

STANDARDPROSJEKT

1 Vi har at $f(x) = e^x$ og vil bruke tilnærmingen til den deriverte gitt ved definisjonen: $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$

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

Testet for ulike h-verdier:

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

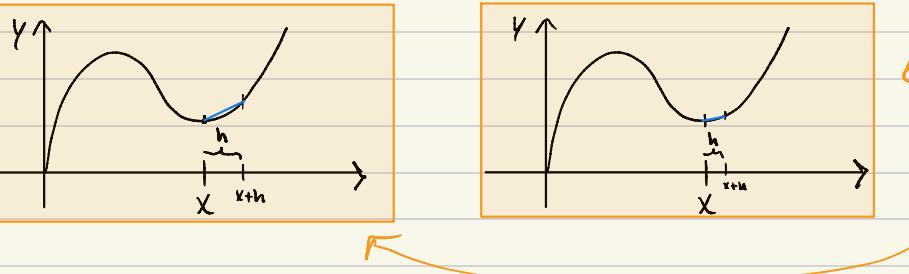
$$h = 10^{-3} : f'(1,5) = \frac{e^{1,501} - e^{1,5}}{0,001} = 4,4839 \times \text{BOM}$$

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

$$h = 10^{-5} : f'(1,5) = \frac{e^{1,50001} - e^{1,5}}{0,00001} = 4,4817 \checkmark \text{ god nok}$$

$$h = 10^{-12} : f'(1,5) = \frac{e^{1,5000001} - e^{1,5}}{0,0000001} = 4,4826 \text{ "går åt skogen"} \quad h = 10^{-12} \text{ for den "går åt skogen"}$$

Her ser man ved jo lavere h-verdi, desto bedre tilnærming. Til det går dårlig (åt skogen). Det er førdi stigningsstallet til sekanten kommer lettere 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: $f'(x) = \frac{f(x+h) - f(x-h)}{2h}$

Brukte python-kode for å løse

PYTHON-KODE:

```
Users > havan > Desktop > approx.py > ...
1 import numpy as np
2
3 x = 1.5
4 h = 1
5
6 for n in range(10):
7     h = h / 10
8     derivate_x = round((np.exp(x+h)-np.exp(x-h)) / (2*h), 4)
9     print(f"\n{h}-verdi: {derivate_x}")
```

$h=10^{-12}$ for den "går åt skogen"

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

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

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

Den totale derivaten vil konsekvens kverande, større igjen med $h, h^3, h^5, \dots, h^{12}$ og vi deler på $2h$. Ender da med h^2, h^4, \dots altså feilen er noe som er proporsjonal 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
4.4817

Her ser man at tek formelen ett
forsøk for å få riktig verdi

$h=10^{-12}$ før "gikk ut skogen"

Python-kode:

```

1 import numpy as np
2
3 def f(x):
4     f = np.exp(x)
5     return f
6
7 def simulate():
8     x = 1.5
9     h = 1
10    for n in range(5):
11        h = h / 10
12        derivate_x = (f(x-2*h)-8*f(x-h)+8*f(x+h)-f(x+2*h))/(12*h)
13        print(round(derivate_x,4))
14
15 simulate()

```

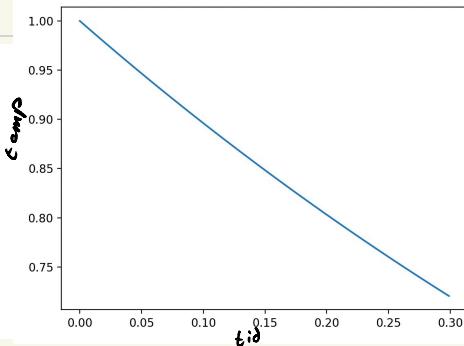
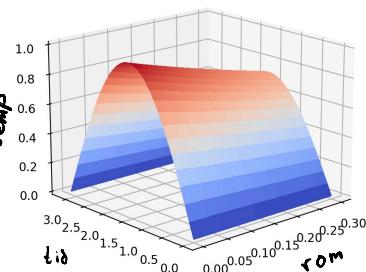
4 EKSPISITT

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

med randkraav $u(0,t) = u(l,t) = 0$

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

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



```

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])

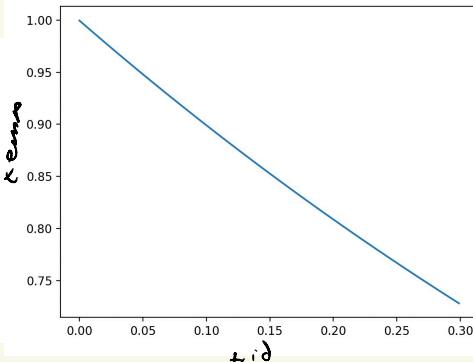
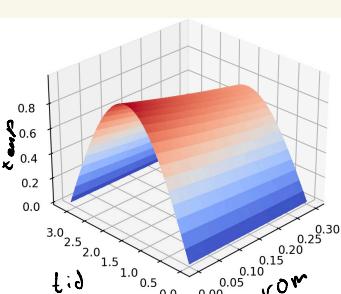
    return t_values, x_values, u_values

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 IMPLISITT

IMPLISITT



```

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):
        # Rører om
        u_values[j + 1][0] = 0
        u_values[j + 1][NUM_STEPS_X - 1] = 0

        # Fiks punkt 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

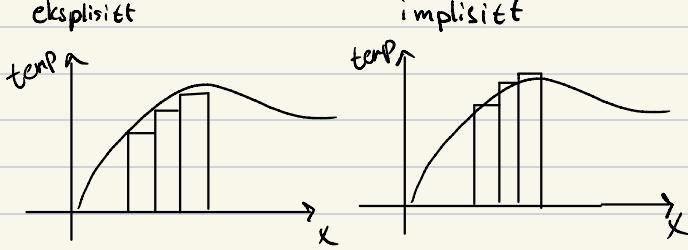
if __name__ == "__main__":
    L = np.pi
    t = 0.3
    k = 0.001
    h = 0.05
    NUM_STEPS_X = int(L / h)

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

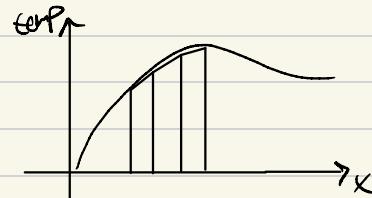
    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
    plt.show()

```

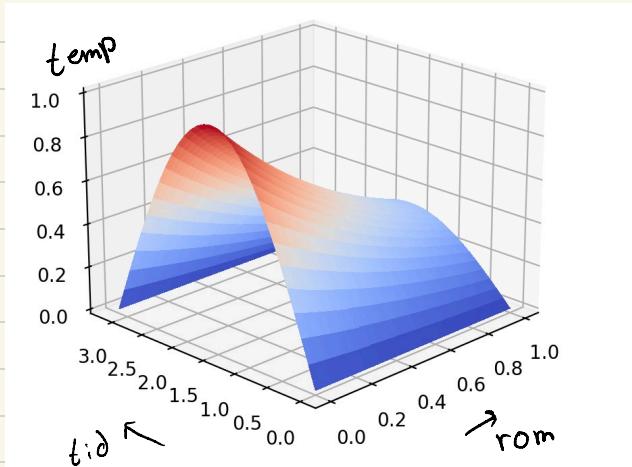
6 eksplisitt



Crank-Nicolson



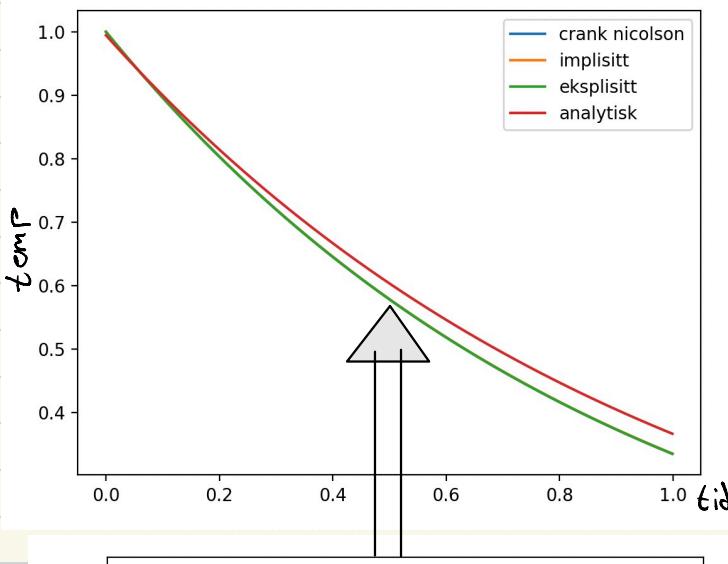
Vi har Crank-Nicolson da den er mer nøyaktig, den summerer halvparten av eksplisitt og halvparten av implisitt.



```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib import cm
4 from matplotlib.ticker import LinearLocator
5
6 # Definer parameterne
7 def crank_nicolson(L, t, k, h):
8     NUM_STEPS_X = int(L / h)
9     NUM_STEPS_T = int(t / k)
10
11 x_values = np.linspace(0, L, NUM_STEPS_X)
12 t_values = np.linspace(0, t, NUM_STEPS_T)
13
14 u_values = np.zeros((NUM_STEPS_T, NUM_STEPS_X))
15 u_values[0] = np.sin(x_values) # initialbetingelse
16
17 x, t, values = np.meshgrid(x_values, t_values, u_values)
18 # sett opp initialverdier i u_values
19 # sett opp u_values[0] = arrayen
20
21 # x_values = np.linspace(0, L, NUM_STEPS_X)
22 for j in range(NUM_STEPS_T - 1):
23     u_values[j+1][0] = 0
24     u_values[j+1][NUM_STEPS_X - 1] = 0
25
26 explicit = np.array(u_values[j])
27
28 for i in range(1, NUM_STEPS_X - 1):
29     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])
30
31 last_values = np.zeros(NUM_STEPS_X)
32
33 implicit = np.array(u_values[j])
34
35 # Fiksunktitasjon for å finne implisitt
36 while (np.sum(np.abs(last_values - implicit)) > 1e-5):
37     last_values = np.array(implicit)
38     for i in range(1, NUM_STEPS_X - 1):
39         implicit[i] = u_values[j][i] + k / np.power(h, 2) * (implicit[i + 1] - 2 * implicit[i] + implicit[i - 1])
40
41 # Crank Nicolson blir et sammenlitt av eksplisitt og implisitt
42 # Med numey regner den ut uttrykket for hver i
43 u_values[j+1] = (explicit + implicit) / 2
44 return t_values, x_values, u_values
45
46 if __name__ == "__main__":
47     L = np.pi
48     t = 0.3
49     k = 0.001
50     h = 0.01
51     NUM_STEPS_X = int(L / h)
52     NUM_STEPS_T = int(t / k)
53     print("Hei!!")
54     t_values, x_values, u_values = crank_nicolson(L, t, k, h)
55     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
56     # ax = fig.add_subplot(111, projection='3d')
57     surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
58     plt.show()
59
60
61

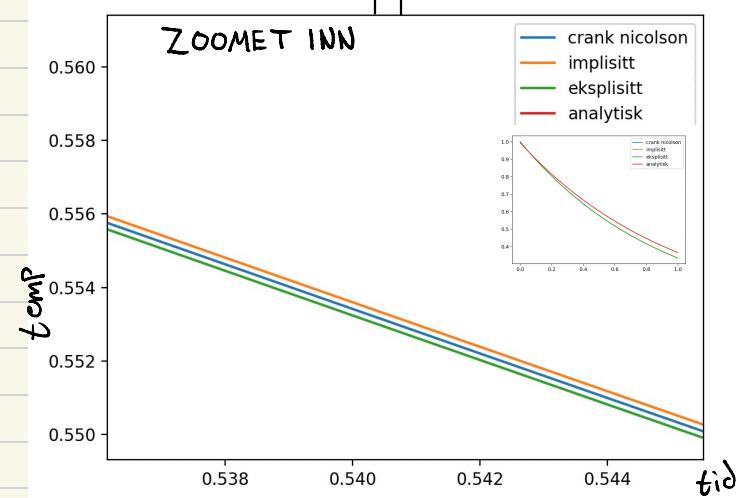
```



```

1 from VarmeligningCrankNicolson import crank_nicolson
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import cm
5 from VarmeligningImplisitt import implisitt
6 from VarmeligningEksplisitt import eksplisitt
7
8
9 if __name__ == "__main__":
10     L = np.pi
11     t = 1
12     k = 0.001
13     h = 0.1
14
15     NUM_STEPS_X = int(L / h)
16     NUM_STEPS_T = int(t / k)
17     t_values, x_values, u_values = crank_nicolson(L, t, k, h)
18     t_values, x_values, u1_values = implisitt(L, t, k, h)
19     t_values, x_values, u2_values = eksplisitt(L, t, k, h)
20
21     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
22
23     surf = ax.plot_surface(t_values, x_values, u_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
24     surf = ax.plot_surface(t_values, x_values, u1_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
25     surf = ax.plot_surface(t_values, x_values, u2_values, cmap=cm.coolwarm, linewidth=0, antialiased=False)
26     plt.show()
27
28     x_point = NUM_STEPS_X // 2
29     x_val = x_values[0, x_point-1]
30     analytic = np.exp(-np.arange(0, t, k)) * np.sin(x_val)
31
32     plt.plot(np.arange(0, t, k), u_values[:, x_point], label="crank nicolson")
33     plt.plot(np.arange(0, t, k), u1_values[:, x_point], label="implisitt")
34     plt.plot(np.arange(0, t, k), u2_values[:, x_point], label="eksplisitt")
35     plt.plot(np.arange(0, t, k), analytic, label="analytisk")
36     plt.legend()
37     plt.show()

```



Ville antatt at implisitt skulle hagne over den analytiske.
og at de numeriske løsningene ikke skulle være så lett.
Men det gjorde den ikke ...