Matematiske metoder
Numerikk
Høst 2021

Løsningsforslag - Øving 2

Fra Sauer 2E, avsnitt 1.3

1 Foroverfeilen er den samme for alle uttrykkene: $3/4 - 0{,}74 = 0{,}01$. Bakoverfeilen for hvert uttrykk blir

a) $4 \cdot 0{,}74 - 3 = -0{,}04 = 4{,}0 \cdot 10^{-2}$.

b) $(4 \cdot 0{,}74 - 3)^2 = 0{,}0016 = 1{,}6 \cdot 10^{-3}$.

c) $(4 \cdot 0{,}74 - 3)^3 = -0{,}000064 = -6{,}4 \cdot 10^{-5}$

d) $(4 \cdot 0{,}74 - 3)^{1/3} \approx 0,341995189 \approx -3{,}42 \cdot 10^{-1}$

3 **a)** Multiplisiteten til roten $r = 0$ til $f(x) = 1 - \cos x$ er lik 2 fordi $f(0) = 0$, $f'(0) = \sin(0) = 0$ og $f''(0) = \cos(0) = 1$. Forover feilen er 0,0001 og bakoverfeilen er $f(0{,}0001) \approx 0,000000005 = 5,0 \cdot 10^{-9}$

Fra Sauer 2E, avsnitt 1.4

3a Vi har $f(x) = x^5 - 2x^4 + 2x^2 - x$ med røtter $r = -1$, $r = 0$, $r = 1$. Vi regner ut et tilstrekkelig antall deriverte. Tabell for de forskjellige verdiene av r

r	-1	0	1
$f'(r) = 5x^4 - 8x^3 + 4x - 1$	8	-1	0
$f''(r) = 20x^3 - 24x^2 + 4$	-40	4	0
$f'''(r) = 60x^2 - 48x$			12
m	1	1	3
$M = \lvert f''(r)/2f'(r)\rvert$	$2{,}5$	2	
$S = (m-1)/m$			$2/3$

Teorem 1.11 sier at newtons metode er kvadratisk konvergent når $f'(r) \neq 0$. Da er $e_{i+1} \approx M\, e_i^2$. Teorem 1.12 sier at newtons metode er lineær konvergent når r har multiplisitet $m > 1$. Da er $e_{i+1} \approx S\, e_i$.

r	
-1	$e_{i+1} \approx 2{,}5\, e_i^2$
0	$e_{i+1} \approx 2\, e_i^2$
1	$e_{i+1} \approx \frac{2}{3}\, e_i$

CP6 Volumet av en kjegle med sirkulær gunnflate med radius r er $\frac{1}{3}hG$, der h er høyden og $G = \pi r^2$ er arealet av grunnflaten. Volumet av halvkulen er $\frac{2}{3}\pi r^3$. Iskremens volum som funksjon av r er derfor $V = \frac{10}{3}\pi r^2 + \frac{2}{3}\pi r^3$. Siden vi vil finne r når $V = 60$ så har vi

likningen $f(r) = \frac{10}{3}\pi r^2 + \frac{2}{3}\pi r^3 - 60 = 0$ Vi deriverer og får $f'(r) = \frac{20}{3}\pi r + 2\pi r^2$. Newtons iterasjonsformel:

$$r_{n+1} = r_n - \frac{f(r_n)}{f'(r_n)}.$$

Vi starter med $r = 1$.

```
import math

# Funksjonen og dens deriverte samt r-f(r)/f'(r)
f = lambda r: (10*math.pi*r**2/3)+2*math.pi*r**3/3-60
Df = lambda r: (20*math.pi*r/3)+2*math.pi*r**2
newton = lambda r: r-f(r)/Df(r)

noyaktighet = 4 # antall signifikante sifre etter komma
maxfeil=0.5*10.0**(-noyaktighet);
r=1 # Vi starter med r=1
i=0

print("r[0] = 1");

while( not (Df(r)==0)):
    newr=newton(r);
    i=i+1;
    print("r[{0}] = {1}".format(i,newr))
    if (math.fabs(newr-r) < maxfeil): break
    r=newr;
```

Koden produserer:

```
r[0] = 1
r[1] = 2.74214536589
r[2] = 2.1505412939
r[3] = 2.02555130674
r[4] = 2.02011739726
r[5] = 2.02010732125
```

Fra Sauer 2E, avsnitt 2.1

2b Vi løser følgende lineære system

$$\begin{aligned} x + 2y - z &= 2 \\ 3y + z &= 4 \\ 2x - \ y + z &= 2 \end{aligned}$$

ved Gauss eliminasjon på totalmatrisen:

$$\left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 2 & -1 & 1 & 2 \end{array}\right] \xrightarrow{(-2)} \sim \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 0 & -5 & 3 & -2 \end{array}\right] \xrightarrow{(5/3)} \sim \left[\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 0 & 0 & 14/3 & 14/3 \end{array}\right].$$

Nederste rad står for $(14/3)z = 14/3$ som gir $z = 1$. Midterste rad står for $3y + z = 4$. Vi setter inn $z = 1$ og vi får $y = 1$. Vi gjør tilsvarende for rad 1 og får $x = 1$. Løsning $(x, y, z) = (1, 1, 1)$.

6 Vi har et system med 5000 ukjente og likninger. I følge oppgaven bruker tilbakeinsetningen n^2 operasjoner. Eliminasjonen bruker $2n^3/3$ operasjoner. Siden tilbakeinsetningen tar 0,005 sekunder så bruker vi $T = \frac{0{,}005}{n^2}$ sekund for hver operasjon. Tiden eliminasjonen tar er derfor $2n^3T/3 = 0{,}01n/3 \approx 16{,}3$ sekunder.

CP1 Vi har koden

```python
import numpy as np
# a: liste med lister av flyttall som definerer en
   kvadratisk matrise
# b: en liste med flyttall. Programmet regner ut losningen
   til ax=b
def GaussElimSolver( A , b ):
    n=len(b)
    # copy A and b so that they are not changed.
    a = np.array(A);
    b = list(b)

    for j in range(0,n-1):
        # Eliminer alle elementer under (j,j)
        for i in range(j+1,n):
            mult = a[i,j]/a[j,j]
            a[i,j]=0
            for k in range(j+1,n):
                a[i,k] = a[i,k] - mult*a[j,k]
            b[i]=b[i] - mult*b[j]

    # Tilbakeinnsetting
    x = [0] * n
    for i in range(n-1,-1,-1):
        for j in range(i+1,n):
            b[i] = b[i] - a[i,j]*x[j]
        x[i] = b[i]/a[i,i]

    return x
```

Denne kan for eksempel lagres i filen *minefunksjoner.py*. Det gjør at vi kan gjenbruke koden eneklt i neste oppgave. Vi bruker koden

```python
import nympy as np
from minefunksjoner import GaussElimSolver
A1 = np.array([[ 2.0,-2.0,-1.0],
               [ 4.0, 1.0,-2.0],
               [-2.0, 1.0,-1.0]])
b1 = [-2.0, 1.0,-3.0]
A2 = np.array([[ 1.0, 2.0,-1.0],
               [ 0.0, 3.0, 1.0],
               [ 2.0,-1.0, 1.0]])
b2 = [ 2.0, 4.0, 2.0]
A3 = np.array([[ 2.0, 1.0,-4.0],
               [ 1.0,-1.0, 1.0],
               [-1.0, 3.0,-2.0]])
b3 = [-7.0,-2.0, 6.0]
```

```
x1 = GaussElimSolver(A1,b1);
x2 = GaussElimSolver(A2,b2);
x3 = GaussElimSolver(A3,b3);

print(x1)
print(x2)
print(x3)
```

gir løsningene $\begin{bmatrix}1 & 1 & 2\end{bmatrix}^T$, $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$, og $\begin{bmatrix}-1 & 3 & 2\end{bmatrix}^T$.

CP2 Følgende skript lager Hilbertmatriser og løser systemet $H\mathbf{x} = \mathbf{b}$.

```
import numpy as np
from minefunksjoner import GaussElimSolver

for n in [2,5,10]:
    X=range(1,n+1)
    Y=range(1,n+1)
    grids = np.meshgrid(X,Y)# Lager matriser I_ij=i og J_ij=j
    I=grids[0]
    J=grids[1]
    H = 1.0/(I+J-1) #1.0 forces use of double floating point.
    b = np.ones([n,1]);
    x = GaussElimSolver(H,b);
    feil = np.max(np.abs(np.dot(H,x)-b)) #bakoverfeil
    print(feil)
```

Kjører scriptet og får ut bakoverfeilen a) 0, b) 1.4211e-14, c) 1.0186e-10

Fra Sauer 2E, avsnitt 2.2

2b

$$\begin{bmatrix}4 & 2 & 0\\ 4 & 4 & 2\\ 2 & 2 & 3\end{bmatrix} \sim \begin{bmatrix}4 & 2 & 0\\ (1) & 2 & 2\\ (1/2) & 1 & 3\end{bmatrix} \sim \begin{bmatrix}4 & 2 & 0\\ (1) & 2 & 2\\ (1/2) & (1/2) & 2\end{bmatrix}$$

Vi har derfor $L = \begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 1/2 & 1/2 & 1\end{bmatrix}$ og $U = \begin{bmatrix}4 & 2 & 0\\ 0 & 2 & 2\\ 0 & 0 & 2\end{bmatrix}$. Vi regner ut

$$LU = \begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 1/2 & 1/2 & 1\end{bmatrix}\begin{bmatrix}4 & 2 & 0\\ 0 & 2 & 2\\ 0 & 0 & 2\end{bmatrix} = \begin{bmatrix}4 & 2 & 0\\ 4 & (2+2) & 2\\ 2 & (1+1) & (1+2)\end{bmatrix} = \begin{bmatrix}4 & 2 & 0\\ 4 & 4 & 2\\ 2 & 2 & 3\end{bmatrix} = A.$$

8 Det tar ca $2n^3/3 = 16/3 \cdot 10^9$ operasjoner å løse et 2000×2000 lineært system. Det gir $10 \cdot 2n^3/3 = 5.33 \cdot 10^{10}$ operasjoner per sekund. k $2n^3/3 + 2k^2$ operasjoner. $k = 100$ systemer av $n = 8000$ likninger i 8000 ukjente med samme koefisientmatrise A krever ca $2n^3/3 + 2kn^2 = 35.41 \cdot 10^{10}$ operasjoner. Det vil ta

$$\frac{35.41}{5.33} \approx 6.7 \text{ sekunder.}$$

CP1 MATLAB-koden gjør en LU-faktorisering

```
def LUtransform(A):
    n=A.shape[0]
    U = np.array(A)
    L = np.eye(n)
    for j in range(0,n-1):
        # Eliminer alle elementer under (j,j)
        for i in range(j+1,n):
            mult = U[i,j]/U[j,j]
            L[i,j] = mult
            U[i,j] = 0
            for k in range(j+1,n):
                U[i,k] = U[i,k] - mult*U[j,k]
    return [L,U]
```

Følgende skript tester koden på matrisene fra oppgave 2

```
import numpy as np
from minefunksjoner import LUtransform

A = np.array([[3,1,2],
              [6,3,4],
              [3,1,5]])
print(LUtransform(A))

B = np.array([[4,2,0],
              [4,4,2],
              [2,2,3]])
print(LUtransform(B))

C = np.array([[1,-1,1, 2],
              [0, 2,1, 0],
              [1, 3,4, 4],
              [0, 2,1,-1]])
print(LUtransform(C))
```

CP2

```
def LUsolve(A,b):
    n=A.shape[0]
    LU=LUtransform(A)
    L=LU[0]
    U=LU[1]
    b=np.array(b)
    c=np.zeros([n,1])
    for i in range(0,n):
        tmp=b[i]
        for k in range(0,i):
            tmp=tmp-L[i,k]*c[k,0]
        c[i]=tmp
    x=np.zeros([n,1])
    for i in range(n-1,-1,-1):
```

```
        tmp=c[i]
        for k in range(i+1,n):
            tmp=tmp-U[i,k]*x[k,0]
        x[i]=tmp/U[i,i]
    return x
```

Følgende script løser likningene i oppgave 4.

```
import numpy as np
from minefunksjoner import LUsolve

A = np.array([[3,1,2],
              [6,3,4],
              [3,1,5]])
b = np.array([[0],[1],[3]])
print(LUsolve(A,b))

A = np.array([[4,2,0],
              [4,4,2],
              [2,2,3]])
b = np.array([[2],[4],[6]])
print(LUsolve(A,b))
```

Fra Sauer 2E, avsnitt 2.3

2b Gitt matrisen $A = \begin{bmatrix} 1 & 2.01 \\ 3 & 6 \end{bmatrix}$ Da er $\|A\|_\infty = 9$ den inverse av A er $A^{-1} = \frac{1}{-0.03}\begin{bmatrix} 6 & -2.01 \\ -3 & 1 \end{bmatrix}$ Da er $\|A^{-1}\| = \frac{8.01}{0.03} = 267$. Derfor er norm $A = 9 \cdot 267 = 2403$.

13a Matrisen $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ har uendelig-norm $\|A\|_\infty = 7$, (Største absolutte radsum.) La $\mathbf{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$. Vi har at $\|A\mathbf{x}\|_1 = \left\|\begin{bmatrix} 3 & 7 \end{bmatrix}^T\right\|_\infty = 7$ og $\|\mathbf{x}\|_\infty = 1$. Derfor er

$$\|A\mathbf{x}\|_\infty = \|A\|_\infty \|\mathbf{x}\|_\infty.$$

14a Matrisen $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ har max-norm $\|A\|_1 = 6$, (Største absolutte kolonnesum.) La $\mathbf{x} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Vi har at $\|A\mathbf{x}\|_1 = \left\|\begin{bmatrix} 2 & 4 \end{bmatrix}^T\right\|_1 = 6$ og $\|\mathbf{x}\|_1 = 1$. Derfor er

$$\|A\mathbf{x}\|_1 = \|A\|_1 \|\mathbf{x}\|_1.$$

CP6 Vi kjører koden

```
import numpy as np
from minefunksjoner import LUsolve

#Versjon 2:
A2=np.array([[10.0**(-20),1],
             [1,2]])
b2=np.array([1,4]);
```

```
x2=LUsolve(A2,b)

#Versjon 3:
A3=np.array([[1,2],
             [10.0**(-20),1]])
b3=np.array([4,1]);
x3=LUsolve(A3,b)
```

og får riktig svar $x = [2, 1]^T$ i versjon 3. Versjon 2 gir svaret $x = [0, 1]^T$ som er feil. Årsaken er oversvømming (swamping).