TMA4106 - Oblig - Ronja Bakketeig

1

```
from math import e

def f(x):
    return e**x

def h_derivative(x, h):
    return (f(x + h) - f(x)) / h

h = 0.01
x = 1.5

print(h_derivative(x, h))
```

```
4.5041723976187775
```

$h = 0{,}01 \Rightarrow f'(1{,}5) = 4{,}504$

$h = 0{,}001 \Rightarrow f'(1{,}5) = 4{,}4839$

$h = 0{,}0001 \Rightarrow f'(1{,}5) = 4{,}4819$

$h = 10^{-10} \Rightarrow f'(1{,}5) = 4{,}48168$

$h = 10^{-15} \Rightarrow f'(1{,}5) = 5{,}329$ → her går det dårleg

2.

```
from math import e

def f(x):
    return e**x

def h_derivative(x, h):
    return (f(x + h) - f(x-h)) / 2*h

h = 0.1
x = 1.5

print(h_derivative(x, h))
```

```
0.04489162287752207
```

$h = 0{,}01 \Rightarrow f'(1{,}5) = 4{,}48 \cdot 10^{-4}$

$h = 0{,}001 \Rightarrow f'(1{,}5) = 4{,}48 \cdot 10^{-6}$

$h = 0{,}0001 \Rightarrow f'(1{,}5) = 4{,}48 \cdot 10^{-8}$

$h = 10^{-16} \Rightarrow f'(1{,}5) = 0{,}0$

når vi set $h = 10^{-16}$, får vi 0 fordi talet er så lite og $(x+h) \approx (x-h)$ vil ikkje dataen klare å skilje tala lenger

Vi kan utvikle rundt x:

$$f(x+h) = f(x) + h \cdot f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \dots$$

$$f(x-h) = f(x) - h \cdot f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(x) + \dots$$

$$f'(x) = f'''(x) = e^x$$

$$\frac{f(x+h) - f(x-h)}{2h} = \frac{2hf'(x) + 2 \cdot \frac{h^3}{6} f'''(x) + \dots}{2h} = f'(x) + \frac{h^2}{3} f'''(x) + O(h^4)$$

$$= e^x + \underbrace{\frac{h^2}{3} e^x}_{\text{feilleddet}}$$

Feilleddet er proporsjonalt med h^2

3

```python
from math import e

def f(x):
    return e**x

def h_derivative(x, h):
    return (f(x - 2* h) - 8*f(x-h)+ 8*f(x+h)- f(x+ 2*h)) / 12*h

h = 0.1
x = 1.5

print(h_derivative(x, h))
```

```
0.04481674113579645
```

$h = 0{,}01 \Rightarrow 0{,}000448$

$h = 0{,}01 \Rightarrow 4{,}48 \cdot 10^{-6}$

$h = 0{,}001 \Rightarrow 4{,}48 \cdot 10^{-8}$

$h = 10^{-16} \Rightarrow -1{,}48 \cdot 10^{-32}$

4, 5, 6 :

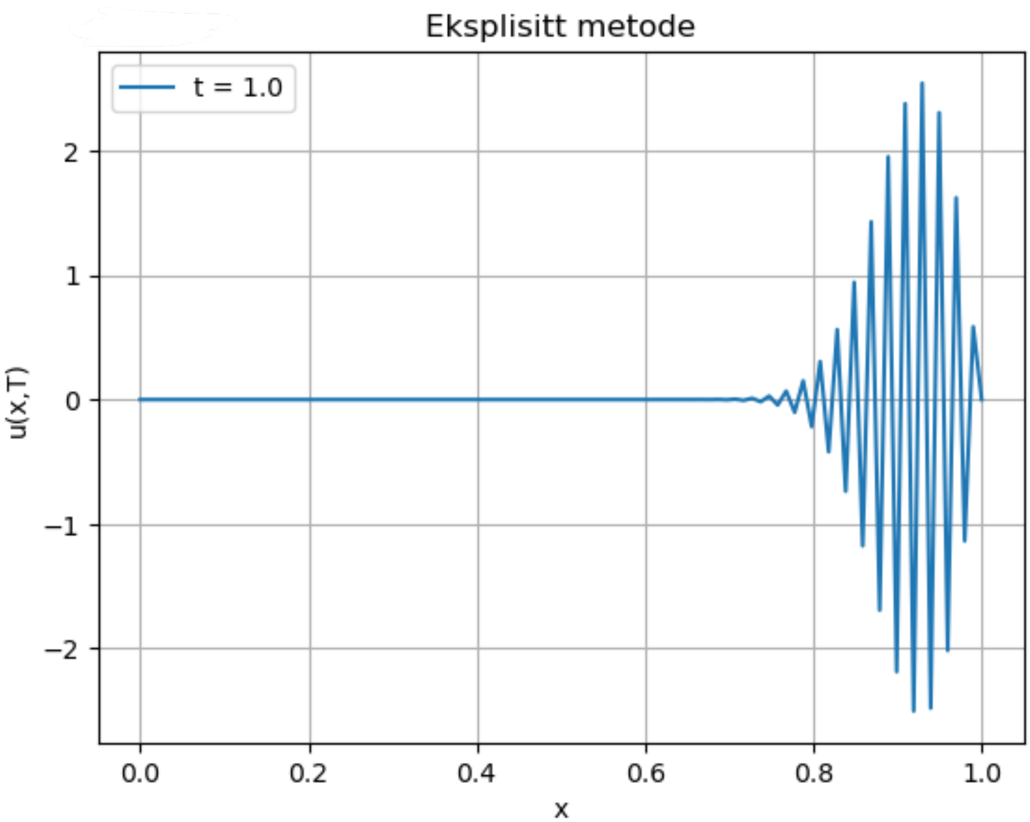
Satte h og k like for alle tre som ligg vedlagt.
ser at implisitt og crank-nicolson har svært lik oppførsel,
noko som kjem av at dei er begge ubetinga. Crank-Nicolson er
(implisitt + eksplisitt)/2, noko som gir nøyaktigheit frå den eksplisitte (ved riktige verdiar)
og den implisitte. Altså både tid og rom.

Den analytiske løysinga får heilt andre verdiar og funksjon.
f.eks skal verdien sin(1) = 0, men funksjonen gir 0,30

Varmelikninga

4

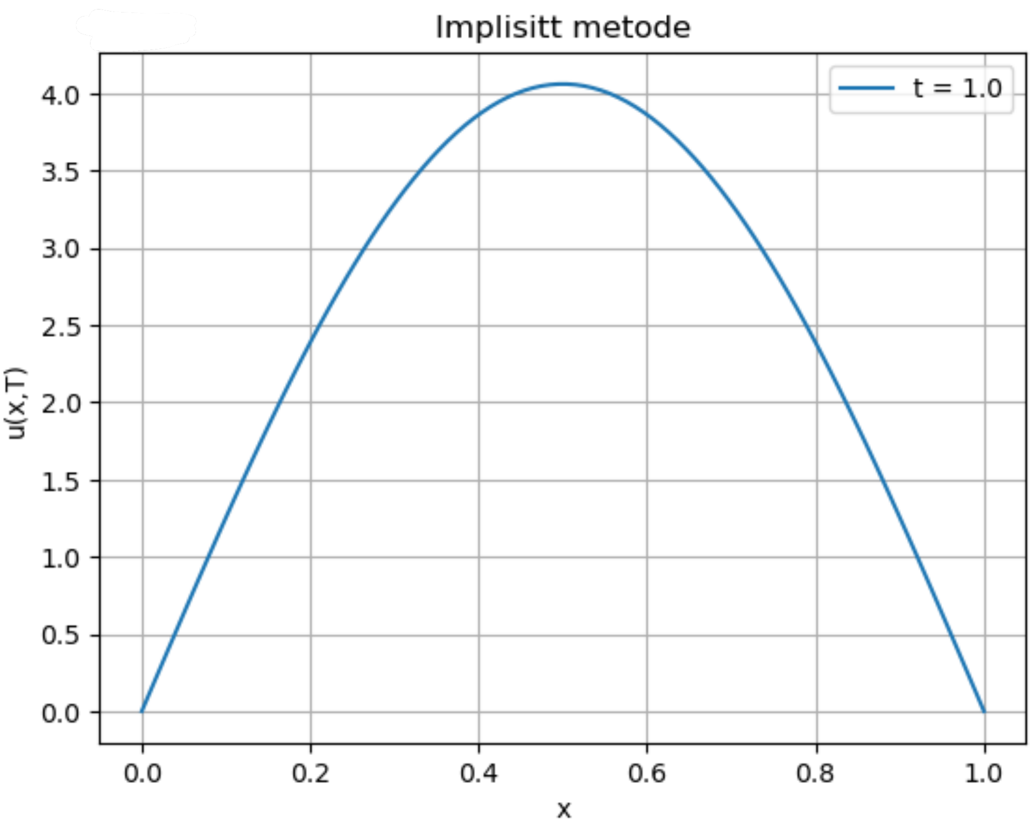
```
import numpy as np
import matplotlib.pyplot as plt


L = 1.0          # Lengde
T = 1.0          # Total tid
nx = 100         # Antall rompunkt
nt = 100         # Antall tidspunkt
h = L / nx       # Romsteg
k = T / nt       # Tidssteg
diff = k / h**2


x = np.linspace(0, L, nx)
u = np.sin(x)           # Initialkrav
u[0] = u[-1] = 0        #Randkrav
u_ny = np.copy(u)

#eksplisitte metode
for j in range(nt):
    for i in range(1, nx - 1):
        u_ny[i] = u[i] + diff * (u[i+1] - 2*u[i] + u[i-1])
    u[:] = u_ny


plt.plot(x, u, label=f't = {T}')
plt.xlabel('x')
plt.ylabel('u(x,T)')
plt.title('Eksplisitt metode')
plt.grid(True)
plt.legend()
plt.show()
```



5

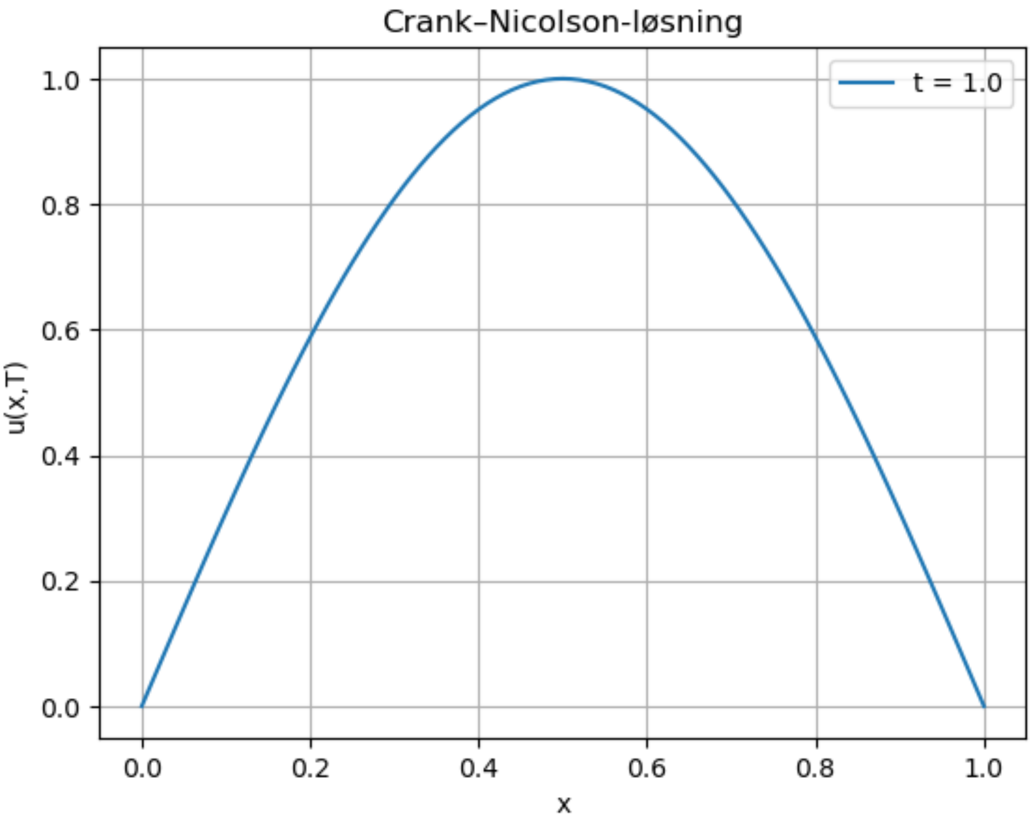
```
from scipy.linalg import solve

L = 1.0           # Lengde
T = 1.0           # Total tid
nx = 100          # Antall rompunkt
nt = 100          # Antall tidspunkt
h = L / nx        # Romsteg
k = T / nt        # Tidssteg
diff = k / h**2


# Rom og tid
x = np.linspace(0, L, nx)
u = np.sin(x)             # Initialkrav
u[0] = u[-1] = 0          # Randkrav

# lagar ei matrise for implisitt metode
A = np.zeros((nx - 2, nx - 2))
np.fill_diagonal(A, 1 + 2*diff)
np.fill_diagonal(A[1:], -diff)
```

```python
np.fill_diagonal(A[:, 1:], -diff)

u_inni = u[1:-1]

# Må løyse for tida
for n in range(nt):
    u_inni = solve(A, u_inni)
    u[1:-1] = u_inni


plt.plot(x, u, label=f't = {T}')
plt.title("Implisitt metode")
plt.xlabel("x")
plt.ylabel("u(x,T)")
plt.grid(True)
plt.legend()
plt.show()
```



6.

```python
#Løysinga for Crank-Nilson-metoden

L = 1.0           # Lengde
T = 1.0           # Total tid
nx = 100          # Antall rompunkt
nt = 100          # Antall tidspunkt
h = L / nx        # Romsteg
k = T / nt        # Tidssteg
diff = k / h**2


x = np.linspace(0, L, nx + 1)
u = np.sin(np.pi * x)    # Initialkrav
u[0] = u[-1] = 0         # Randkrav

# Matriser
N = nx
diagonal_A = (1 + diff) * np.ones(N - 1)
andre_diagonal = -diff / 2 * np.ones(N - 2)
A = np.diag(diagonal_A) + np.diag(andre_diagonal, 1) + np.diag(andre_diagonal,␣
 ↪-1)

diagonal_B = (1 - diff) * np.ones(N - 1)
B = np.diag(diagonal_B) + np.diag((diff / 2) * np.ones(N - 2), 1) + np.
 ↪diag((diff / 2) * np.ones(N - 2), -1)

u_inni = u[1:-1]  # Indre verdiar


# Tid
for n in range(nt):
    rhs = B @ u_inni          # den eksplisitte delen
    u_inn = solve(A, rhs)     # implisitt
    u[1:-1] = u_inni


plt.plot(x, u, label=f't = {T}')
plt.title("Crank-Nicolson-løsning")
plt.xlabel("x")
plt.ylabel("u(x,T)")
plt.grid(True)
plt.legend()
plt.show()
```


Crank–Nicolson-løsning
t = 1.0
u(x,T)
x
0.0
0.2
0.4
0.6
0.8
1.0

Analytisk

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

x = np.linspace(0, 1, 100)
t = 1.0
u = np.sin(x) * np.exp(-t)

plt.plot(x, u)
plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.title("Analytisk løysing")
plt.grid(True)
plt.show()
```