STANDARDPROSJEKT

1 Vi har at $f(x) = e^x$ og vil bruke tilnærminger til den deriverte gitt ved definisjonen: $f'(x) = \lim_{\to} \frac{f(x+h) - f(x)}{h}$

Riktig verdi av den deriverte:
$f'(1{,}5) = e^{1{,}5} = 4{,}4817$

Tester for lave h-verdier:

$h = 10^{-2}$: $f'(1{,}5) = \frac{e^{1{,}51} - e^{1{,}5}}{0{,}01} = 4{,}5041$ ✗ BOM

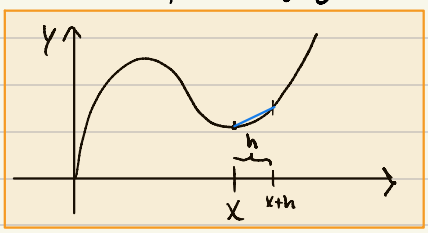
$h = 10^{-3}$: $f'(1{,}5) = \frac{e^{1{,}501} - e^{1{,}5}}{0{,}001} = 4{,}4839$ ✗ BOM

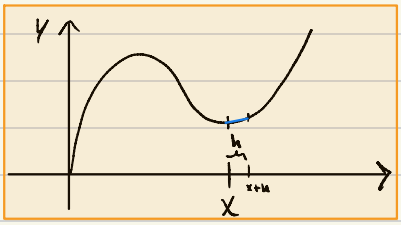
$h = 10^{-4}$: $f'(1{,}5) = \frac{e^{1{,}5001} - e^{1{,}5}}{0{,}0001} = 4{,}4819$ ✗ BOM

$h = 10^{-5}$: $f'(1{,}5) = \frac{e^{1{,}50001} - e^{1{,}5}}{0{,}00001} = 4{,}4817$ ✓ god nok ← $h = 10^{-5}$ før den "går åt skogen"

$h = 10^{-12}$: $f'(1{,}5) = \frac{e^{1{,}500001} - e^{1{,}5}}{0{,}000001} = 4{,}4826$ "går åt skogen"

Her ser man ved jo lavere h-verdi, desto bedre tilnærming. Til det går dårlig (åt skogen)
Det er fordi stigningstallet til sekanten kommer tettere på det faktiske stigningstallet til f.





Illustrasjon av at lavere h-verdi, vil gi en bedre approksimasjon av en vilkårlig funksjon

2 Gjentar nå med formelen: $f'(x) = \frac{f(x+h) - f(x-h)}{2h}$

Bruker python-kode for å løse

```
h=0.1: 4.4892 ✗
h=0.01: 4.4818 ✗
h=0.001: 4.4817✓
h=0.0001: 4.4817
```

Tok 3 forsøk

$h = 10^{-12}$ før den "går åt skogen"

PYTHON-KODE:

Users > havan > Desktop > approx.py > ...

```python
import numpy as np

x = 1.5
h = 1

for n in range(10):
    h = h / 10
    derivate_x = round((np.exp(x+h)-np.exp(x-h)) / (2*h),4)
    print(f"{h}-verdi: {derivate_x}")
```

h-verdien er mye større her før den "går åt skogen", det er på grunn av Taylorrekken

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

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

partallsderivete vil kansellere hverandre, står igjen med $h, h^3, h^5 \dots h^{alle}$ og vi deler på $2h$
Ender da med $h^2, h^4 \dots$ altså feilen er noe som er proporsjonalt med h^2 når h er veldig liten

3 Testet med formelen:

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

Brukte python til å beregne

```
4.4817
4.4817
4.4817
4.4817
4.4817
```

Her ser man det tok formelen ett forsøk for å få riktig verdi

$h = 10^{-12}$ for "gikk åt skogen"

Python-kode:

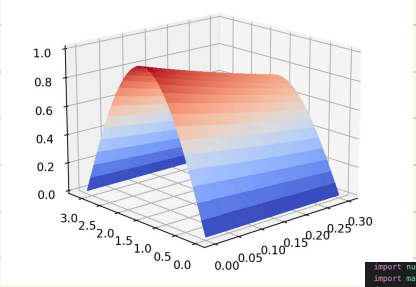
```python
import numpy as np

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

def simulate():
    x = 1.5
    h = 1
    for n in range(5):
        h = h / 10
        derivate_x = (f(x-2*h)-8*f(x-h)+8*f(x+h)-f(x+2*h))/(12*h)
        print(round(derivate_x,4))

simulate()
```

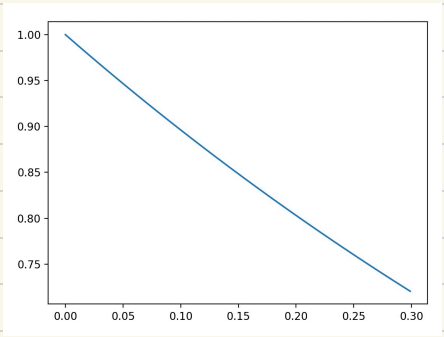
4 Varmeligningen: $u(x,t) = u''(x,t)$

med randkrav $u(0,t) = u(1,t) = 0$

og initialkrav $u(x,0) = f(x)$

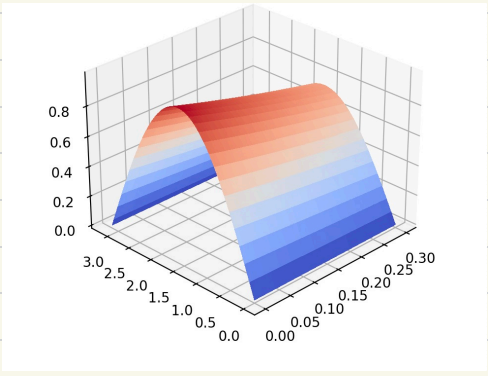
initialkrav $u(x,0) = \sin(x)$

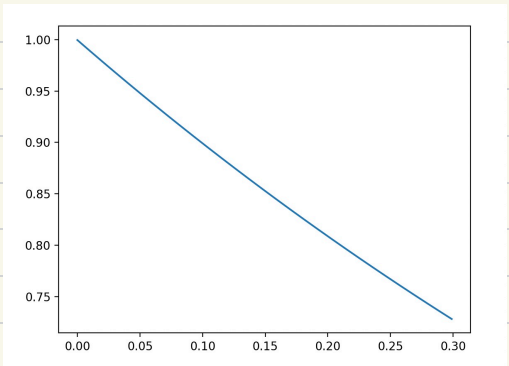





```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator

# Definer parameterne


def eksplisitt(L, t, k, h):
    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)

    x_values = np.linspace(0, L, NUM_STEPS_X)
    t_values = np.linspace(0, t, NUM_STEPS_T)

    u_values = np.zeros((NUM_STEPS_T, NUM_STEPS_X))
    u_values[0] = np.sin(x_values) #initialbetingelse

    x_values, t_values = np.meshgrid(x_values, t_values)
    # sett opp initialverdier i u_values
    # sett opp u_values[0] - arrayen

    # x_values = np.linspace(0, L, NUM_STEPS_X)

    for j in range(NUM_STEPS_T - 1):
        u_values[j][0] = 0
        u_values[j][NUM_STEPS_X - 1] = 0
        for i in range(1, NUM_STEPS_X - 1):
            u_values[j + 1][i] = u_values[j][i] + k / np.power(h, 2) * (u_values[j][i + 1] - 2 * u_values[j][i] + u_values[j][i - 1])
    return t_values, x_values, u_values


if __name__ == "__main__":
    L = np.pi
    t = 0.3
    k = 0.001
    h = 0.1
    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)
    t_values, x_values, u_values = eksplisitt(L, t, k, h)
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

    surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()
```

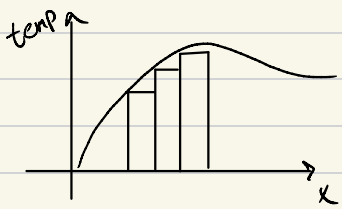
5



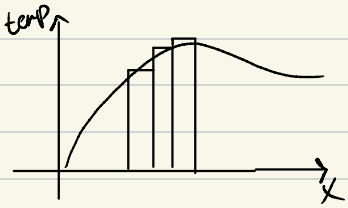



```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator

# Definer parameterne

def implisitt(L, t, k, h):
    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)

    x_values = np.linspace(0, L, NUM_STEPS_X)
    t_values = np.linspace(0, t, NUM_STEPS_T)

    u_values = np.zeros((NUM_STEPS_T, NUM_STEPS_X))
    u_values[0] = np.sin(x_values) #initialbetingelse

    x_values, t_values = np.meshgrid(x_values, t_values)
    # sett opp initialverdier i u_values
    # sett opp u_values[0] - arrayen

    # x_values = np.linspace(0, L, NUM_STEPS_X)

    for j in range(NUM_STEPS_T - 1):
        # Randkrav
        u_values[j+1][0] = 0
        u_values[j+1][NUM_STEPS_X - 1] = 0

        # Fikspunkt iterasjon
        u_values[j + 1] = np.array(u_values[j])
        last_values = np.zeros(NUM_STEPS_X)

        while (np.sum(np.abs(last_values - u_values[j + 1])) > 1e-8):
            last_values = np.array(u_values[j + 1])
            for i in range(1, NUM_STEPS_X - 1):
                u_values[j + 1][i] = u_values[j][i] + k / np.power(h, 2) * (u_values[j + 1][i + 1] - 2 * u_values[j + 1][i] + u_values[j+1][i - 1])
    return t_values, x_values, u_values

if __name__ == "__main__":
    L = np.pi
    t = 0.3
    k = 0.001
    h = 0.05
    NUM_STEPS_X = int(L / h)

    t_values, x_values, u_values = implisitt(L, t, k, h)

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()

    plt.plot(np.arange(0, t, k), u_values[:, NUM_STEPS_X // 2])
    plt.show()
```

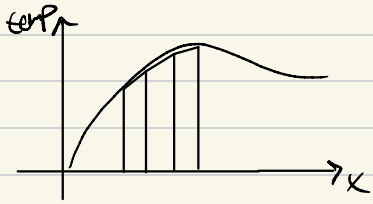
6 eksplisitt



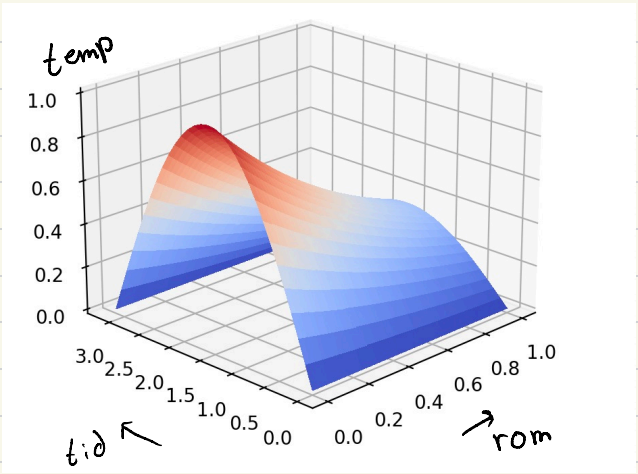
implisitt



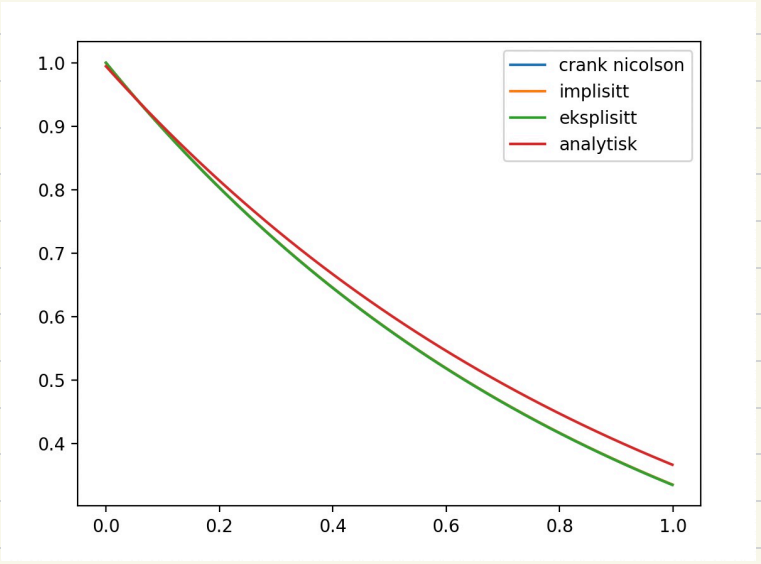
Crank-Nicolson



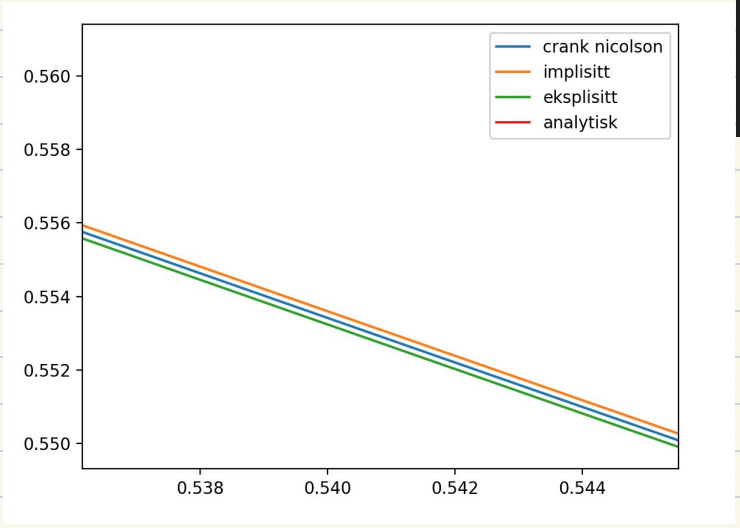
Vi har Crank-Nicolson da den er mer nøyaktig, den summerer halvparten av eksplisitt og halvparten av implisitt.




```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator

# Definer parameterne
def crank_nicolson(L, t, k, h):
    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)

    x_values = np.linspace(0, L, NUM_STEPS_X)
    t_values = np.linspace(0, t, NUM_STEPS_T)

    u_values = np.zeros((NUM_STEPS_T, NUM_STEPS_X))
    u_values[0] = np.sin(x_values) #initialbetingelse

    x_values, t_values = np.meshgrid(x_values, t_values)
    # sett opp initialverdier i u_values
    # sett opp u_values[0] - arrayen

    # x_values = np.linspace(0, L, NUM_STEPS_X)
    for j in range(NUM_STEPS_T - 1):
        u_values[j+1][0] = 0
        u_values[j+1][NUM_STEPS_X - 1] = 0

        explicit = np.array(u_values[j])

        for i in range(1, NUM_STEPS_X - 1):
            explicit[i] = u_values[j][i] + k / np.power(h, 2) * (u_values[j][i + 1] - 2 * u_values[j][i] + u_values[j][i - 1])

        last_values = np.zeros(NUM_STEPS_X)

        implicit = np.array(u_values[j])

        # Fikspunktiterasjon for å finne implisitt
        while (np.sum(np.abs(last_values - implicit)) > 1e-5):
            last_values = np.array(implicit)
            for i in range(1, NUM_STEPS_X - 1):
                implicit[i] = u_values[j][i]  + k / np.power(h, 2) * (implicit[i + 1] - 2 * implicit[i] + implicit[i - 1])
        # Crank Nicolson blir gjennomsnitt av eksplisitt og implisitt
        # Med numpy regner den ut uttrykket for hver [i]
        u_values[j+1] = (explicit + implicit) / 2
    return t_values, x_values, u_values

if __name__ == "__main__":
    L = np.pi
    t = 0.3
    k = 0.001
    h = 0.01
    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)
    print("Heiii")
    t_values, x_values, u_values = crank_nicolson(L, t, k, h)
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    # fig = plt.figure()
    # ax = fig.add_subplot(111, projection='3d')
    surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()

    plt.plot(np.arange(0, t, k), u_values[:, NUM_STEPS_X // 2])
    plt.show()
```




```python
from VarmeligningCrankNicolson import crank_nicolson
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from VarmeligningImplisitt import implisitt
from VarmeligningEksplisitt import eksplisitt


if __name__ == "__main__":
    L = np.pi
    t = 1
    k = 0.001
    h = 0.1

    NUM_STEPS_X = int(L / h)
    NUM_STEPS_T = int(t / k)
    t_values, x_values, u_values = crank_nicolson(L, t, k, h)
    t_values, x_values, u1_values = implisitt(L, t, k, h)
    t_values, x_values, u2_values = eksplisitt(L, t, k, h)
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

    surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    surf = ax.plot_surface(t_values, x_values, u1_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    surf = ax.plot_surface(t_values, x_values, u2_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()

    x_point = NUM_STEPS_X // 2
    x_val = x_values[0, x_point-1]
    analytic = np.exp(-np.arange(0, t, k)) * np.sin(x_val)

    plt.plot(np.arange(0, t, k), u_values[:, x_point], label="crank nicolson")
    plt.plot(np.arange(0, t, k), u1_values[:, x_point], label="implisitt")
    plt.plot(np.arange(0, t, k), u2_values[:, x_point], label="eksplisitt")
    plt.plot(np.arange(0, t, k), analytic, label="analytisk")
    plt.legend()
    plt.show()
```



Jeg ville trodd at implisitt skulle havnet over analytisk, men det gjorde den ikke