Jeg har valgt å gjøre standardprosjektet.

1.

Jeg tester formelen $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ for mindre og mindre verdier for h. Resultatet er i denne tabellen:

h	Tilnærmet $f'(1.5)$	Feil
10^{-1}	4.7134	$2.3174 \cdot 10^{-1}$
10^{-2}	4.5042	$2.2483 \cdot 10^{-2}$
10^{-3}	4.4839	$2.2416 \cdot 10^{-3}$
10^{-4}	4.4819	$2.2409 \cdot 10^{-4}$
10^{-5}	4.4817	$2.2409 \cdot 10^{-5}$
10^{-6}	4.4817	$2.2411 \cdot 10^{-6}$
10^{-7}	4.4817	$2.3378 \cdot 10^{-7}$
10^{-8}	4.4817	$6.0318 \cdot 10^{-9}$
10^{-9}	4.4817	$6.1569 \cdot 10^{-7}$
10^{-10}	4.4817	$5.9448 \cdot 10^{-6}$
10^{-11}	4.4817	$5.9235 \cdot 10^{-5}$
10^{-12}	4.4826	$9.4741 \cdot 10^{-4}$
10^{-13}	4.4853	$3.6119 \cdot 10^{-3}$
10^{-14}	4.5297	$4.8021 \cdot 10^{-2}$
10^{-15}	5.3291	$8.4738 \cdot 10^{-1}$
10^{-16}	0.0000	4.4817

Her kan man se at tilnærmingen er best med $h = 10^{-8}$. I hvert fall er det sånn hvis jeg har skrevet Python-koden min riktig. Etter det blir feilen større, og når $h = 10^{-16}$ eller mindre, klarer ikke datamaskinen å skille $f(x+h) - f(x)$ fra 0.

2.

Jeg gjør samme eksperiment på nytt, men med formelen $\frac{f(x+h)-f(x-h)}{2h} = f'(x)$. Resultatet er i denne tabellen:

h	Tilnærmet $f'(1.5)$	Feil
10^{-1}	4.4892	$7.4732 \cdot 10^{-3}$
10^{-2}	4.4818	$7.4695 \cdot 10^{-5}$
10^{-3}	4.4817	$7.4695 \cdot 10^{-7}$
10^{-4}	4.4817	$7.4685 \cdot 10^{-9}$
10^{-5}	4.4817	$9.6605 \cdot 10^{-11}$
10^{-6}	4.4817	$2.5867 \cdot 10^{-10}$
10^{-7}	4.4817	$2.8500 \cdot 10^{-9}$
10^{-8}	4.4817	$5.0441 \cdot 10^{-8}$
10^{-9}	4.4817	$1.7160 \cdot 10^{-7}$
10^{-10}	4.4817	$1.5039 \cdot 10^{-6}$
10^{-11}	4.4817	$1.4827 \cdot 10^{-5}$
10^{-12}	4.4822	$5.0332 \cdot 10^{-4}$
10^{-13}	4.4809	$8.2894 \cdot 10^{-4}$
10^{-14}	4.4853	$3.6119 \cdot 10^{-3}$
10^{-15}	4.8850	$4.0329 \cdot 10^{-1}$
10^{-16}	0.0000	4.4817

Her kan man se at feilen synker raskere enn i oppgave 1. Dette kan forklares med taylorrekker:

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

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

Når vi trekker disse fra hverandre, får vi $f(x+h) - f(x-h) = f'(x)2h + \frac{f'''(x)}{6}2h^3 + \cdots$.

Så deler vi på $2h$ som i formelen og får $\frac{f(x+h)-f(x-h)}{2h} = f'(x) + \frac{f'''(x)}{6}h^2 + \cdots$. Dette viser at feilen nå er proporsjonal med h^2 istedenfor h.

Etter $h = 10^{-5}$ begynner feilen å bli større, og akkurat som i oppgave 1, går det helt galt når $h = 10^{-16}$.

3.

Jeg har virkelig lyst til å slå på stortrommen, så jeg gjør eksperimentet nok en gang, men med formelen $f'(x) = \frac{f(x-2h)-8f(x-h)+8f(x+h)-f(x+2h)}{12h}$. Resultatet er i denne tabellen:

h	Tilnærmet $f'(1.5)$	Feil
10^{-1}	4.4817	$1.4957 \cdot 10^{-5}$
10^{-2}	4.4817	$1.4939 \cdot 10^{-9}$
10^{-3}	4.4817	$3.5527 \cdot 10^{-13}$
10^{-4}	4.4817	$3.8458 \cdot 10^{-13}$
10^{-5}	4.4817	$5.2195 \cdot 10^{-11}$
10^{-6}	4.4817	$3.3268 \cdot 10^{-10}$
10^{-7}	4.4817	$3.5901 \cdot 10^{-9}$
10^{-8}	4.4817	$7.2645 \cdot 10^{-8}$
10^{-9}	4.4817	$3.1963 \cdot 10^{-7}$
10^{-10}	4.4817	$2.9842 \cdot 10^{-6}$
10^{-11}	4.4817	$2.9630 \cdot 10^{-5}$
10^{-12}	4.4826	$8.7340 \cdot 10^{-4}$
10^{-13}	4.4801	$1.5691 \cdot 10^{-3}$
10^{-14}	4.4853	$3.6119 \cdot 10^{-3}$
10^{-15}	5.1070	$6.2534 \cdot 10^{-1}$
10^{-16}	-1.4803	5.9620
10^{-17}	-7.4015	11.8832

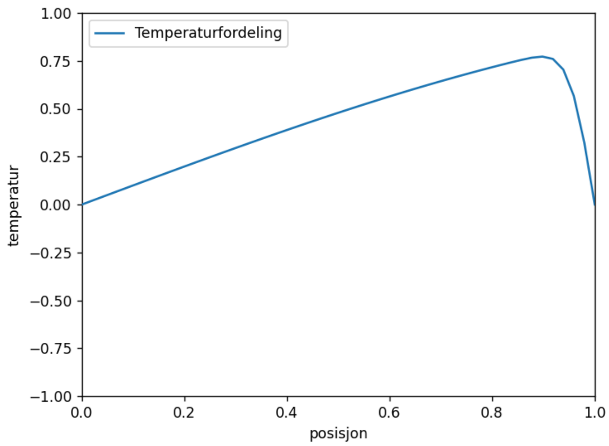
Her ser vi at feilen synker enda raskere enn for de to andre formlene. Allerede ved $h = 10^{-3}$ er feilen på sitt laveste, og deretter øker den igjen på samme måte som med de andre formlene. Denne gangen får vi ikke 0 ved $h = 10^{-16}$, men vi får faktisk negative tall.

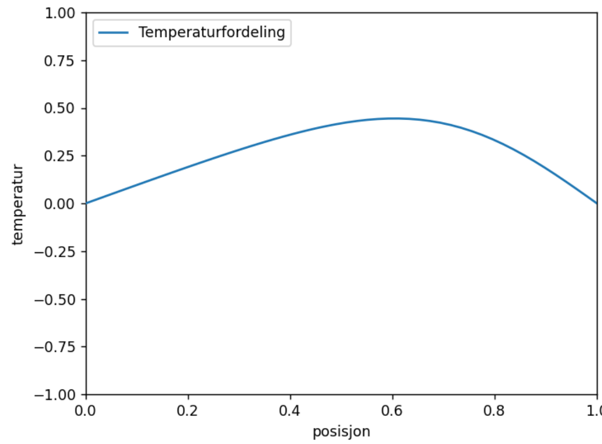
4.

Starter med å skrive om fra $\frac{u_{i,j+1}-u_{ij}}{k} = \frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}$ til $u_{i,j+1} = u_{ij} + \lambda(u_{i+1,j} - 2u_{i,j} + u_{i-1,j})$ der $\lambda = \frac{k}{h^2}$. Så var det å sette seg ned å finne ut av hvordan i alle dager man kan animere i Python. Det viste seg å være ganske greit å gjøre med matplotlib.animation.

Når jeg setter h og k lik hverandre, funker ikke koden hvis verdien er mindre enn 2. Det samme gjelder hvis jeg setter k større enn $\frac{h^2}{2}$. Med andre ord virker det som om jeg trenger at $\lambda \leq \frac{1}{2}$. Det virker også som at jo større h er i forhold til k, jo saktere synker temperaturen.

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

Nx = 50     # Gitterpunkter langs x-retningen
Nt = 1000   # Gitterpunkter langs t-retningen
h = 0.1
k = 0.001
Lambda = k / h**2

x = np.linspace(0, 1, Nx)
u = np.sin(x)
u[0] = u[-1] = 0
u_new = np.zeros_like(u)

sol = [u.copy()]

for n in range(Nt):
    for i in range(1, Nx-1):
        u_new[i] = u[i] + Lambda*(u[i+1] - 2*u[i] + u[i-1]) # Omskrivningen som er beskrevet i teksten
    u[:] = u_new[:]
    sol.append(u.copy())

fig, ax = plt.subplots()
ax.set_xlim(0, 1)
ax.set_ylim(-1, 1)
ax.set_xlabel("posisjon")
ax.set_ylabel("temperatur")
line, = ax.plot(x, sol[0], label="Temperaturfordeling")
ax.legend()

def update(frame):
    line.set_ydata(sol[frame])
    return line,

ani = animation.FuncAnimation(fig, update, frames=len(sol), interval=20, blit=True)
plt.show()
```





5.

Starter igjen med en omskrivning som er lettere å implementere i koden, men denne gangen blir det litt mer komplisert:

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

$$u_{i,j+1} = u_{ij} + \lambda\left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right)$$

$$u_{i,j+1} - \lambda u_{i+1,j+1} + 2\lambda u_{i,j+1} - \lambda u_{i-1,j+1} = u_{ij}$$

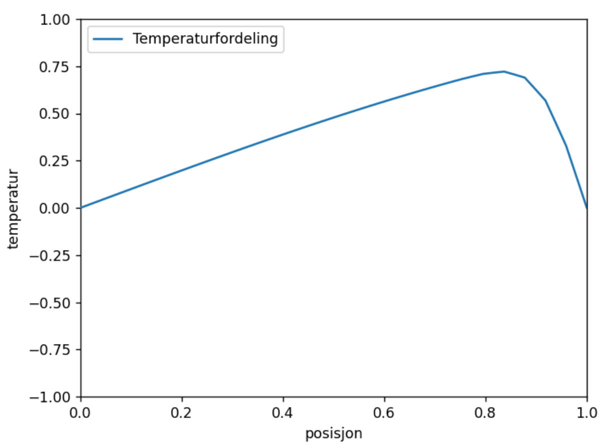
$$u_{i,j+1}(1 + 2\lambda) = u_{ij} + \lambda\left(u_{i+1,j+1} + u_{i-1,j+1}\right)$$

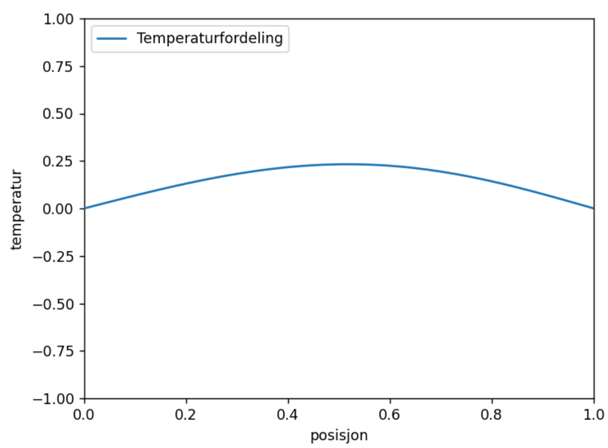
$$u_{i,j+1} = \frac{u_{ij} + \lambda\left(u_{i+1,j+1} + u_{i-1,j+1}\right)}{1 + 2\lambda}$$

λ er fortsatt $\frac{k}{h^2}$.

Når jeg setter h og k lik hverandre, funker koden for alle verdier jeg har prøvd (bortsett fra 0 selvfølgelig), og det virker ikke som om det er noen grenser for hva λ kan være med implisitt metode. På samme måte som med eksplisitt metode, vil temperaturen synke raskere jo høyere h er i forhold til k. Nå som k kan være større enn $\frac{h^2}{2}$, kan man også se at temperaturen synker raskere og raskere jo større k er i forhold til h.

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

Nx = 50     # Gitterpunkter langs x-retningen
Nt = 1000   # Gitterpunkter langs t-retningen
h = 0.1
k = 3
Lambda = k / h**2

x = np.linspace(0, 1, Nx)
u = np.sin(x)
u[0] = u[-1] = 0
sol = [u.copy()]

for n in range(Nt):
    u_new = u.copy()
    for i in range(1, Nx-1):
        u_new[i] = (u[i] + Lambda*(u[i-1] + u[i+1])) / (1 + 2*Lambda)   # Omskrivningen som er beskrevet i teksten
    u = u_new.copy()
    sol.append(u.copy())

fig, ax = plt.subplots()
ax.set_xlim(0, 1)
ax.set_ylim(-1, 1)
ax.set_xlabel("posisjon")
ax.set_ylabel("temperatur")
line, = ax.plot(x, sol[0], label="Temperaturfordeling")
ax.legend()

def update(frame):
    line.set_ydata(sol[frame])
    return line,

ani = animation.FuncAnimation(fig, update, frames=len(sol), interval=20, blit=True)
plt.show()
```





Animasjonen oppfører seg på samme måte som med eksplisitt metode

6.

Jeg starter nok en gang med å skrive om formelen:

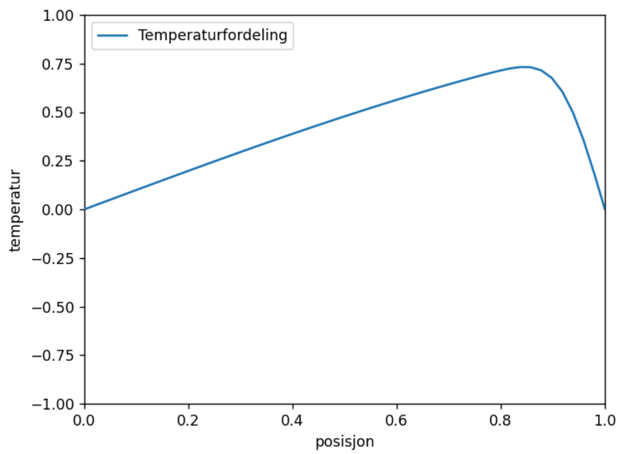
$$\frac{u_{i,j+1} - u_{ij}}{k} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{2h^2} + \frac{u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}}{2h^2}$$

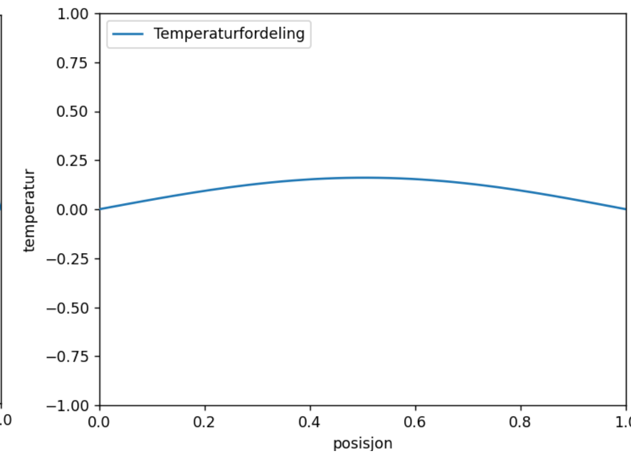
$$u_{i,j+1} = u_{ij} + \frac{\lambda}{2}\left(\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right) + \left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right)\right)$$

λ er fremdeles $\frac{k}{h^2}$.

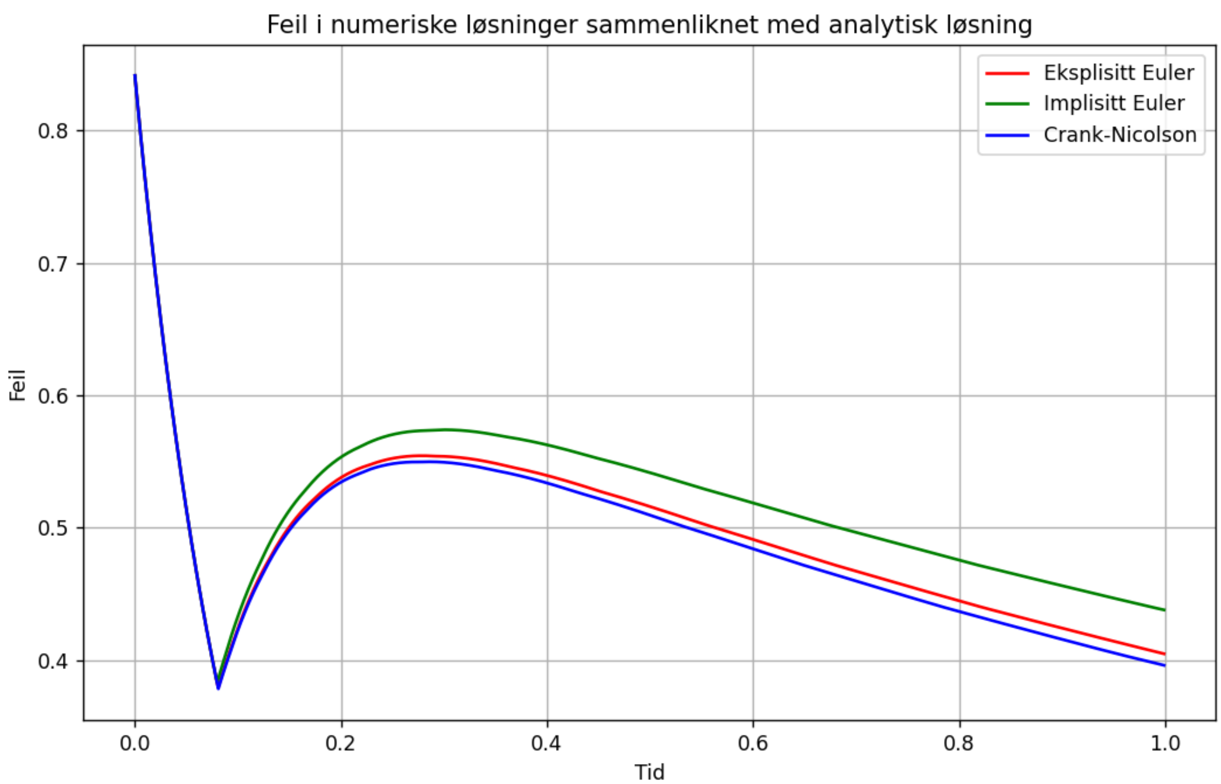
Det virker som at når jeg setter $h = k$, funker ikke koden hvis denne verdien er mindre enn 1.5. Grensen for hvor stor k kan være i forhold til h er ikke den samme her som i eksplisitt. Jeg har ikke klart å finne ut hva grensen er, men det ser ut til at den er litt mindre vrang enn eksplisitt.

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

Nx = 50     # Gitterpunkter langs x-retningen
Nt = 1000   # Gitterpunkter langs t-retningen
h = 3
k = 3
Lambda = k / h**2

x = np.linspace(0, 1, Nx)
u = np.sin(x)
u[0] = u[-1] = 0
sol = [u.copy()]

for n in range(Nt):
    u_new = u.copy()
    for i in range(1, Nx-1):
        u_new[i] = u[i] + 0.5*Lambda*(u[i+1] - 2*u[i] + u[i-1] + (u_new[i+1] - 2*u_new[i] + u_new[i-1]))
    u = u_new.copy()
    sol.append(u.copy())

fig, ax = plt.subplots()
ax.set_xlim(0, 1)
ax.set_ylim(-1, 1)
ax.set_xlabel("posisjon")
ax.set_ylabel("temperatur")
line, = ax.plot(x, sol[0], label="Temperaturfordeling")
ax.legend()

def update(frame):
    line.set_ydata(sol[frame])
    return line,

ani = animation.FuncAnimation(fig, update, frames=len(sol), interval=20, blit=True)
plt.show()
```





Den analytiske løsningen av varmelikningen er $u(x,t) = e^{-\pi^2 t} \sin x$. Her har jeg plottet feil i de tre numeriske løsningene med $h = 0.1$ og $k = 0.001$ i forhold til den analytiske løsningen:



Det virker som at Crank-Nicolson er best med tanke på at den gir lavest feil. Selv om implisitt Euler er fin å bruke med tanke på at valg av h og k ikke er særlig begrenset, viser det seg at den gir størst feil av de tre metodene.