Varmelikningen

Jens Aasgard

April 2025

1 Introduksjon

Contents

1.1 Oppgave 1

Skal bruka formelen $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ til å finne stigningstallet til ein funksjon. Testar formelen på funksjonen $f(x) = e^x$ i x = 1.5. Ved ulike h-verdiar blir resultatet som vist i tabell 1. Kode i seksjon 2.1.

h-steg	**Stigningstall**
0.1	4.713433540570504
10^{-2}	4.5041723976187775
10^{-3}	4.483930662008362
10^{-4}	4.481913162264206
10^{-5}	4.4817114789097445
10^{-6}	4.48169131139764
10^{-7}	4.481689304114411
10^{-8}	4.481689064306238
10^{-9}	4.481689686031132

Table 1: Stigningstall for ulike h-verdier

1.2 Oppgave 2

Dersom vi bruker formel(1) til å finne den deriverte til $f(1.5) = e^{1.5}$ blir tilnerminga betre.

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) \tag{1}$$

Ressultatet ved ulike h-verdiar er vist i tabell 2.

h-steg	**Stigningstall**
1	5.266886345001673
10^{-1}	4.489162287752202
10^{-2}	4.481763765529401
10^{-3}	4.481689817286139
10^{-4}	4.48168907780655
10^{-5}	4.481689070434669
10^{-6}	4.481689070079398
10^{-7}	4.481689073188022
10^{-8}	4.481689019897317

Table 2: Stigningstall for ulike h-verdier

Kvifor formelen tilnermar den deriverte raskare kan forklarast med taylorrekker. Taylorrekka er gitt ved:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots \tag{2}$$

Ved å sette in x+h og x-h får vi dei to taylorrekkene i formel(1).

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)h^2}{2!} - \frac{f'''(x)h^3}{3!} \tag{3}$$

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2!} + \frac{f'''(x)h^3}{3!} \tag{4}$$

Deretter kan vi sette in i formel(1).

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{f'''(x)h^2}{3!} \tag{5}$$

Vi ser av likning 5 at feilen $\frac{f'''(x)h^2}{3!}$ minkar med kvadratet av h. Det betyr at dersom vi minkar h-steg med 10 så vil feilen minke med 100.

1.3 Oppgave 3

For å få få ei enda betre tilnerming av den deriverte kan vi bruka formel(6).

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} \tag{6}$$

Med denne formelen får vi verdiar som vist i tabell 3.

h-steg	**Stigningstall**
1	4.313438351753924
10^{-1}	4.481674113579637
10^{-2}	4.481689068844186
10^{-3}	4.481689070337709
10^{-4}	4.481689070338449
10^{-5}	4.481689070390259
10^{-6}	4.481689070005383
10^{-7}	4.481689073928171
10^{-8}	4.481688997692856

Table 3: Stigningstall for ulike h-verdier

Vi ser ut i fra tabell 3 at feilen her minkar raskare en metoden i oppgave 2. Vi ser i oppgave 1, 2 og 3 at det er ei grense for kor presise tall vi får. Når h blir tilstrekkelig liten så begynner feilen å bli større igjen.

1.4 Oppgave 4

Euler eksplisitt er gitt ved:

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \tag{7}$$

Dersom vi gongar med k og flytter $u_{i,j}$ over får vi utrykket:

$$u_{i,j+1} = \frac{k}{h^2}u_{i+1,j} + (1 - \frac{2k}{h^2})u_{i,j} + \frac{k}{h^2}u_{i-1,j} \tag{8}$$

Dersom vi deler opp x aksen opp i punkter så kan vi løyse likning (8) for kvart av punkta på xaksen. Når vi har løyst for alle xverdiane kan koden gå vidare

til å løyse for neste tidssteg. Det vil seie at vi gjer ei slik utrekning for kvart av punkta i gitteret i figur 1.

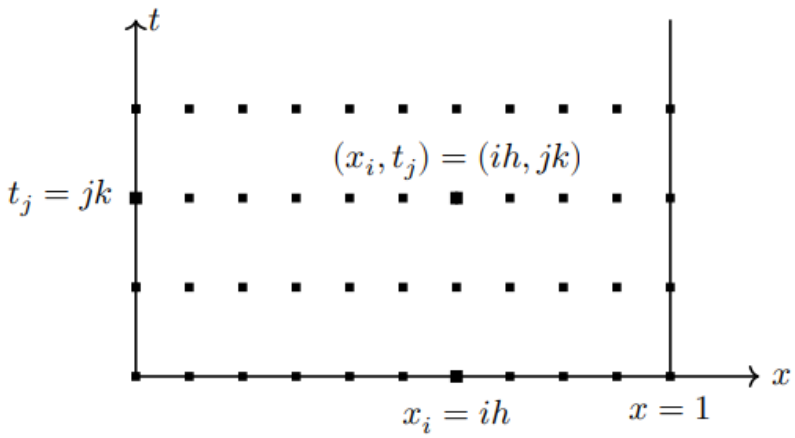
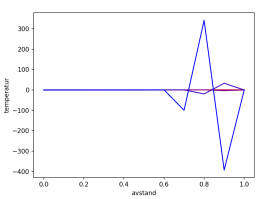
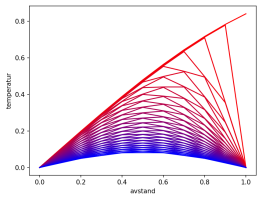


Figure 1: Gitter for numerisk løysing

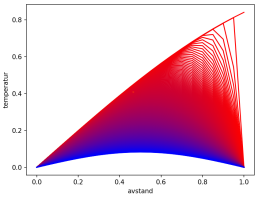
Eit plott av løysinga er vist i figur 2. Kode i seksjon 2.2.



(a) $h = 0.1k = 0.05$



(b) h = 0.1 k = 0.0005



(c) h = 0.05 k = 0.0008

Figure 2: Eksplisitt metode

Ved å teste forskjellige verdiar viser det seg at den eksplistitte formelen er kresen på kva verdiar for h og k. Når k er mykje mindre en h så fungerer den eksplisitte metoden. Vi ser fra figur 2(c) at for å dele xaksen opp i mindre biter så må k verdien også bli mykje mindre for at løysinga skal fungere.

1.5 Oppgave 5

likninga for euler implisitt er gitt ved:

$$\frac{u_{i,j+1} - u_{i,j}}{k} = \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{h^2} \tag{9}$$

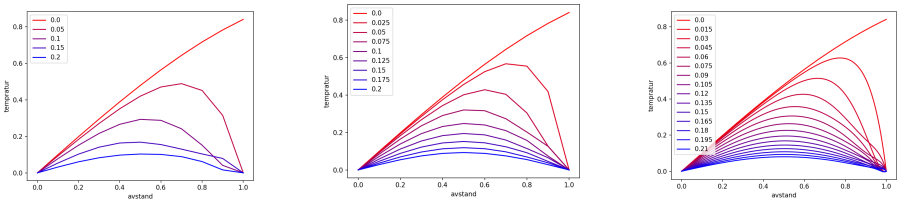
Dersom vi løyser likninga slik at $u_{i,j}$ kommer aleine så får vi:

$$u_{i,j} = -\frac{k}{h^2}u_{i+1,j+1} + (1 + \frac{2k}{h^2})u_{i,j+1} - \frac{k}{h^2}u_{i-1,j+1} \tag{10}$$

I likhet med den eksplisitte metoden kan vi også løyse denne likningen for kvart punkt i gitteret. Men sidan vi har fleire en 1 ukjent i likninga over så må vi bruke fleire likningar saman og løyse dei som eit likningsystem. Vi kan sette opp likningsystemet som Ax = b:

$$\begin{bmatrix} 1+\frac{2k}{h^2} & -\frac{k}{h^2} & 0 & \cdots & 0 \\ -\frac{k}{h^2} & 1+\frac{2k}{h^2} & -\frac{k}{h^2} & \cdots & 0 \\ 0 & -\frac{k}{h^2} & 1+\frac{2k}{h^2} & -\frac{k}{h^2} & \vdots \\ \vdots & \vdots & -\frac{k}{h^2} & \ddots & -\frac{k}{h^2} \\ 0 & 0 & \cdots & -\frac{k}{h^2} & 1+\frac{2k}{h^2} \end{bmatrix} \cdot \begin{bmatrix} u_{1,i+1} \\ u_{2,i+1} \\ u_{3,i+1} \\ \vdots \\ u_{n-1,i+1} \\ u_{n,i+1} \end{bmatrix} = \begin{bmatrix} u_{1,i} \\ u_{2,i} \\ u_{3,i} \\ \vdots \\ u_{n-1,i} \\ u_{n,i} \end{bmatrix}$$

Dette likningsystemet kan vi løyse. B-vektoren er kjente tall som har blitt rekna ut. x vektoren er ukjent, men vi kan rekne ut x med np.linalg.solve(A,b) i python. Ved å plotte ressultatet får vi grafer som vist i figur 3. Kode i seksjon 2.3.



(a) $h = 0.1k = 0.05$ (b) h = 0.1 k = 0.25 (c) h = 0.001 k = 0.015

Figure 3: Implisitt

Figur 3 viser at den implisitt løysinga fungerer bra for forskjellige verdiar av h og k. Den fungerer når k er mindre en h og når k er større en h.

1.6 Oppgave 6

Crank-Nicolson er gitt ved formel(11).

$$\frac{u_{i,j+1}-u_{i,j}}{k} = \frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{2h^2} + \frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{2h^2} \quad (11)$$

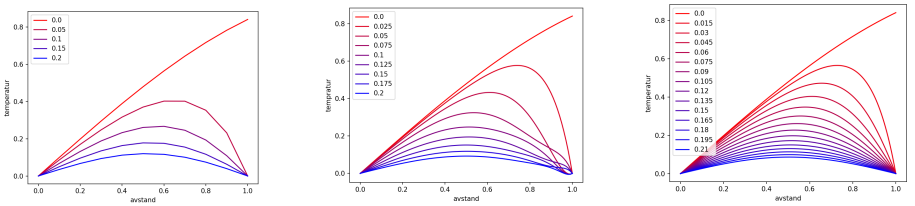
Dette kan vi skrive om til:

$$\frac{k}{h^2}u_{i+1,j} + 2(1-\frac{k}{h^2})u_{i,j} + \frac{k}{h^2}u_{i-1,j} = \frac{k}{h^2}u_{i+1,j+1} - 2(1+\frac{k}{h^2})u_{i,j+1} + \frac{k}{h^2}u_{i-1,j+1} \quad (12)$$

Dersom vi setter opp ei slik likning for kvart punkt på gitteret kan vi sette likningane saman til eit likningsystem som:

$$\begin{bmatrix} 2+\frac{2k}{h^2} & -\frac{k}{h^2} & 0 & \cdots & 0 \\ -\frac{k}{h^2} & 2+\frac{2k}{h^2} & -\frac{k}{h^2} & \cdots & 0 \\ 0 & -\frac{k}{h^2} & 2+\frac{2k}{h^2} & -\frac{k}{h^2} & \vdots \\ \vdots & \vdots & -\frac{k}{h^2} & \ddots & -\frac{k}{h^2} \\ 0 & 0 & \cdots & -\frac{k}{h^2} & 2+\frac{2k}{h^2} \end{bmatrix} \cdot \begin{bmatrix} u_{1,i} \\ u_{2,i} \\ u_{3,i} \\ \vdots \\ u_{n-1,i} \\ u_{n,i} \end{bmatrix} = \begin{bmatrix} 1+\frac{2k}{h^2} & -\frac{2k}{h^2} & 0 & \cdots & 0 \\ -\frac{2k}{h^2} & 1+\frac{2k}{h^2} & -\frac{2k}{h^2} & \cdots & 0 \\ 0 & -\frac{2k}{h^2} & 1+\frac{2k}{h^2} & -\frac{2k}{h^2} & \vdots \\ \vdots & \vdots & -\frac{2k}{h^2} & \ddots & -\frac{2k}{h^2} \\ 0 & 0 & \cdots & -\frac{2k}{h^2} & 1+\frac{2k}{h^2} \end{bmatrix} \cdot \begin{bmatrix} u_{1,i+1} \\ u_{2,i+1} \\ u_{3,i+1} \\ \vdots \\ u_{n-1,i+1} \\ u_{n,i+1} \end{bmatrix}$$

I python koden brukar vi ei forløkke som løyser likningssystemet for kvar verdi av t. ved å plotte løysinga får vi grafane vist i figur 4. Kode i seksjon 2.4.



(a) $h = 0.1k = 0.05$ (b) h = 0.1 k = 0.25 (c) h = 0.001 k = 0.015

Figure 4: Crank-Nicolson

Figur 4 viser at Crank-Nicolson fungerer bra og i likheit med den implisitte metoden kan h både vere større og mindre en k.

1.7 analytisk løysing av varmelikningen

Varmalikningen er gitt ved:

$$\dot{u}(x,t) = u''(x,t) \tag{13}$$

$$\frac{y''(x)}{y(x)} = \frac{\dot{z}(t)}{z(t)} = k \tag{14}$$

$$y''(x) - ky(x) = 0 \tag{15}$$

Bruker løysing y(x) = e^{rx}

$$r^2 - k = 0 \tag{16}$$

$$k = -\lambda^2 \tag{17}$$

$$r = \pm i\lambda \tag{18}$$

k er negativ derfor:

$$y(x) = Acos(\lambda x) + Bsin(\lambda x) \tag{19}$$

$$y(0) = Acos(\lambda x) \implies A = 0 \tag{20}$$

$$y(1) = 0 = Bsin(\lambda) = 0 \implies \lambda = n\pi \tag{21}$$

$$\dot{z} = -\lambda^2 z(t) \implies z(t) = e^{-\lambda^2 t} = A_n e^{-n^2\pi^2 t} \tag{22}$$

$$\dot{u}(x,t) = A_n sin(n\pi t) e^{-n^2\pi^2 t} \tag{23}$$

$$A_n = 2\int_0^1 sin(x) sin(n\pi x) dx = \left[\frac{sin((1-n\pi)x}{1-n\pi} - \frac{sin(1+n\pi)x}{1+n\pi} \right]_0^1 \tag{24}$$

Når vi løyser likningen for dei samme punktene som dei numeriske metodane får vi plottet vist i figur 5. Koden er i seksjon 2.5.

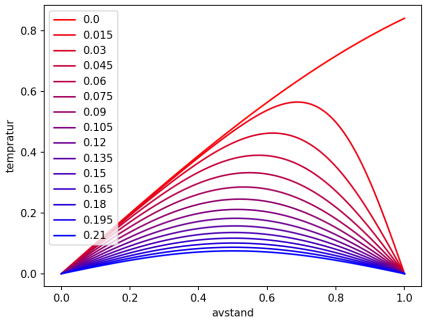


Figure 5: analytisk metode

Figur 5 av den analytiske løysinga liknar veldig på spesielt den implisitte metoden og Crank-Nicolson. Det ser ut som at den analytiske metoden er meir presis en dei to andre og avheger mindre av verdiane h og k.

2 Kode

2.1 Kode for numerisk derivasjon

```
import numpy as np
h = 10**(-2)

x = 1.5

def df(h, x):
    return (np.exp(x+h)-np.exp(x))/h
resultat = df(h,x)
print(resultat)

def d2f(h,x):
    return(np.exp(x+h)-np.exp(x-h))/(2*h)
resultat2 = d2f(h,x)
print(resultat2)

def f(x,h):

    return (np.exp(x-2*h)-8*np.exp(x-h)+8*np.exp(x+h)-np.exp(x+2*h))/(12*h)
resultat3 = f(x,h)
print (resultat3)
```

2.2 Kode for eksplisitt metode

```
import numpy as np
import matplotlib.pyplot as plt

# velger verdiar for h og k
h = 0.05
k = 0.0008
x  = np.arange(0, 1+h,h)
t = np.arange(0, 0.2+k,k)

n = len(x)
m = len(t)
#lagar ei matrise der antall punkt i tid blir representer i antall kolonner
og antall punkt i x blir antall rader.
#setter så første og siste rad til 0, og første kolonne til initialfunksjonen.
T = np.zeros((n,m))
T[-1,:] = 0
T[0,:] = 0
T[:,0] = np.sin(x)


faktor = k/h**2

# går så gjennom kvart av punkta og løyser med den eksplisitte formelen.
for j in range(1,m):
    for i in range(1,n-1):
        T[i,j]=faktor*T[i-1,j-1]+ (1-2*faktor)*T[i,j-1]+ faktor*T[i+1,j-1]

for j in range(m):
    R= np.linspace(1,0,m)
    B = np.linspace(0,1,m)
    G = 0
    plt.plot(x,T[:,j], color = [R[j],G,B[j]], label = round(t[j],3))
#plt.plot(T)
#plt.legend(t)
plt.xlabel("avstand")
plt.ylabel("temperatur")
plt.show()
```

2.3 Kode for implisitt metode

```
import numpy as np
import matplotlib.pyplot as plt
h = 0.1
k = 0.025
x = np.arange(0,1+h,h)
t = np.arange(0,0.2+k,k)



n = len(x)
m = len(t)
T = np.zeros((n,m))
T[0,:] = 0
T[-1,:] = 0
T[:,0] = np.sin(x)
faktor = k/h**2

A = np.diag([1+2*faktor]*(n-2),0)+ np.diag([-faktor]*(n-3),-1)+ np.diag([-faktor
#print (A)
for j in range(1,m):
    b = T[1:-1,j-1].copy()

    resultat = np.linalg.solve(A,b)
    T[1:-1,j] = resultat
    print (resultat)

print (b.round(3))
print (T.round(3))

R= np.linspace(1,0,m)
B = np.linspace(0,1,m)
G = 0
for j in range(m):
    plt.plot(x,T[:,j], color = [R[j],G,B[j]], label = round(t[j],3))

plt.xlabel('avstand')
plt.ylabel('temperatur')
plt.legend()
plt.show()
```

2.4 Kode for Crank-Nicolsons metode

```
import numpy as np
import matplotlib.pyplot as plt

h = 0.1
k = 0.05
h = 0.1
k = 0.025
h = 0.001
k = 0.015
x = np.arange(0,1+h,h)
t = np.arange(0,0.2+k,k)



n = len (x)
m = len(t)
#lagar ei matrise der antall punkt i tid blir representer i antall kolonner og a
#setter så første og siste rad til 0, og første kolonne til initialfunksjonen.
T = np.zeros((n,m))
T[0,:] = 0
T[-1,:] = 0
T[:,0] = np.sin(x)




faktor = k/h**2
#Setter opp dei to matrisene som ingår i likningsystmet. np.diag plaserer elemen
#som er n-2 lang. np.diag([-faktor]*(n-3),-1) plaserer -faktor n-3 ganger på dia
A = np.diag([2+2*faktor]*(n-2),0) + np.diag([-faktor]*(n-3),-1)+ np.diag([-fakto
B = np.diag([2-2*faktor]*(n-2),0) + np.diag([faktor]*(n-3),-1) + np.diag([faktor

for j in range(0,m-1):
    b = T[1:-1,j].copy()
    # tar prikkproduktet mellom matrisa B og kollona ved tidspunktet j.
    b = np.dot(B,b)
    b[0] = b[0] + faktor*(T[0,j]+ T[0,j+1])
    b[-1] = b[-1] + faktor*(T[-1,j] + T[-1,j+1])
    #bruker til å løyse likningsystemet og få den neste kolonna i matrisa.
    løysing = np.linalg.solve(A,b)
    #legger til kolonna i matrisa.
    T[1:-1,j+1] = løysing
R= np.linspace(1,0,m)
```

```
B = np.linspace(0,1,m)
G = 0
for j in range(m):
    plt.plot(x,T[:,j], color = [R[j],G,B[j]], label = round(t[j],3))

plt.xlabel('avstand')
plt.ylabel('tempratur')
plt.legend()
plt.show()
```

2.5 Kode for analytisk løysing av varmelikningen

```
import numpy as np
import matplotlib.pyplot as plt

def An(n):
    return (np.sin((1-n*np.pi)*1)/(1-n*np.pi)-np.sin((1+n*np.pi)*1)/(1+n*np.pi))
    -(np.sin((1-n*np.pi)*0)/(1-n*np.pi)-np.sin((1+n*np.pi)*0)/(1+n*np.pi))


h = 0.001
k = 0.015
x = np.arange(0,1+h,h)
t = np.arange(0,0.2+k,k)

initialkrav = np.sin(x)

n = len (x)
m = len(t)

T2 = np.zeros((n,m))
T2[:,0] = initialkrav


def u(x,t):
    resultat = 0
    for w in range(10):
        resultat += An(w)*np.sin(w*np.pi*x)*np.exp(-(w*np.pi)**2*t)
    return resultat

for j in range(1,m):
    for i in range (1,n-1):
        T2[i,j] = u(i*h,j*k)
T2[:,0] = initialkrav

R= np.linspace(1,0,m)
B = np.linspace(0,1,m)
G = 0


for j in range(m):
    plt.plot(x,T2[:,j], color = [R[j],G,B[j]], label = round(t[j],3))

plt.xlabel('avstand')
plt.ylabel('tempratur')
plt.legend()
```

```
plt.show()
```