Øving 8 - Likningsløsere - LF

Obligatoriske oppgaver

1 See the lecture notes.

2 **Matlab:**

```
%initialgjetning
x=1

y=x-1;
tol=.00001;

%Antall iterasjoner
n=0;
%fikspunkttierasjon x=cos x
while abs(y-x) > tol
    n = n+1;

    y=x;
    x=cos(x);
    disp([n,x, abs(x-y)]);
    pause(1);
end
```

Python:

```
import numpy as np
import time

#denne koden skj nner jeg

#initialgjetning
x=1

#denne koden er basert p  at vi overskriver en enkelt variabel istedet for ...
      lagre hele rekken av iterasjoner. derfor trenger vi en ekstra ...
   variabel for aa holde styr paa avstanden mellom iterasjonene.
y=x-1

#hvor tett skal det vaere mellom iterasjonene foer vi er fornoyde?
tol=10.0**-5

print("x(n)\t\t\t\t","|x(n+1)-x(n)|",sep='')
# teller antall iterasjoner utf rt
n = 0
#fikspunkt for x-cos x = 0
while np.abs(y-x) > tol: # vi holder paa saa lenge avstanden mellom ...
    iterasjonene er storre en tol
    y=x # y er naa forrige iterasjon
    x=np.cos(x) #dette er fikspunktiterasjonen
```

```
    time.sleep(.4) #en liten pause, slik at vi skal faa tid til aa nyte ...
        synet av den nye iterasjonen
    print(x, np.abs(x-y),sep='\t') #som printes her
    n = n + 1
print("-----------------------------------------------------")
print("Antall iterasjoner:",n)
```

3 Matlab:

```
x=1
y=x-1;
tol=.00001;

%Antall iterasjoner.
n=0;
%newton for x-cos x = 0
while abs(y-x) > tol
    n=n+1;

    y=x;
    x=x-(x-cos(x))/(1+sin(x));
    disp([n, x, abs(x-y)])
    pause
end
```

Python:

```
import numpy as np
import time

#denne koden er identisk i struktur med oving_8_1. se den koden for ...
    kommentarer.
x=1
y=x-1
tol=10.0**-6

print("x(n)\t\t\t\t","|x(n+1)-x(n)|",sep='')
#newton for x-cos x = 0
n = 0
while np.abs(y-x) > tol:
    y=x
    x=x-(x-np.cos(x))/(1+np.sin(x))
    time.sleep(.5)
    print(x, np.abs(x-y),sep="\t")
    n = n + 1
print("-----------------------------------------------------")
print("Antall iterasjoner:",n)
```

4 Python:

```
import numpy as np

def F(x,y):
    return np.array([x**2+y**2-4,x*y-1])
```

```
# I dette tilfellet er det enkleste    bruke formelen for inversen til en ...
    2*2 matrise.
# Som regel er ikke inversen s   lett tilgjengelig som her, da ville vi i ...
    stedet ha l st et likningssett for hvert steg i metoden

def J_inv(x,y):
    return 1/(2*x**2-2*y**2)*np.array(([x,-2*y],[-y,2*x]))

# N r vi har mer enn  n  dimensjon m  vi ta i bruk et nytt m l for    ...
    beregne hvor stor forbedringen er for hvert steg.
# En m te    gj re det p  er    bruke den evklidiske distansen mellom de ...
    nye koordinatene og de fra forrige steg,
# hvilket er det som er implementert i koden her.

def newton(x,y,tol):
    d = 1 # differansen mellom estimatet for n+1 og n.
    n = 0 # antall iterasjoner
    xp, yp = x,y # verdier fra forrige steg  s  man kan regne ut et feilestimat
    while tol<d:
        vec = np.array([x,y])-J_inv(x,y)@F(x,y) # vektor med oppdaterte x- ...
            og y-verdier
        x = vec[0]
        y = vec[1]
        d = np.linalg.norm(np.array([x,y])-np.array([xp,yp])) # regner ut ...
            lengden mellom de to vektorene.
        n = n+1
        xp,yp = x,y
    return x,y,n

if __name__ == "__main__":
    tol = 10.0 ** -14
    x0 = 2
    y0 = 0
    x,y,n = newton(x0,y0,tol)
    print("L sning:",np.array([x,y]))
    print("F(x,y) =",F(x,y))
    x_e = np.sqrt(2+np.sqrt(3)) # eksakte l sninger
    y_e = np.sqrt(4-x_e**2)
    print("Feil i svaret:",np.linalg.norm(np.array([x,y])-np.array([x_e,y_e])))
    print("Antall iterasjoner:",n)

    # Oppf lgningssp rsm l: Hva blir l sningen dersom man starter fra ...
        x0 = 0, y0 = 2?
```

5 Python:

```
import numpy as np

# Her har vi brukt at tredjeroten til 7 er en l sning av likningen x^3-7=0.
# Dersom vi anvender newtons metode p  denne likningen f r vi ...
    fikspunktiterasjonen vi l ser for i koden under
# Ellers er logikken den samme som i 08_2

def g(x):
    return (2*x**3+7.0)/(3*x**2)

def newton(x,tol):
    d = 1 # differanse mellom ny og forrige x
    n = 0
```

```
    while d>tol:
        xp = x
        x = g(x)
        d = np.abs(x-xp)
        n = n + 1
    return x,n

if __name__ == "__main__":
    tol = 10.0**-16
    x0 = 1.9 # Vi vet at 2^3=8, s  roten b r ligge rundt denne verdien. ...
       1.9 er derfor en fornuftig initialverdi.
    x,n = newton(x0,tol)
    print("L sning: x =",x)
    print("x^3 =",x**3.0)
    print("Feil:",x-7.0**(1.0/3))
    print("Antall iterasjoner:", n)
```

Anbefalte oppgaver

1 We want to solve

$$x \ln x = 1.$$

Naively multiplying both sides with x we get

$$x = x^2 \ln x =: g(x),$$

and this equation we will try to solve with the classical fixed point method.

Numerical test.

I tried using the fixed point method for several start values, but nothing seemed to work. Either the iterations went to $+\infty$ or to negative values ($\ln x$ is not defined for negative numbers). The question is why.

Analysis.

Instead of continuing with mindless computations, we want to discuss our expectations of the method. We know that

$$x = 1 \Rightarrow \ g(x) = 1^2 \cdot \ln 1 = 0 < 1.$$
$$x = e \Rightarrow \ g(x) = e^2 \cdot \ln e = e^2 > e.$$

meaning that there exists a fixed point on the interval $(1, e)$ thanks to the intermediate value theorem (write for example $f(x) = g(x) - x$ and use the theorem for this function). However we know that on the interval

$$(e, +\infty)$$

there is no solution. **Why?** Because here we have

$$\ln x > 1$$

and

$$x^2 > x,$$

so ultimately

$$x^2 \ln x > x,$$

so there cannot possibly exist a solution in this interval.

Choosing starting values

The point is, if we for example choose

$$x_0 = e$$

then $x_1 = e^2 > x_0$. Further, we get $x_2 = 2e^4 > x_1$, and so on. The point is that then $x_0 < x_2 < x_3 < \ldots$, all in the interval $(e, +\infty)$ **BUT** we just showed that there is no fixed point in this interval. Hence the fixed point method **diverges** with this start value!

However, we can do better.

Assume that we actually know the fixed point, let's call it y. We then know that

$$y \ln y = 1$$

and that $y \in (1, e)$. Let us choose our starting point x_0 a very small distance away from y,

$$x_0 = y + \epsilon.$$

Plugging this into our fixed point scheme we see

$$\begin{aligned} x_1 = g(x_0) &= (y+\epsilon)^2 \ln(y+\epsilon) \\ &= (y^2 + 2\epsilon y + \epsilon^2) \ln(y+\epsilon) \\ &\geq (y(y+2\epsilon) + \epsilon^2) \ln y \\ &\geq y \ln y(y+2\epsilon) = y + 2\epsilon > y + \epsilon, \end{aligned}$$

so that $x_1 > x_0$. Thus, starting above our fixed point leads to disaster, as the iterations give us bigger and bigger numbers.

What about starting below the fixed point???

We can do a similar technique. However, this is a bit more complicated. Take $x_0 = y - \epsilon$, then

$$\begin{aligned} x_1 = g(x_0) &= (y-\epsilon)^2 \ln(y-\epsilon) \\ &= (y^2 - 2\epsilon y + \epsilon^2) \ln(y-\epsilon) \\ &\leq (y - 2\epsilon) \ln y + \epsilon^2 \ln(y-\epsilon) \\ &< y - \epsilon. \end{aligned}$$

You need to argue that the term $\epsilon^2 \ln(y-\epsilon)$ is not making any trouble. Then we get $x_1 < x_0$, showing that we are moving away from our fixed point!

Conclusion:

The fixed point method will not work in this case. We are traveling away from our fixed point!

Other methods.

I tried also to define $g(x) = x\ln x + x - 1$, but numerical experiments (close to the fixed point) gave really bad results in this case as well. The morale of the story is that fixed point methods often lead to disaster ¯_(ツ)_/¯

2 We can define

$$f(x) = x\ln x - 1$$

and then by the product rule

$$f'(x) = 1 + \ln x.$$

The Newton iterations are then given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n \ln x_n - 1}{1 + \ln x_n}$$

We choose $x_0 = 2$. Using the calculator to do the iterations we end up with

$$x_n = 1.763222834$$

after $n = 3$ iterations. (Much better than the fixed point method!)

3 Let $f(x) = x^3 - x^2 + x + 2$. For a sufficiently large interval containing zero, we can observe that f has different signs in the end points. For example $I = [-10, 10]$, $f(10) = 912 > 0$ and $f(-10) = -1108 < 0$. By the intermediate value theorem, at least one solution of $f(x) = 0$ must exist on that interval. For $x \neq 0$ we have

$$f'(x) = 3x^2 - 2x + 1 > x^2 - 2x + 1 = (x-1)^2 \geq 0.$$

So since $f'(0) = 1 > 0$, we have $f'(x) > 0$ for all x. That is, the function $f(x)$ is strictly monotonically increasing on the real line. Thus, we must have a unique solution in the interval I. By using the bisection method on the interval I, we can narrow this interval to be $[-1, 0]$.

We have the following approximation of the error

$$\begin{aligned}\varepsilon_{n+1} &\approx -\frac{f''(s)}{2f'(s)}\varepsilon_n^2 \approx -\left[\frac{f''(s)}{2f'(s)}\right]^{1+2}\varepsilon_{n-1}^{2\cdot 2} \approx -\left[\frac{f''(s)}{2f'(s)}\right]^{1+2+4}\varepsilon_{n-2}^{2^3} \\ &\approx -\left[\frac{f''(s)}{2f'(s)}\right]^{2^0+2^1+2^2+2^3}\varepsilon_{n-3}^{2^4} \approx \cdots \approx -\left(\frac{f''(s)}{2f'(s)}\right)^{M}\varepsilon_0^{M+1}\end{aligned}$$

with

$$M = 2^0 + 2^1 + \cdots + 2^n = \sum_{m=0}^{n} 2^m = 2^{n+1} - 1.$$

n	x_n	$\lvert\varepsilon_n\rvert$
0	-1	0.189 464 286
1	$-0.833\,333\,333$	0.022 797 620
2	$-0.810\,916\,179$	0.000 380 466
3	$-0.810\,535\,822$	0.000 000 108

Tabell 1: Newton iterations from problem 1b.

Using one step of Newtons method, we estimate $s \approx -\frac{5}{6}$, such that we have $\varepsilon_0 = s - x_0 \approx \frac{1}{6}$. Moreover, we have

$$C = \frac{f''(s)}{2f'(s)} = \frac{1}{2}\frac{6s-2}{3s^2-2s+1} \approx -\frac{14}{19}.$$

Demanding 5 decimals accuracy ($|\varepsilon_{n+1}| < 10^{-5}$), we get

$$|\varepsilon_{n+1}| \approx \left| -\left(\frac{f''(s)}{2f'(s)}\varepsilon_0\right)^M \varepsilon_0 \right| \approx |C\varepsilon_0|^M |\varepsilon_0| < 10^{-5} \quad \Rightarrow \quad M > \frac{-5 - \log_{10}|\varepsilon_0|}{\log_{10}|C\varepsilon_0|}$$

$$\Rightarrow \quad n+1 > \log_2\left[1 + \frac{-5-\log_{10}|\varepsilon_0|}{\log_{10}|C\varepsilon_0|}\right] \approx 2.49$$

Using $n+1 = 3$ iterations we then get a solution with 5 decimals accuracy.

The iterations are computed with the formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^3 - x_n^2 + x_n + 2}{3x_n^2 - 2x_n + 1} = \frac{2x_n^3 - x_n^2 - 2}{3x_n^2 - 2x_n + 1}$$

Starting with $x_0 = -1$ yields the convergence table in Table 1.

4 See the lecture notes.