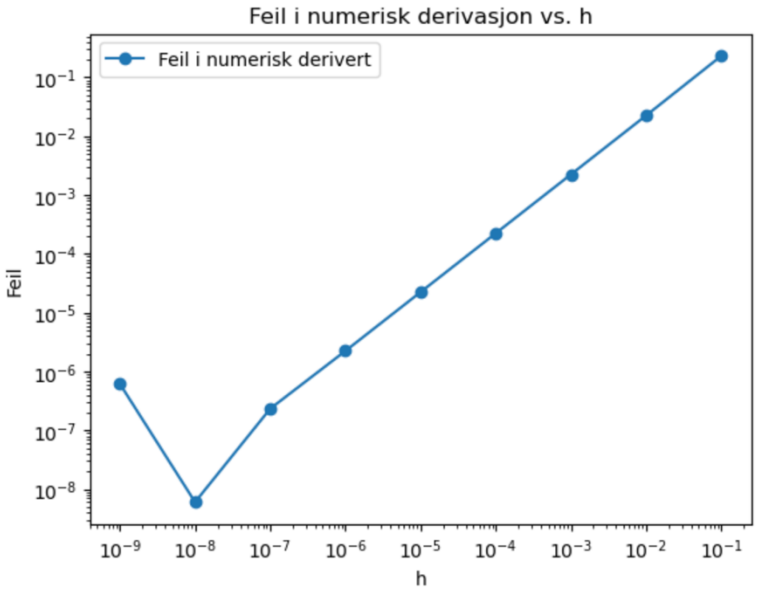
Obligatorisk oppgave i TMA4106

Skrevet av Cathrine Grønbech og Charlotte Schreiner

Kode for oppgave 1, 2 og 3 ligger under oppgave 3. Tabellverdier kommer fra python.

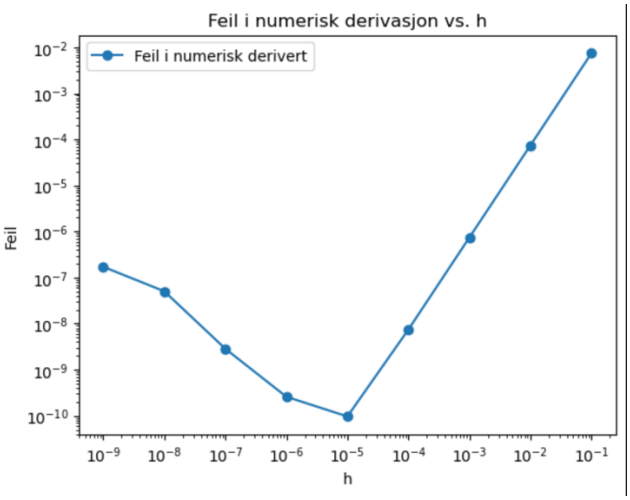
Oppgave 1:

Verdier av h	**Numerisk derivert**	**Feil**
0.10000000	4.713434	2.317445e-01
0.01000000	4.504172	2.249519e-02
0.00100000	4.489319	7.515885e-03
0.00010000	4.481711	1.627736e-03
0.00001000	4.481091	1.006990e-04
0.00000100	4.481690	5.698975e-05
0.00000010	4.481690	1.519630e-04
0.00000001	4.481690	1.519630e-03



Feilen blir mindre frem til h går mot 10^{-8} . Når h er større enn det vil feilen øke.

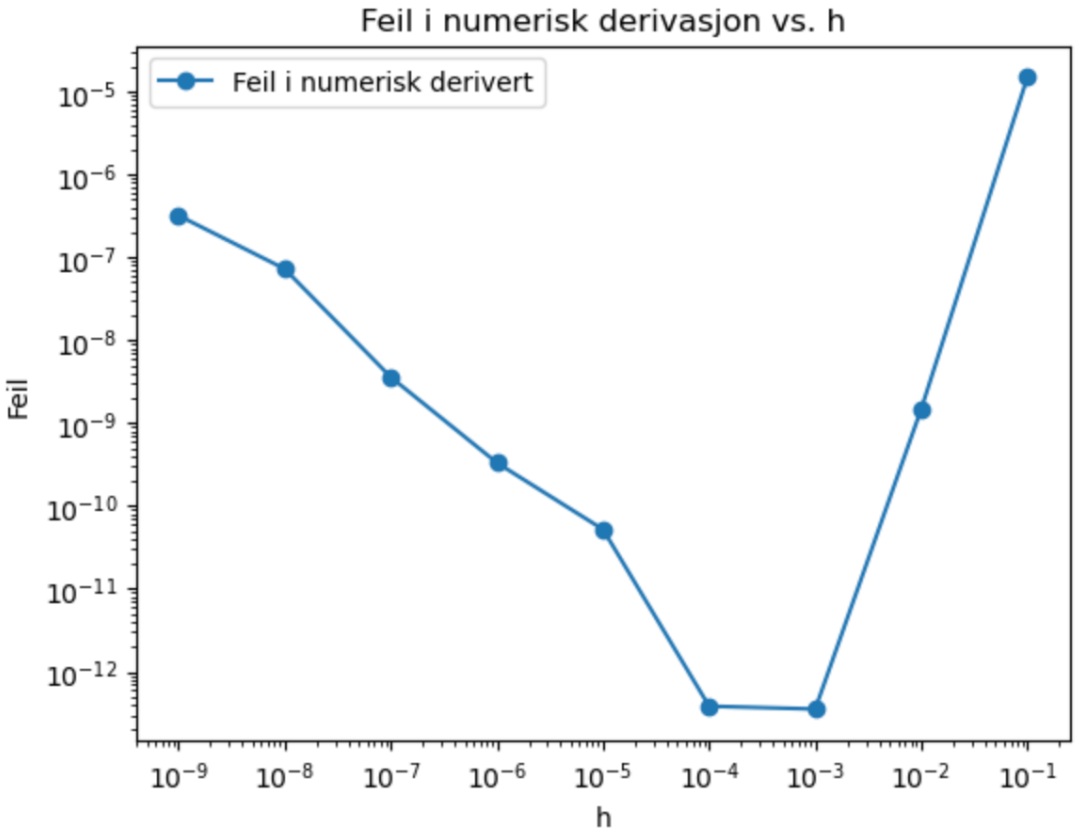
Oppgave 2:



Verdier av *h*	Numerisk derivert	Feil
0.1000000000	4.489162	7.473217e-03
0.0100000000	4.481764	7.469519e-05
0.0010000000	4.481690	7.469481e-07
0.0001000000	4.481689	7.468485e-09
0.0000100000	4.481689	9.660450e-11
0.0000010000	4.481689	2.586669e-10
0.0000001000	4.481689	2.849958e-09
0.0000000100	4.481690	5.044075e-08
0.0000000010	4.481689	1.716039e-07

Oppgave 3:

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

def f(x):
    return np.exp(x)

def df_eksakt(x):
    return np.exp(x)

x = 1.5

h_verdier = [10**(-i) for i in range(1, 10)]

feil = []
numerisk_derivasjon = []
feil_1 = []
numerisk_derivasjon_1 = []
feil_2 = []
numerisk_derivasjon_2 = []

for h in h_verdier:
    df_numerisk = (f(x + h) - f(x)) / h
    numerisk_derivasjon.append(df_numerisk)
    avvik = abs(df_numerisk - df_eksakt(x))
    feil.append(avvik)

for h in h_verdier:
    df_numerisk_1 = (f(x + h) - f(x - h)) / (2 * h)
    numerisk_derivasjon_1.append(df_numerisk_1)
    avvik_1 = abs(df_numerisk_1 - df_eksakt(x))
    feil_1.append(avvik_1)

for h in h_verdier:
    df_numerisk_2 = (f(x - 2 * h) - 8 * f(x - h) + 8 * f(x + h) - f(x + 2 * h)) / (12 * h)
    numerisk_derivasjon_2.append(df_numerisk_2)
    avvik_2 = abs(df_numerisk_2 - df_eksakt(x))
    feil_2.append(avvik_2)
```



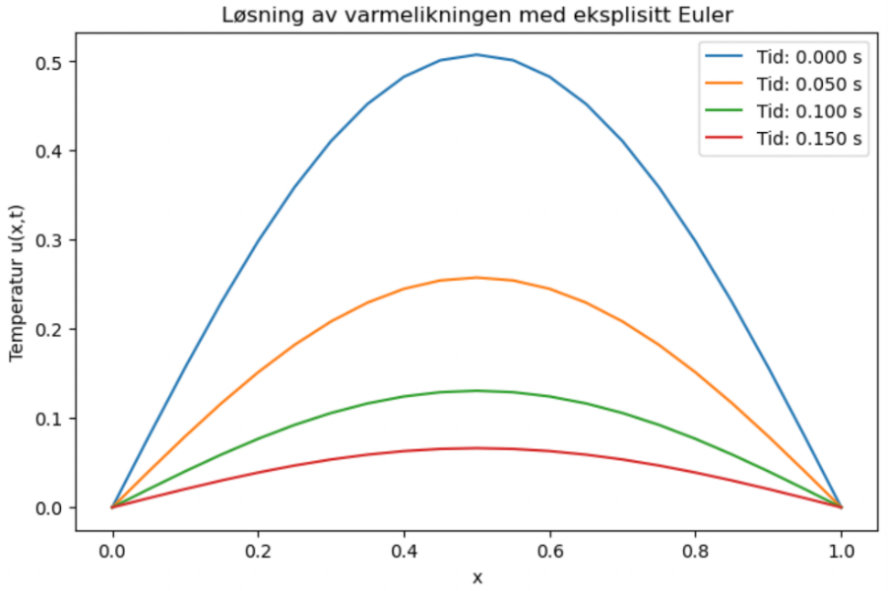
Verdier av *h*	**Numerisk derivert**	**Feil**
0.1000000000	4.481674	1.495676e-05
0.0100000000	4.481689	1.493879e-09
0.0010000000	4.481689	3.552714e-13
0.0001000000	4.481689	3.845813e-13
0.0000100000	4.481689	5.219469e-11
0.0000010000	4.481689	3.326814e-10
0.0000001000	4.481689	3.590107e-09
0.0000000100	4.481689	7.264521e-08
0.0000000010	4.481689	3.196336e-07

Oppgave 4

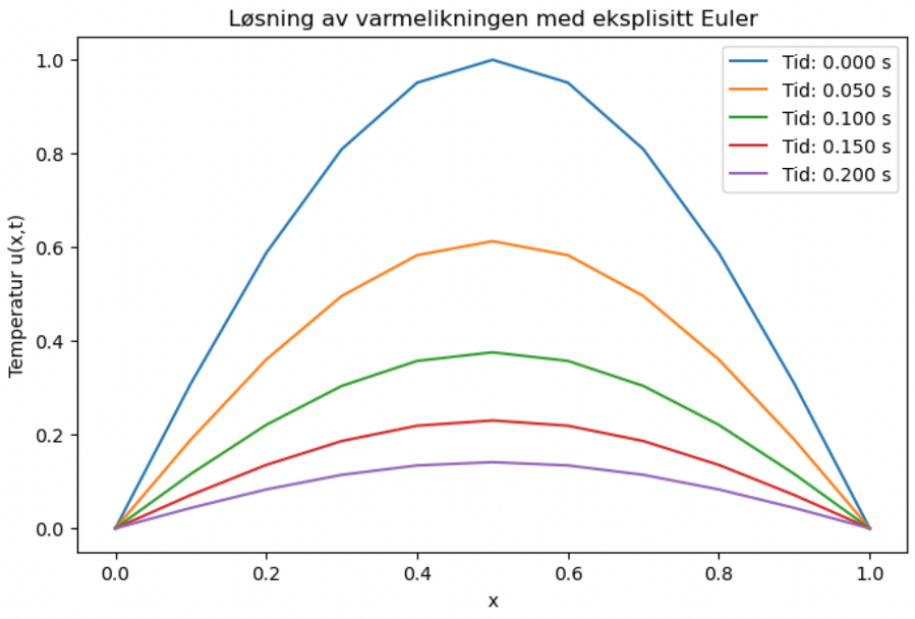
Vi starter med å bruke formelen for Eksplisitt Euler og løser for $u_{i,\,j+1}$ og får:

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

Her er $h = k$:

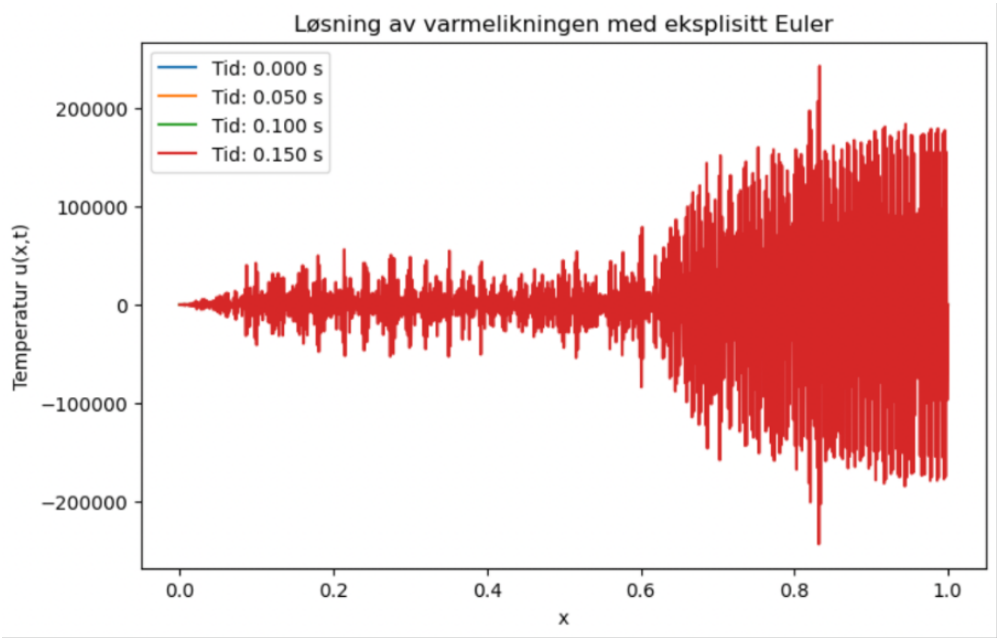


Her er $h >> k$:



Løsningen er fortsatt stabil, men figuren er mer «hakkete». Oppløsningen på grafen blir dårligere.

Her er $h << k$:



Vi testet for ulike verdier av h og k ved å endre parameterene m og n i koden vår. Vi så at løsningen blir ustabil dersom $h << k$, men fungerer dersom $h = k$ og dersom $h >> k$.

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

L = 1.0
T = 0.2
m = 10000
n = 10
h = L / n
k = T / m

x = np.linspace(0, L, n+1)
t = np.linspace(0, T, m+1)


u = np.zeros((n+1, m+1))
u[:, 0] = np.sin(np.pi*x)

for j in range(0, m):
    for i in range(1, n):
        u[i, j+1] = u[i, j] + k * (u[i+1, j] - 2*u[i, j] + u[i-1, j]) / h**2

tid_grafer = [0, m//4, m//2, 3*m//4, m]
plt.figure(figsize=(8, 5))

for j in tid_grafer:
    plt.plot(x, u[:, j], label=f'Tid: {j * k:.3f} s')

plt.xlabel('x')
plt.ylabel('Temperatur u(x,t)')
plt.title('Løsning av varmelikningen med eksplisitt Euler')
plt.legend()
plt.show()
```

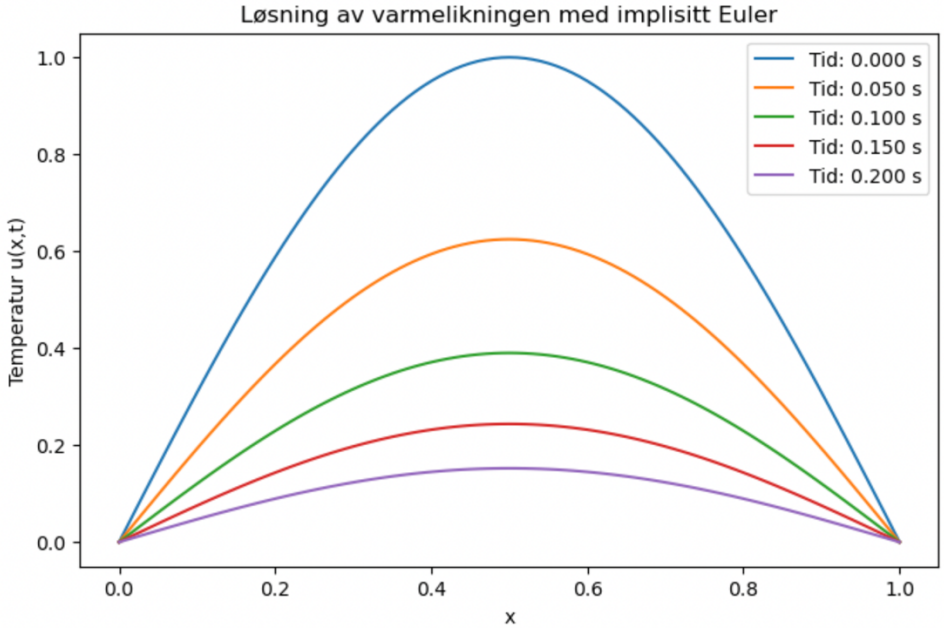
Oppgave 5

Vi løser Implisitt Euler for $u_{i,\,j+1}$

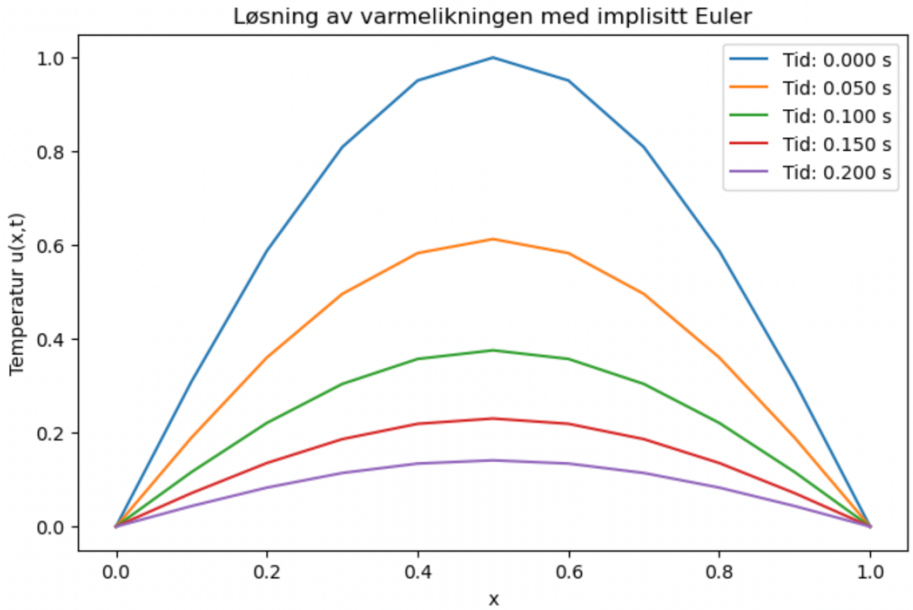
$$u_{i,\,j+1} = u_{i,j} + \frac{k}{h^2}\left(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\right)$$

Implisitt Euler kan ikke løses med en dobbel for-løkke. Vi har et likningssett som representeres som matriser i koden.

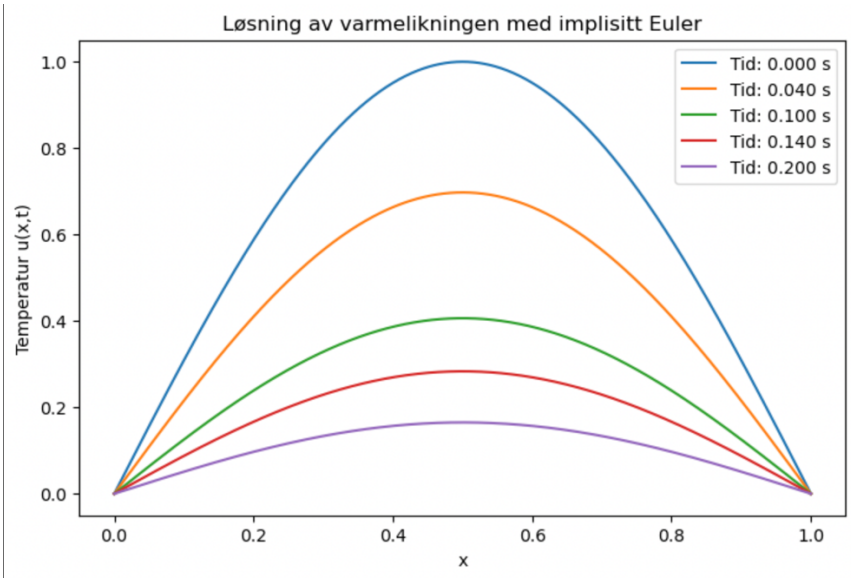
Her er $h = k$:



Her er $h >> k$:



Her er $h << k$:



Vi ser at systemet er stabilt selv for $h \ll k$. Dette gir mening fordi Implisitt Euler er alltid stabil uavhengig av verdiene for h og k. Hvis h blir veldig liten vil koden ta lang tid å kjøre fordi vi har så mange målepunkter. Derfor gjelder det å finne en kombinasjon av verdiene for h og k slik at vi får høy nok oppløsning, uten at koden tar for lang til å kjøre.

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

L = 1.0
T = 0.2
m = 10
n = 2000
h = L / n
k = T / m

x = np.linspace(0, L, n+1)
t = np.linspace(0, T, m+1)

u = np.zeros((n+1, m+1))
u[:, 0] = np.sin(np.pi * x)

S = k / h**2
A = np.diag((1 + 2*S) * np.ones(n-1)) + np.diag(-S * np.ones(n-2), 1) + np.diag(-S * np.ones(n-2), -1)

for j in range(0, m):
    u_inner = np.linalg.solve(A, u[1:n, j])
    u[1:n, j+1] = u_inner

tid_grafer = [0, m//4, m//2, 3*m//4, m]
plt.figure(figsize=(8, 5))

for j in tid_grafer:
    plt.plot(x, u[:, j], label=f'Tid: {j * k:.3f} s')

plt.xlabel('x')
plt.ylabel('Temperatur u(x,t)')
plt.title('Løsning av varmelikningen med implisitt Euler')
plt.legend()
plt.show()
```

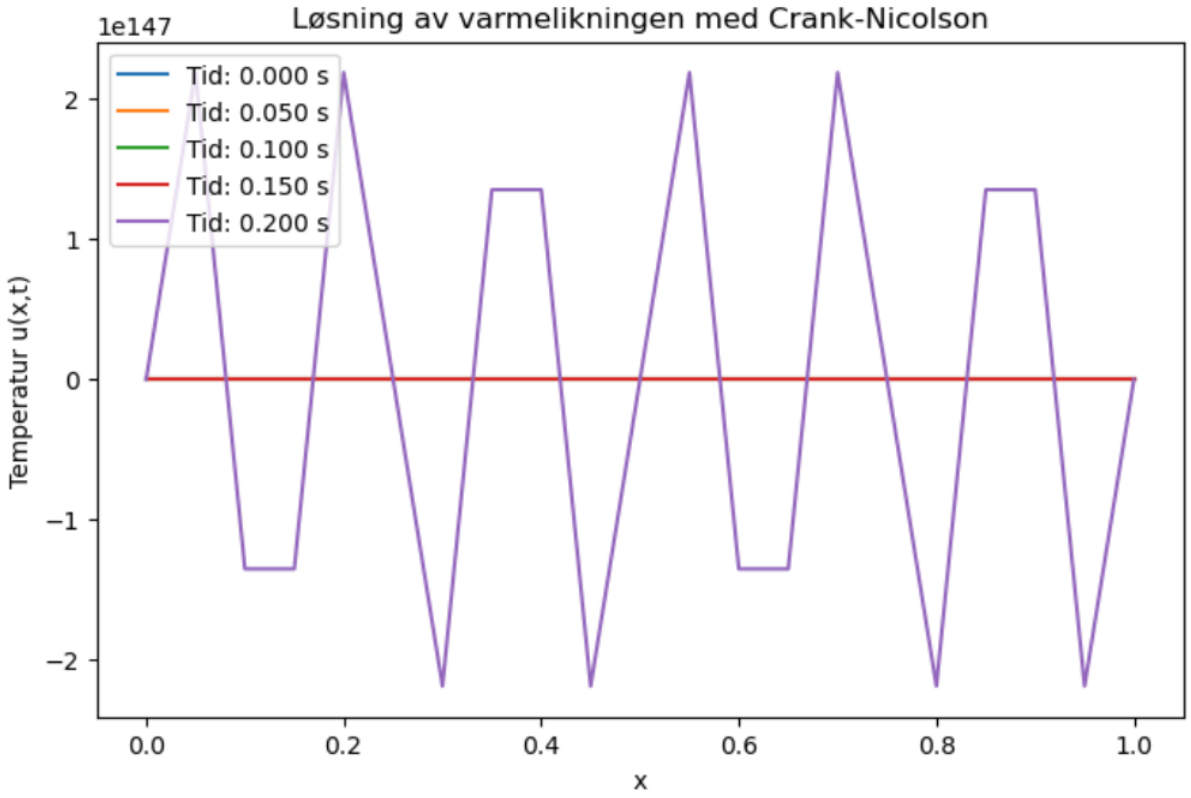
Oppgave 6:

Vi løser Crank-Nicolson på samme måte:

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

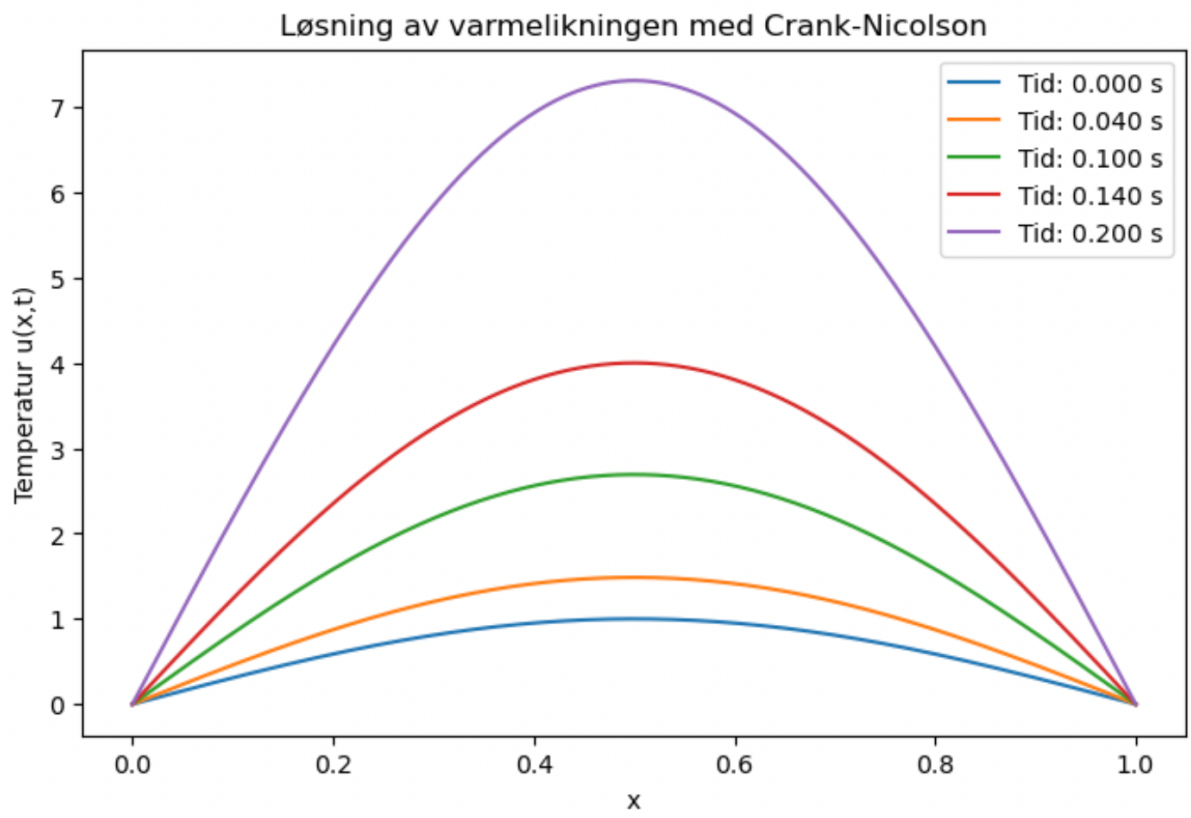
Siden Crank-Nicolson er en kombinasjon av en eksplisitt og en implisitt metode, trenger vi å også å løse denne oppgaven med matriser.

Her er $h = k$:



Dette er ikke forventet resultat, siden Crank-Nicolson-metoden i teorien skal være stabil uavhengig av verdien for S. Likevel så ser vi tydelige tegn på ustabilitet. Vi mistenker at feilen kan ligge i at matrisen $M1$, som er definert ved S, siden ved små verdier av S så fikk vi at determinanten til $M1$ ble tilnærmet lik null som muligens har ført til numeriske feil i løsningen av ligningssystemet.

Her er $h << k$:



Likevel fikk vi at systemet var stabilt dersom S ble veldig stor, som i bilde over der $S = 80000$

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve

L = 1.0
T = 0.2
m = 10
n = 2000
h = L / n
k = T / m

x = np.linspace(0, L, n+1)
t = np.linspace(0, T, m+1)

u = np.zeros((n+1, m+1))
u[:, 0] = np.sin(np.pi * x)

S = k / h**2
A = np.diag(2 * np.ones(n-1)) + np.diag(-1 * np.ones(n-2), 1) + np.diag(-1 * np.ones(n-2), -1)

M1 = np.eye(n-1) - (S/2) * A
M2 = np.eye(n-1) + (S/2) * A

print("Determinant av M1:", np.linalg.det(M1))

for j in range(0, m):
    b = np.dot(M2, u[1:n, j])
    u_inner = solve(M1, b)
    u[1:n, j+1] = u_inner

tid_grafer = [0, m//4, m//2, 3*m//4, m]
plt.figure(figsize=(8, 5))

for j in tid_grafer:
    plt.plot(x, u[:, j], label=f'Tid: {j * k:.3f} s')

plt.xlabel('x')
plt.ylabel('Temperatur u(x,t)')
plt.title('Løsning av varmelikningen med Crank-Nicolson')
plt.legend()
plt.show()
```