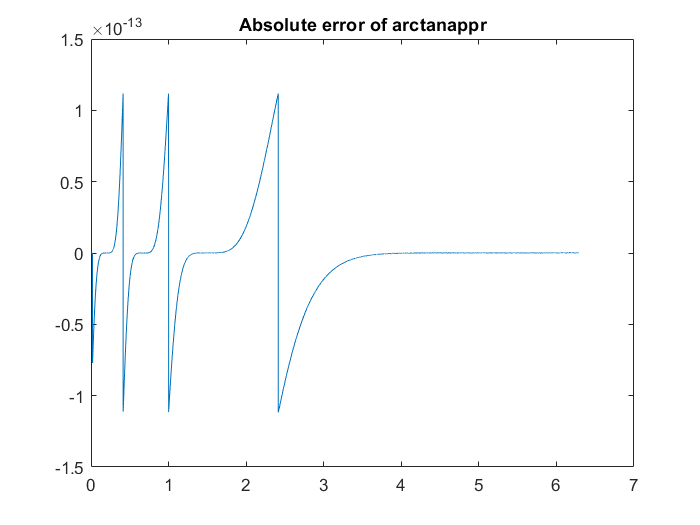
plot(t,rt-at)

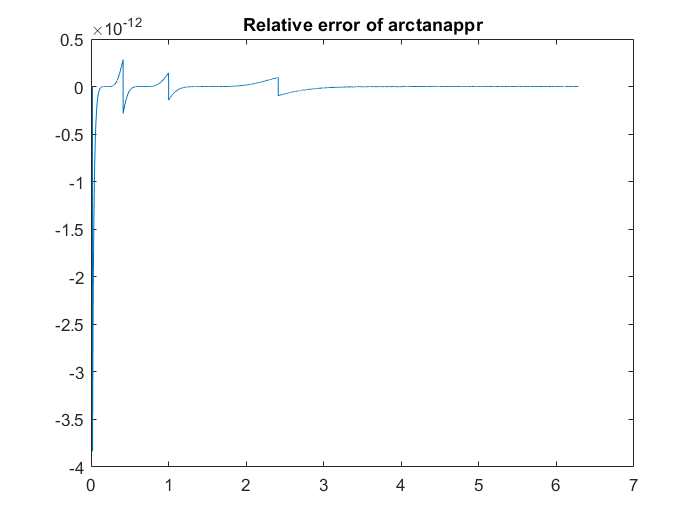
title("Absolute error of arctanappr")

figure(2);

plot(t,(rt-at)./rt)

title("Relative error of arctanappr")





We can see that the absolute error of the arctan approximation method is prediodic in 3 cycles for the first pi. After the first pi, the error stabilizes and its very close to 0. The relative error also shows the same characteristic as the absolute error. Since the magnitudes of the erros are extremely small (10 to the power of 13), this approximation is a useful implementation