

1

Jo mindre h , jo bedre approksimering (i teorien)

tester for $h = 0,01$, $h = 0,001$, $h = 0,0001$, $h = 10^{-5}$, $h = 10^{-12}$

$$f'(1,5) = \frac{e^{1,51} - e^{1,5}}{0,01} = 4,5042$$

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

$$f'(1,5) = \frac{e^{1,4999} - e^{1,5}}{0,0001} = 4,4819$$

$$f'(1,5) = \frac{e^{1,4817} - e^{1,5}}{10^{-5}} = 4,4817$$

$$f'(1,5) = \frac{e^{1,48168} - e^{1,5}}{10^{-12}} = 4,4816$$

fra 10^{-5} havner videre approksimasjoner på 4,4817

helt frem til $h = 10^{-12}$, da svaret stårer å divergere

litt ungle fra 4,4817. Som indikerer starten på

numerisk ustabilitet, siden man subtraherer to nesten helt like tall

2

vi benytter

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

$$\text{for } h = 10^{-5} \Rightarrow 4,48168$$

$$\text{for } h = 10^{-7} \Rightarrow 4,48168$$

$$\text{for } h = 10^{-9} \Rightarrow 4,48168$$

$$\text{for } h = 10^{-11} \Rightarrow 4,48170$$

$$\text{for } h = 10^{-12} \Rightarrow 4,48169$$

Denne formelen gir en bedre og mer nøyaktig approksimering med mindre avvik enn den forrige, helt ned til en viss størrelse

Når h blir veldig liten, observerer vi dog igjen samme problem som ved forrige formel

Oppførselen kan forklares ved Taylor-rekker, $f(x+h)$ og $f(x-h)$ kan utvides om x , og se at feilen er proporsjonal med h^2 . Dette gjør det til en bedre approksimasjon enn forrige metode, som har feil proporsjonalt med h .

Når h minker, blir approksimasjonen bedre helt til h blir såpass lav at presisjonen går tapt på grunn av den begrensede flyttall representasjonen på datamaskiner

3

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

Kjører denne og så for ulike h

$$h = 10^{-5} \Rightarrow 7,4695$$

$$h = 10^{-7} \Rightarrow 7,4695$$

$$h = 10^{-9} \Rightarrow 7,4695$$

$$h = 10^{-11} \Rightarrow 7,4694$$

$$h = 10^{-13} \Rightarrow 7,4614$$

4

Er det $k_{h^2} < 1/2$?

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

# Parametere
L = 1
T = 0.1
h = 0.1
k = 0.004

# Berenger punkter i tid og rom
N_space = int(L / h) + 1
N_time = int(T / k) + 1

# Stabilitetsparameter
r = k / h**2

if r == 0.5:
    raise ValueError("Stability condition k/h^2 < 0.5 is not satisfied!")

def initial_condition(x):
    return np.sin(np.pi * x)

# Lage grid
x_space = np.linspace(0, L, N_space)
t_time = np.linspace(0, T, N_time)

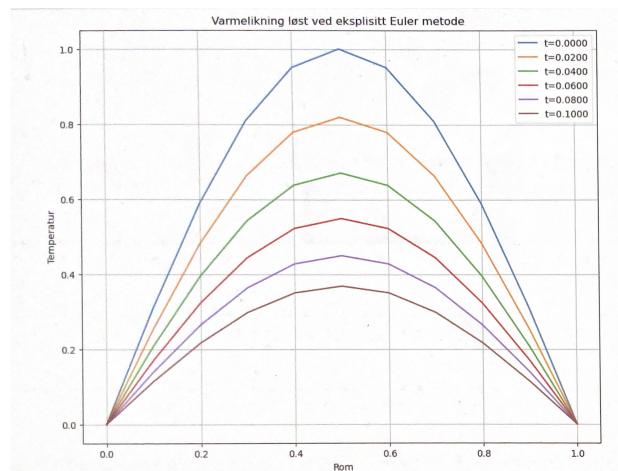
u = np.zeros((N_space, N_time))

u[:, 0] = initial_condition(x_space)

for j in range(0, N_time - 1):
    for i in range(1, N_space - 1):
        u[i, j+1] = r*u[i-1, j] + (1 - 2*r)*u[i, j] + r*u[i+1, j]

plt.figure(figsize=(10, 8))
for j in range(0, N_time, N_time // 5): # Plot some time steps
    plt.plot(x_space, u[:, j], label=f't={t_time[j]:.4f}')

plt.title('Varmelikning løst ved eksplisitt Euler metode')
plt.xlabel('Rom')
plt.ylabel('Temperatur')
plt.legend()
plt.grid(True)
plt.show()
```



5

```

import numpy as np
import matplotlib.pyplot as plt

L = 1
T = 0.1
h = 0.1
k = 0.01

N_space = int(L / h) + 1
N_time = int(T / k) + 1

r = k / h**2

def initial_condition(x):
    return np.sin(np.pi * x)

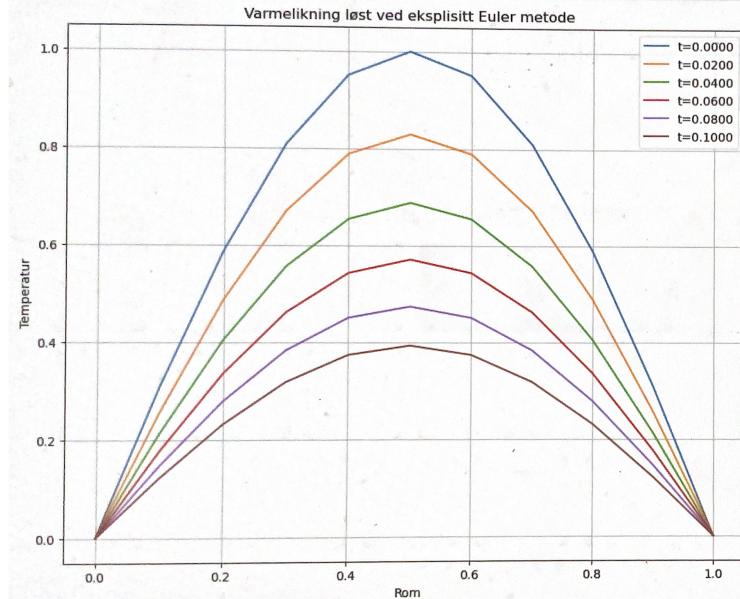
main_diagonal = (1 + 2 * r) * np.ones(N_space - 2)
off_diagonal = -r * np.ones(N_space - 3)
A = np.diag(main_diagonal) + np.diag(off_diagonal, k=1) + np.diag(off_diagonal, k=-1)

u = np.zeros((N_space, N_time))
x_space = np.linspace(0, L, N_space)
u[:, 0] = initial_condition(x_space)

for j in range(0, N_time - 1):
    b = u[1:-1, j]
    u[1:-1, j+1] = np.linalg.solve(A, b)

plt.figure(figsize=(10, 8))
for j in range(0, N_time, N_time // 5):
    plt.plot(x_space, u[:, j], label=f't={(j * k):.4f}')
plt.title('Varmelikning løst ved eksplisitt Euler metode')
plt.xlabel('Rom')
plt.ylabel('Temperatur')
plt.legend()
plt.grid(True)
plt.show()

```



6

```

import numpy as np
import matplotlib.pyplot as plt

L = 1
T = 0.1
h = 0.1
k = 0.01
N_space = int(L / h) + 1
N_time = int(T / k) + 1
r = k / (h ** 2)

A = np.diag((2 + 2*r) * np.ones(N_space - 2)) - \
    np.diag(r * np.ones(N_space - 3), 1) - \
    np.diag(r * np.ones(N_space - 3), -1)
B = np.diag((2 - 2*r) * np.ones(N_space - 2)) + \
    np.diag(r * np.ones(N_space - 3), 1) + \
    np.diag(r * np.ones(N_space - 3), -1)

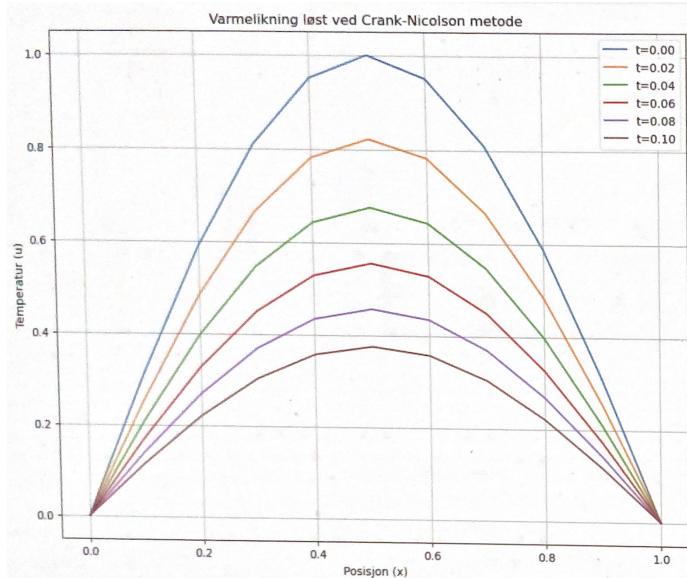
def initial_condition(x):
    return np.sin(np.pi * x)

u = np.zeros((N_space, N_time))
x_space = np.linspace(0, L, N_space)
u[:, 0] = initial_condition(x_space)

for j in range(0, N_time - 1):
    u[1:-1, j+1] = np.linalg.solve(A, B @ u[1:-1, j])

plt.figure(figsize=(10, 8))
for time_step in range(0, N_time, N_time // 5):
    plt.plot(x_space, u[:, time_step], label=f't={(time_step * k):.2f}')
plt.title('Varmelikning løst ved Crank-Nicolson metode')
plt.xlabel('Posisjon (x)')
plt.ylabel('Temperatur (u)')
plt.legend()
plt.grid(True)
plt.show()

```



Crank - Nicholson brukes fordi den har en god balanse mellom presisjon og stabilitet. Den er andordens i både tid og rom, slik at den har mindre numerisk spredning enn de første ordens eksplisitte og implisite