Hadde litt problemer med programmeringen på de tre siste oppgavene, men vi har svart noe og virkelig prøvd.

Oblig 2025

søndag 23. mars 2025 12:25

Varmeligningen er grei å starte med:

Varmeligningen: $\dot{u}(x,t) = \alpha \Delta u(x,t)$

Den deriverte til en funksjon f:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Hvor grenseverdien eksisterer

Stigningstallet til sekanten til f mellom punktene x og x+h

$$\frac{f(x+h) - f(x)}{h}$$

Dette er en grei tilnærming for stigningstallet $\to^{f(x)}$ ved små h.

1

Anta at:

$f(x) = e^x$

og bruker derivasjon punktet f(1,5)

$f'(1,5) = e^{1,5} = 4,4817$

Når $h = 0,01$ får vi fra formelen.

$$f'(1,5) = \frac{e^{1,51} - e^{1,5}}{0,01} = 4,5017$$

Her "kommer" den med ca $2 \cdot 10^{-2}$

Prøver med $h = 0,001$ og får da:

$$f'(1,5) = \frac{e^{1,501} - e^{1,5}}{0,001} = 4,4839.$$

Begynner å nærme oss. kommer med ca $2 \cdot 10^{-3}$

Prøver med $h = 0,0001$ og får da.

$$f'(1,5) = \frac{e^{1,5001} - e^{1,5}}{0,0001} = 4,4819$$

Her kommer den med $2 \cdot 10^{-4}$ (Begynner å bli en god tilnærming)

Prøver med $h = 0,00001$ og får:

$f'(1,5) = \frac{e^{1,500001} - e^{1,5}}{0,000001} = 4,4817$ (Det vi vil ha, men prøver en ekstra gang for å se om alle desimalene blir som forventet)

Prøver $h = 0,0000001$ og får

$f'(1,5) = \frac{e^{1,5000001} - e^{1,5}}{0,0000001} = 4,48169$ (litt under det vi vil ha)

Prøver med $h = 0,00000001$ og får

$f'(1,5) = \frac{e^{1,50000001} - e^{1,5}}{0,00000001} = 4,481688$

Prøver å se når det går til helvedet.

$h = 0,000000001$

$f'(1,5) = 4,48168$

$h = 0,0000000001$

$f'(1,5) = 4,48164$

$h = 0,00000000001$

$f'(1,5) = 4,481825$

$h = 0,000000000001$

$f'(1,5) = 4,47792$

Video ble kun delvis riktig. Jeg løste utregningene i CAS, men Jeah (vi jobber sammen) løste oppgavene med kalkulator. Hun fikk litt andre svar enn meg og kunne se at når

$h = 0,0000000000001$

så ble $f(1,5) = 0$, noe det gikk ved 8 desimaler her.

For meg skjedde det samme når

$h = 0,00000000000000001$

2

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

Antar at

$f(x) = e^x$

Bruker fortsatt premissen $f(1,5)$

2

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

Antar at

$f(x) = e^x$

Bruker derivasjonsprogrammet $f(1,5)$

$f'(1,5) = 4,48167$

Tester først med $h = 0,1$

Får da:

$$f'(1,5) = \frac{e^{1,6} - e^{1,4}}{0,1 \cdot 2} = 4,48916$$

Avviker på $7,4 \cdot 10^{-3}$

Prøver så med $h = 0,01$ og får da:

$$f'(1,5) = \frac{e^{1,501} - e^{1,499}}{2 \cdot 0,01} = 4,48176$$ (begynner å bli en god tilnærming)

Avviker på ca $6 \cdot 10^{-5}$

Prøver med $h = 0,001$ og får:

$$f'(1,5) = \frac{e^{1,501} - e^{1,499}}{2 \cdot 0,001} = 4,48168$$ (går nærmere)

Avviker på $2 \cdot 10^{-5}$

$h = 0,0001$

$f'(1,5) = 4,481689$

Avviker på $1,1 \cdot 10^{-5}$

$h = 0,00001$

$f'(1,5) = 4,4816890$

Avviker på $1,7 \cdot 10^{-5}$

$h = 0,000001$

$f'(1,5) = 4,4816890$

Samme avvik

$h = 0{,}0000001$
$f'(1{,}5) = 4{,}4816889$
Avvik $1{,}77 \cdot 10^{-5}$

$h = 0{,}00000001$
$f'(1{,}5) = 4{,}4816886$
Avvik $1{,}74 \cdot 10^{-5}$

$h = 0{,}000000001$
$f'(1{,}5) = 4{,}4816772$
Avvik $2{,}28 \cdot 10^{-5}$ (begynner å bli store avvik)

$h = 0{,}0000000001$
$f'(1{,}5) = 4{,}4815351$
Avvik $1{,}649 \cdot 10^{-4}$

$h = 0{,}00000000001$
$f'(1{,}5) = 4{,}4821035$
Avvik $9{,}035 \cdot 10^{-4}$

$h = 0{,}000000000001$
$f'(1{,}5) = 4{,}4622083$

Avvik 0,0194917 (stort avvik)

$h = 0{,}0000000000001$
$f'(1{,}5) = 4{,}5474735$
Avvik: 0,0657735

$h = 0{,}00000000000001$
$f'(1{,}5) = 5{,}6843418$

$h = 0{,}000000000000001$
$f'(1{,}5) = 28{,}421709$

Her har det skjedd noe rart jeg ikke skjønner

$h = 0{,}0000000000000001$
$f'(1{,}5) = 28{,}421709$

$h = 0{,}00000000000000001$
$f'(1{,}5) = 284{,}21709$

$h = 0{,}000000000000000001$
$f'(1{,}5) = 2842{,}17$

Her har det skjedd noe rart jeg ikke skjønner. Hvorfor har skjedde høyde utregningen i CAS. Virker som at svaret bare øker med 10 hver gang. Leah løste oppgaven i Pythen og fikk at når $h = 0{,}0000000000000001$ ble svaret 0.0.

- Vi får bedre en bedre tilnærming ved å bruke formelen

$$f''(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Dette kan forklares ved hjelp av Taylor og feilledd.
I forrige oppgave brukte vi formelen:

$$f''(x) \approx \frac{f(x+h) - f(x)}{h}$$

Taylorutvikler vi denne får vi.

$$f'(x) = f'(x) + \frac{f''(x)}{2}h + \frac{f'''(x)}{6}h^2 + \dots$$

Her blir den dominerende feilleddet

$\frac{f''(x)}{2} \cdot h$ (feilen er proporsjonal med h. Så hvis vi reduserer h med 10, vil feilen også bli omtrent 10 ganger mindre)

Hvis vi gjør det samme med formelen

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2 \cdot h}$$

og Taylor utvikler denne får vi.

$$f'(x) \approx f'(x) + \frac{f'''(x)}{6} \cdot h^2$$

Der er feilen $\frac{f'''(x)}{6} \cdot h^2$

Feilen er proporsjonal med h^2.
Så hvis vi reduserer h med 10, vil feilen bli 100 ganger mindre.

Dermed gir $f'(x) \approx \frac{f(x+h) - f(x-h)}{2 \cdot h}$ en bedre tilnærming

3

Siden mine utregninger i CAS ble litt vare bestemte vi oss med å bruke Python videre i utregningene

Koden vi brukte var ikke ut og vi endrer størrelsen på h for hver runde.

Dermed gir $f'(x) \approx \frac{f(x+h)-f(x-h)}{2 \cdot h}$ en bedre tilnærming

3

Siden mine utregninger i CAS ble litt vare bestemte vi oss med å bruke Python videre i utregningene

Koden vi brukte var slik og vi endrer størrelsen på h for hver runde.

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

h=0.1
x=1.5

f_der=(np.exp(x-2*h)-8*np.exp(x-h)+8*np.exp(x+h)-np.exp(x+2*h))/(12*h)

print(f_der)
```

$h = 0{,}1$

$f'(1{,}5) = 4{,}481716\ldots$ (Veldig nærme)

$h = 0{,}01$

$f'(1{,}5) = 4{,}487689$

$h = 0{,}001$

$f'(1{,}5) = 4{,}4816890$

$h = 0{,}0001$

$f'(1{,}5) = 4{,}4816890$ (Ikke lenger noen forskjell i desimalene etter dette)

$h = 0{,}00001$

$f'(1{,}5) = 4{,}4816890$

$h = 0{,}000001$

$f'(1{,}5) = 4{,}4816890$

$h = 0{,}0000001$

$f'(1{,}5) = 4{,}4816890$

$h = 0{,}00000001$

$f'(1{,}5) = 4{,}4816889$

$h = 0{,}000000001$

$h = 0{,}0000001$
$f'(1{,}5) = 4{,}4816890$

$h = 0{,}00000001$
$f'(1{,}5) = 4{,}4816889$

$h = 0{,}000000001$
$f'(1{,}5) = 4{,}4816893$

$h = 0{,}0000000001$
$f'(1{,}5) = 4{,}481692$

$h = 0{,}00000000001$
$f'(1{,}5) = 4{,}481718$ (nærmer det seg ikke da)

$h = 0{,}000000000001$ (her har det muligens skjedd)
$f'(1{,}5) = 4{,}482562$

$h = 0{,}0000000000001$
$f'(1{,}5) = 4{,}480119$

$h = 0{,}00000000000001$
$f'(1{,}5) = 4{,}485301$

$h = 0{,}000000000000001$ (her endte det)
$f'(1{,}5) = 5{,}707025$

$h = 0{,}0000000000000001$ (her gikk det feil skjønner)
$f'(1{,}5) = -9{,}480297$

[4] Vi har gått fremover:

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

Denne kan skrives om til (vi er interessert i temperaturen i neste tidspunkt):

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

Dette betyr at temperaturen på tidspunktet $j+1$ beregnes ved:

- Nåværende temperatur: $u_{i,j}$
- "Naboene" $u_{i+1,j}$ og $u_{i-1,j}$

h = romsteg (tettheten langs x-aksen)

Dette betyr at temperaturen på tidspunktet $j+1$ beskrives av:

- Nåværende temperatur: $u_{i,j}$
- "Naboene" $u_{i+1,j}$ og $u_{i-1,j}$

h = romsteg (tetthet langs x-aksen)

k = tidssteg (tetthet langs t-aksen)

Initialbetingelser: $u(x,0) = \sin(x)$ is temp ved $t=0$

Randbetingelsene: $u(0,t)=0$ $u(1,t)=0$ is temp ved kantene

For stabilitet må $\frac{k}{h^2} \leq 0{,}5$

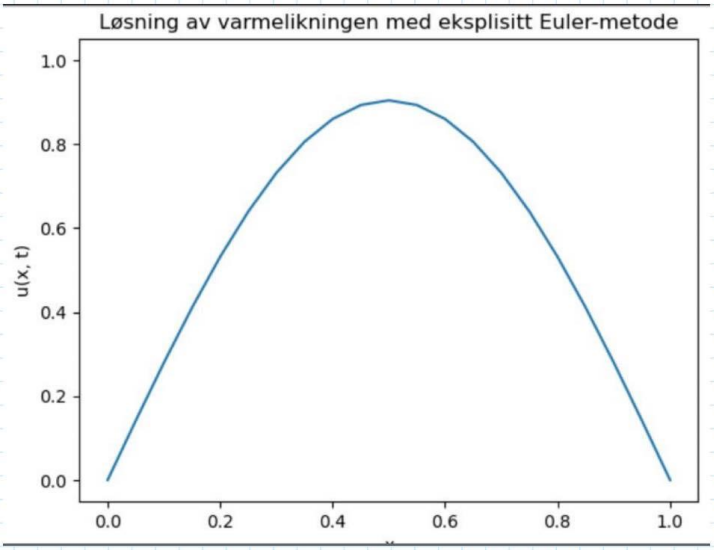
Løser oppgaven i Python.

is ønsker å lage en animasjon.

Prøvde med mange forskjellige koder, med litt hjelp fikk vi denne koden

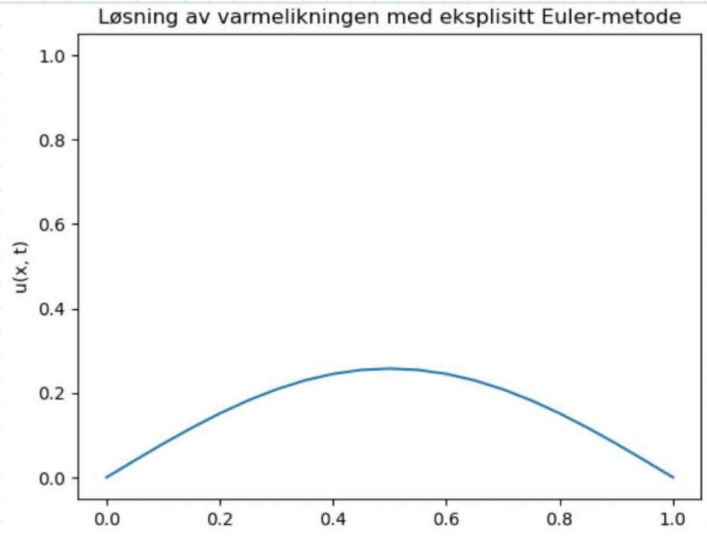
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Parametre
L = 1.0          # Lengde på stangen
T_max = 0.2      # Maksimal tid
h = 0.05         # Gitteravstand i x-retningen
k = 0.005        # Gitteravstand i t-retningen
x = np.arange(0, L + h, h)  # Romgitterpunkter
n = len(x)       # Antall punkter i x
t = np.arange(0, T_max + k, k)  # Tidsgitterpunkter
m = len(t)       # Antall tidspunkter

# Initialbetingelse u(x, 0) = sin(x)
u = np.sin(np.pi * x)

# Animasjon
fig, ax = plt.subplots()
line, = ax.plot(x, u)

def update(frame):
    global u
    u_new = np.copy(u)
    for i in range(1, n - 1):
        u_new[i] = u[i] + (k / h**2) * (u[i + 1] - 2 * u[i] + u[i - 1])
    u = np.copy(u_new)
    line.set_ydata(u)  # Oppdaterer kurven
    return line,
```

```
ani = FuncAnimation(fig, update, frames=m, interval=50, blit=True)
plt.xlabel('x')
plt.ylabel('u(x, t)')
plt.title('Løsning av varmelikningen med eksplisitt Euler-metode')
plt.show()
```



Prøvde oss fram med litt forskjellige verdier for både k og h.
Fant ikke annet ut enn hvis h er mye mindre enn k så kræsjer hele koden.

Hvis jeg satte k = h skjedde det ikke noe endringer ved noen verdier, men ved f.eks h = k = 0,05 ble kjøremiljøet klarer



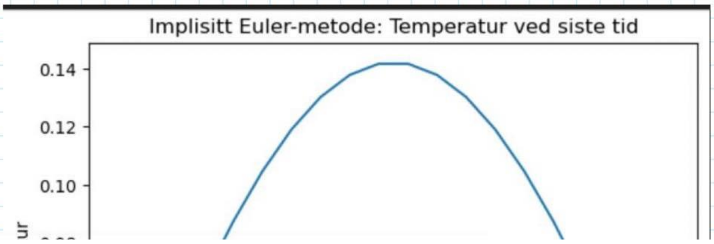
Ved andre kombinasjoner ble grafen den samme. Diskuterte en god stund sammen, men kom ikke frem til hvorfor det ble sånn. Kan være koden vår som er feil, men vi har prøvd.

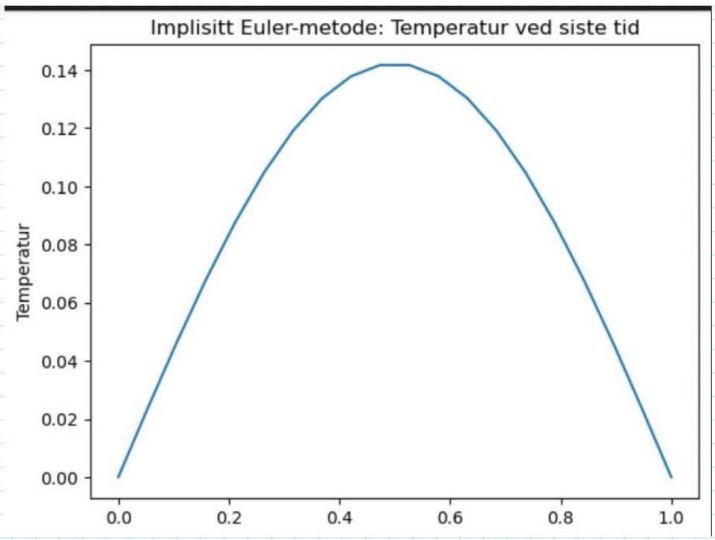
Eksplisitt Euler: Den er enkel, men kan være ustabil hvis tidsstegene er for store.
↳ Dette er fordi denne metoden bruker temperaturen på hvert punkt fra forrige tidssteg for å beregne temperaturen ved neste tidssteg
↳ Blir ustabil fordi stabiliteten avhenger av forholdet mellom k og h.

5

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

# Parametre
L = 1  # Lengden på stangen
T = 0.2  # Total tid
n = 20  # Antall punkter i x-retningen
m = 100  # Antall tidssteg
h = L / (n-1)  # Gitteravstand i x-retningen
k = T / m  # Tidssteg

# Initialisering
x = np.linspace(0, L, n)
u = np.sin(np.pi * x)  # Initialbetingelse u(x,0) = sin(pi * x)

# Funksjon for implisitt Euler-metode
def implicit_euler(u, h, k, n, m):
    r = k / (h**2)  # Størrelsen på raten
    A = np.diag((1 + 2*r)*np.ones(n-2)) - np.diag(r*np.ones(n-3), -1) - np.diag(r*np.ones(n-3), 1)
```

```
    for j in range(m):
        b = u[1:-1]
        u[1:-1] = np.linalg.solve(A, b)
    return u

# Løsning
u_final = implicit_euler(u, h, k, n, m)

# Plot resultatet
plt.plot(x, u_final)
plt.title("Implisitt Euler-metode: Temperatur ved siste tid")
plt.xlabel("x")
plt.ylabel("Temperatur")
plt.show()
```





Implisitt formel:
Hva skjer når h reduseres:
→ Vi får flere punkter i romretningen, gir mer nøyaktig og detaljert løsning
Er også mer stabil for større k-verdier.

Hva skjer når k reduseres?
- Oppdaterer løsningen oftere, gir mer nøyaktig tidsutvikling av temperaturen.
- Men enkelte k-verdier kan være tidkrevende

Så implisitt metoden er stabil, men mer beregningskrevende.

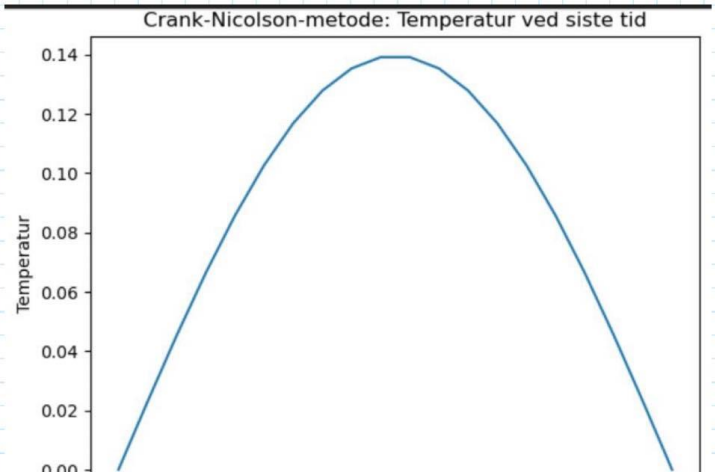
6

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

# Parametre
L = 1  # Lengden på stangen
T = 0.2  # Total tid
n = 20  # Antall punkter i x-retningen
m = 100  # Antall tidssteg
h = L / (n-1)  # Gitteravstand i x-retningen
k = T / m  # Tidssteg

# Initialisering
x = np.linspace(0, L, n)
u = np.sin(np.pi * x)  # Initialbetingelse u(x,0) = sin(pi * x)

# Funksjon for Crank-Nicolson-metoden
def crank_nicolson(u, h, k, n, m):
    r = k / (2*h**2)
    A = np.diag((1 + 2*r)*np.ones(n-2)) - np.diag(r*np.ones(n-3), -1) - np.diag(r*np.ones(n-3), 1)
    B = np.diag((1 - 2*r)*np.ones(n-2)) + np.diag(r*np.ones(n-3), -1) + np.diag(r*np.ones(n-3), 1)
```

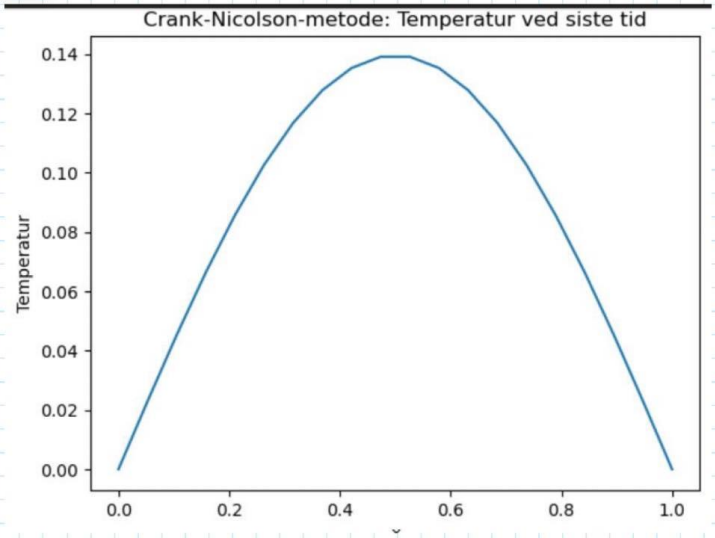
```python
    for j in range(m):
        b = np.dot(B, u[1:-1])
        u[1:-1] = np.linalg.solve(A, b)
    return u
```

[6]

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

# Parametre
L = 1  # Lengden på stangen
T = 0.2  # Total tid
n = 20  # Antall punkter i x-retningen
m = 100  # Antall tidssteg
h = L / (n-1)  # Gitteravstand i x-retningen
k = T / m  # Tidssteg

# Initialisering
x = np.linspace(0, L, n)
u = np.sin(np.pi * x)  # Initialbetingelse u(x,0) = sin(pi * x)

# Funksjon for Crank-Nicolson-metoden
def crank_nicolson(u, h, k, n, m):
    r = k / (2*h**2)
    A = np.diag((1 + 2*r)*np.ones(n-2)) - np.diag(r*np.ones(n-3), -1) - np.diag(r*np.ones(n-3), 1)
    B = np.diag((1 - 2*r)*np.ones(n-2)) + np.diag(r*np.ones(n-3), -1) + np.diag(r*np.ones(n-3), 1)
    for j in range(m):
        b = np.dot(B, u[1:-1])
        u[1:-1] = np.linalg.solve(A, b)
    return u

# Løsning
u_final = crank_nicolson(u, h, k, n, m)

# Plot resultatet
plt.plot(x, u_final)
plt.title("Crank-Nicolson-metode: Temperatur ved siste tid")
plt.xlabel("x")
plt.ylabel("Temperatur")
plt.show()
```



```
show()
```



Hva skjer når h reduseres?
- Reduksjon i h betyr flere punkter i romretningen og gir også detaljert og nøyaktig løsning.
- Også stabil

Reduksjon i k:
- Gir mer detaljert tidsoppløsning av løsningen
  Gir en mer presis beskrivelse av hvordan temperaturen endrer seg over tid.

Denne metoden gir god balanse mellom nøyaktighet og stabilitet og er derfor ofte foretrukket. Bruker informasjon fra både nåværdi og neste tidssteg og er mer nøyaktig.