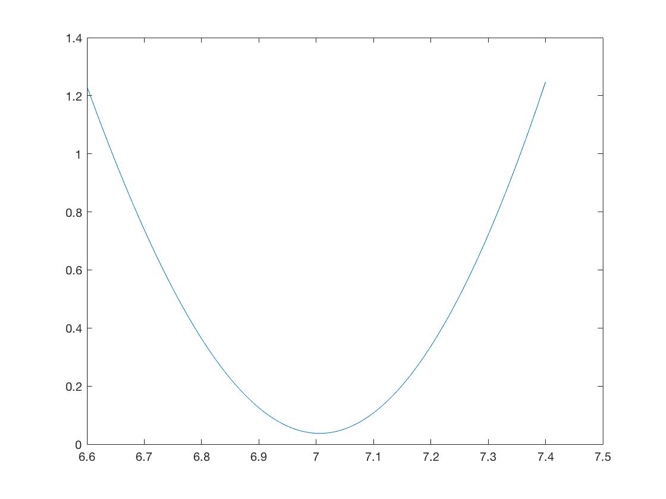
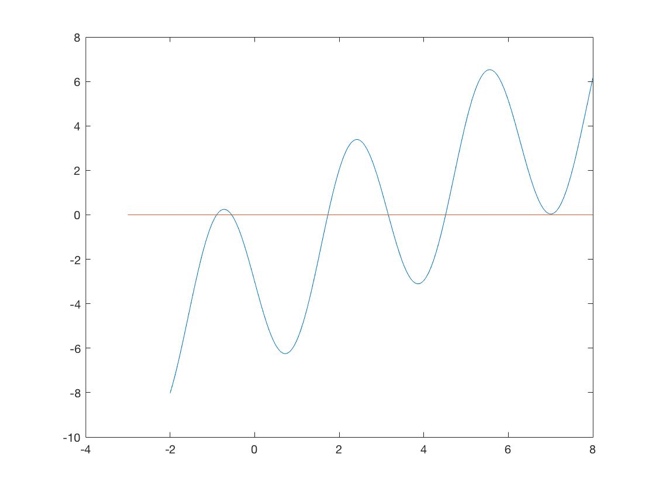
Nummen Lab 2

# Uppgift 1:

Bestäm samtliga rötter till med minst 10 siffrors noggrannhet

## Grafen för

Grafen har 5 rötter, se plottar:



Kod för graferna (ändra på x-vektorn för att växla mellan graferna):

x = -3:.0001:8;

y = x - 4\*sin(2\*x) - 3;

z = -3:8;

l = 0 \* z;

figure(1)

plot(x, y)

hold on

plot(z, l)

## Beräkna rötterna med fixpunktsmetoden för med 10 siffrors noggrannhet.

Detta beräknades med följande MATLAB kod:

Startvärdet x = 1 ger

x2 = -0.5444424006

efter 20 iterationer

Startvärdet x = 3 ger

x4 = 3.1618264866

efter 76 iterationer

x = 3;

x\_i = -sin(2\*x) + (5/4) \* x - 3/4;

counter = 1;

while abs(x - x\_i) > 10^(-10)

x = x\_i;

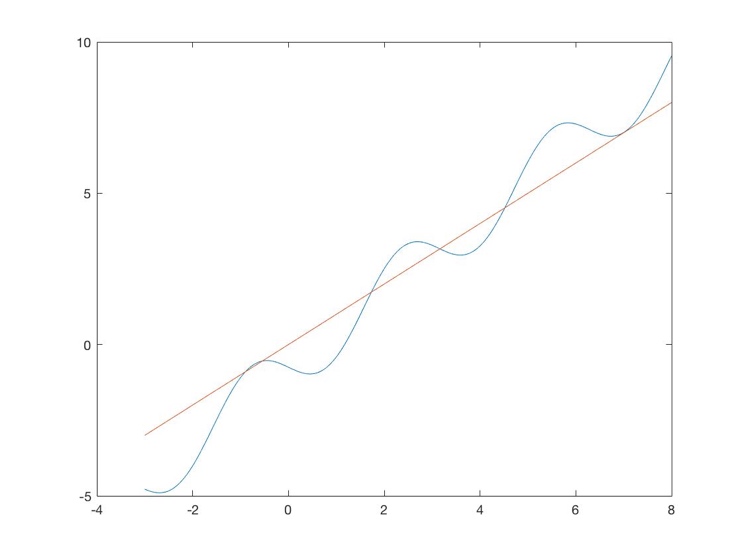
x\_i = -sin(2\*x) + (5/4) \* x - 3/4;

counter = counter + 1;

end

sprintf('%.10f' , x)

Anledningen till att vi bara fann en approximativ lösning till dessa två rötter är för att de är de ända vars absolut belopp av derivatan i den roten är mindre än ett. Detta kan även illustreras med Cobweb diagram i grafen för derivatan av . Se graf:



## Beräkna rötterna med Newtons metod med 10 siffrors noggrannhet. Kolla om antalet korrekta värdesiffror fördubblas för varje iteration.

Detta beräknades med följande MATLAB kod:

x = -1;

Startvärdet x = -1 ger

x1 = -0.8983565815

efter 6 iterationer

Startvärdet x = 2 ger

x3 = 1.7320695021

efter 6 iterationer

Startvärdet x = 5 ger

x5 = 4.5177895142

efter 6 iterationer

f = x - 4\*sin(2\*x) - 3;

fp = 1 - 8\*cos(2\*x);

x\_i = x - f/fp;

counter = 1;

while abs(x - x\_i) > 10^(-80)

x = x\_i;

f = x - 4\*sin(2\*x) - 3;

fp = 1 - 8\*cos(2\*x);

x\_i = x - f/fp;

counter = counter + 1;

end

sprintf('%.10f' , x)

Det stämmer ganska bra att värdet förbättras med 2 värdesiffror per iteration. Det krävdes 6 iterationer för att förbättra resultatet med 10 värdesiffror.

## Plotta felet efter iteration som funktion av för båda metoderna i samma figur. Plotta även som en funktion av en i loglog för de båda metoderna och avgör baserat på detta metodernas konvergensordning.

Vi valde att undersöka felet i x4 , och gjorde detta med koden:

x\_true = 3.161826486551946;

x\_newton = 2.8;

f = @(x\_newton) x\_newton - 4\*sin(2\*x\_newton) - 3;

fp = @(x\_newton) 1 - 8\*cos(2\*x\_newton);

x\_newton\_i = x\_newton - f(x\_newton)/fp(x\_newton);

x\_fix = 2.8;

x\_fix\_i = @(x\_fix) -sin(2\*x\_fix) + (5/4) \* x\_fix - 3/4;

error\_newton\_vector = zeros(1, 100);

error\_fix\_vector = zeros(1, 100);

for n = 1:100

error\_newton\_vector(n) = abs(x\_true - x\_newton\_i);

x\_newton = x\_newton\_i;

f(x\_newton);

fp(x\_newton);

x\_newton\_i = x\_newton - f(x\_newton)/fp(x\_newton);

error\_fix\_vector(n) = abs(x\_true - x\_fix\_i(x\_fix));

x\_fix = x\_fix\_i(x\_fix);

end

x = 1:100;

figure(1)

semilogy(x, error\_newton\_vector)

hold on

semilogy(x, error\_fix\_vector)

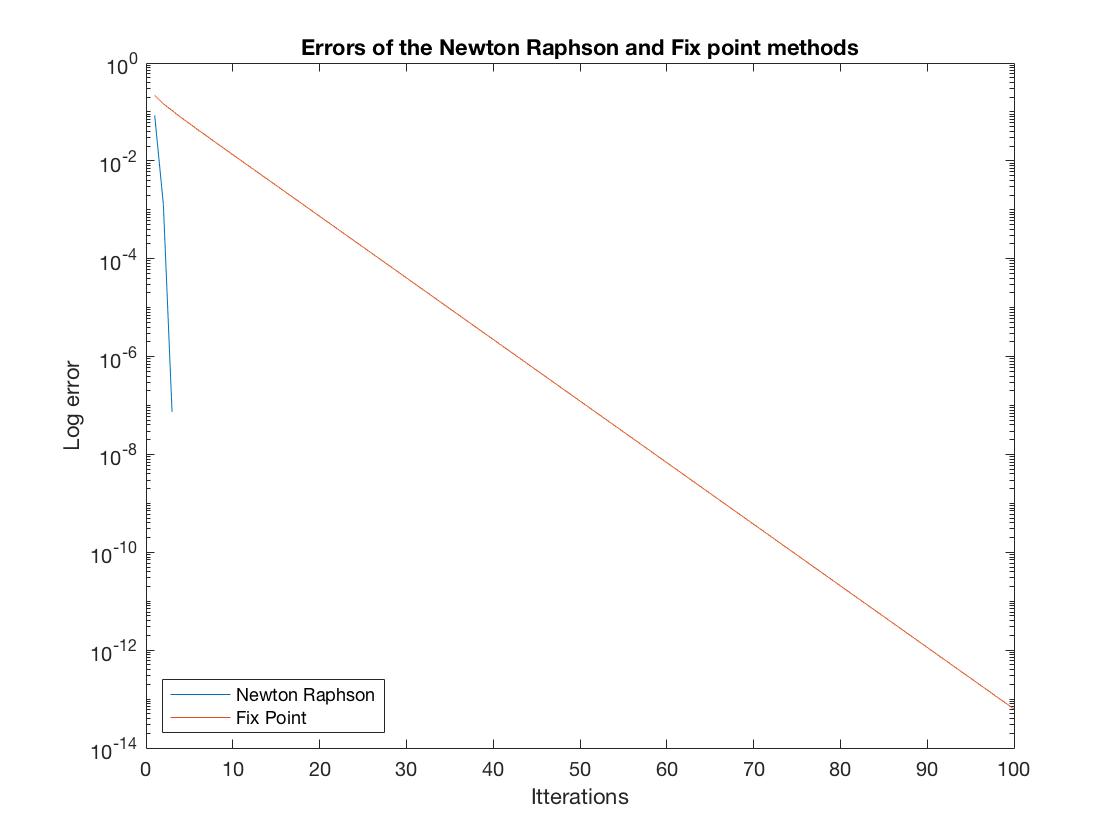
title('Errors of the Newton Raphson and Fix point methods')

xlabel('Itterations')

ylabel('Log error')

legend({'Newton Raphson','Fix Point'},'Location','southwest')

Vilket gav plotten:

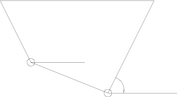


# Uppgift 2: Ett olinjärt ekvationssystem: Vikter på en lina

a

**B**

**A**



L2

L1

L

u3

u2

u1

m1

m2

u1, u2 och u3 är de vinklar som ska beräknas i ekvationssystemet:

Där och , och .

Ekvationssystemet ska lösas fyra gånger med följande olika värden på , och :

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 1 | 1 |
| 1 | 10 | 1 |

## Koden

**Funktioner:**

function J = jac\_matrix(u\_vector, constant\_vector)

J = [-sin(u\_vector(1)) -sin(u\_vector(2)) -constant\_vector(1)\*sin(u\_vector(3)); cos(u\_vector(1)) cos(u\_vector(2)) constant\_vector(1)\*cos(u\_vector(3)); constant\_vector(3)/cos(u\_vector(1))^2 -(constant\_vector(2) + constant\_vector(3))/cos(u\_vector(2))^2 constant\_vector(2)/cos(u\_vector(3))^2];

end

function vector = function\_vector\_matrix(u\_vector, constant\_vector)

f = cos(u\_vector(1)) + cos(u\_vector(2)) + constant\_vector(1) \* cos(u\_vector(3)) - 2;

g = sin(u\_vector(1)) + sin(u\_vector(2)) + constant\_vector(1) \* sin(u\_vector(3));

h = constant\_vector(3)\*tan(u\_vector(1)) - (constant\_vector(2) + constant\_vector(3))\*tan(u\_vector(2)) + constant\_vector(2)\*tan(u\_vector(3));

vector = [f g h]';

end

**Main-koden:**

% Initial guesses

u\_initial\_matrix = [1 1 pi/3 1; 0.5 0.5 pi/3-0.01 0.5; -1 -1 -pi/3 -1];

constants = [1 1 2 1; 1 1 1 10; 1 2 1 1];

for i = 1:4

u\_vector = u\_initial\_matrix(:,i);

constant\_vector = constants(:,i);

% The Jacobian Matrix

J = jac\_matrix(u\_vector, constant\_vector);

% Vector with the results from the three functions

func\_vector = function\_vector\_matrix(u\_vector, constant\_vector);

% J\func\_vector is the same as inv\_jacobian \* function\_vector

u\_next\_vector = u\_vector - J\func\_vector;

while abs(sum(u\_vector - u\_next\_vector)) > 10^(-3)

u\_vector = u\_next\_vector;

J = jac\_matrix(u\_vector, constant\_vector);

func\_vector = function\_vector\_matrix(u\_vector, constant\_vector);

u\_next\_vector = u\_vector - J\func\_vector;

end

u\_vector = u\_next\_vector;

u\_1 = u\_vector(1);

u\_2 = u\_vector(2);

u\_3 = u\_vector(3);

Ax = 0;

Ay = 0;

Bx = 2;

By = 0;

m1x = cos(abs(u\_1));

m1y = -sin(abs(u\_1));

m2x = Bx - constant\_vector(1)\*cos(abs(u\_3));

m2y = By - constant\_vector(1)\*sin(abs(u\_3));

x = [Ax m1x m2x Bx Ax];

y = [Ay m1y m2y By Ay];

figure(i);

plot(x, y);

end