Oppgave 1

Når h blir så liten som 10^{-10} begynner datamaskinen å regne feil.

```python
import numpy as np

x= 1.5
h= [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000000001]

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

def df1(x, h):
    return (f(x+h) -f(x)) / h

s_2 = f(x)
for i in h:
    s_1 = df1(x, i)
    feil = s_1-s_2
    print("h:", i, ", feil: ", feil)
```

```
h: 0.1 , feil:  0.2317444702324396
h: 0.01 , feil:  0.022483327280713006
h: 0.001 , feil:  0.0022415916702973604
h: 0.0001 , feil:  0.00022409192614158968
h: 1e-05 , feil:  2.2408571680010425e-05
h: 1e-06 , feil:  2.2410595752475615e-06
h: 1e-10 , feil:  5.944763585397084e-06
```

Oppgave 2

Dette er en bedre tilnærming enn nr. 1. Dette kan man se med rekkeutvikling fordi leddet med første-ordensfeil blir kansellert.

```python
def df2(f, x, h):
    return (f(x+h) -f(x-h)) / (2*h)

for i in h:
    s_3 = df2(f, x, i)
    feil =abs(s_3-s_2)
    print("h:", i, ", feil: ", feil)
```

```
h: 0.1 , feil:  0.007473217414137423
h: 0.01 , feil:  7.469519133618263e-05
h: 0.001 , feil:  7.469480740596168e-07
h: 0.0001 , feil:  7.468485385686563e-09
h: 1e-05 , feil:  9.660450217552352e-11
h: 1e-06 , feil:  2.5866686570452657e-10
h: 1e-10 , feil:  1.5038714868964576e-06
```

Oppgave 3

Dette er den mest presise metoden av de tre. Fordi det er en fjerde-ordens ledd i likningen.

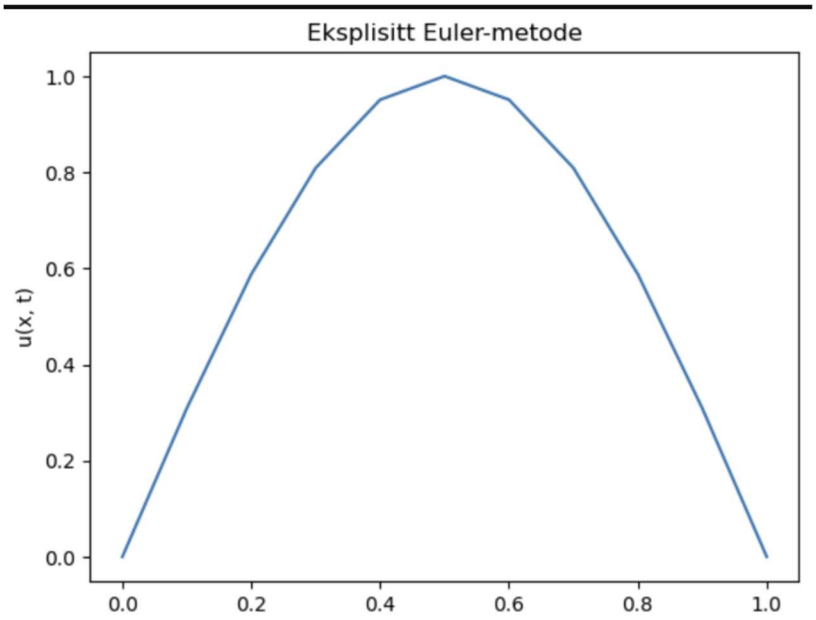
```python
def df3(f, x, h):
    return (f(x-2*h) -8*f(x-h) +8*f(x+h) -f(x+2*h))/(12*h)

for i in h:
    s_4 = df3(f, x, i)
    error3 = abs(s_4-s_2)



    print("h:", i, ",feil: ", error3)
```

```
h: 0.1 ,feil:  1.4956758427331351e-05
h: 0.01 ,feil:  1.4938787984419832e-09
h: 0.001 ,feil:  3.552713678800501e-13
h: 0.0001 ,feil:  3.8458125573015423e-13
h: 1e-05 ,feil:  5.219469301209756e-11
h: 1e-06 ,feil:  3.326814379533971e-10
h: 1e-10 ,feil:  2.9841688533593924e-06
```

Oppgave 4

```python
import matplotlib.pyplot as plt
X = 1
T = 0.5
h = 0.1
k = 0.0001

n = int(X/h)
m = int(T/k)

x = np.linspace(0, X, n+1)
t = np.linspace(0, T, m+1)
u = np.zeros((n+1, m+1))

u[:, 0] = np.sin(np.pi*x)

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

plt.plot(x, u[:, 0])
plt.xlabel('x')
plt.ylabel('u(x, t)')
plt.title('Eksplisitt Euler-metode')
plt.show()
#blir ustabil for store k
```

Denne metoden fungerer best for liten h og den blir ustabil for store k.

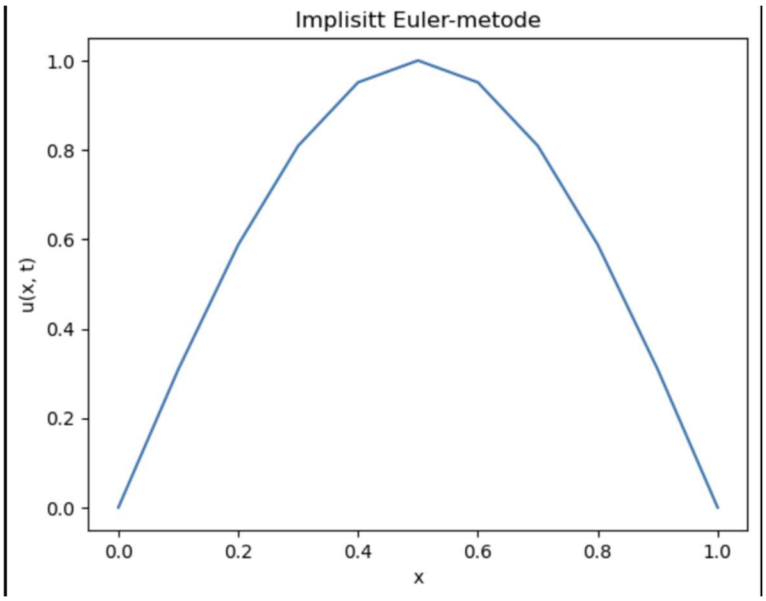


Oppgave 5

Denne metoden er mer stabil enn den eksplisitte metoden, of den er mer nøyaktig for små k.

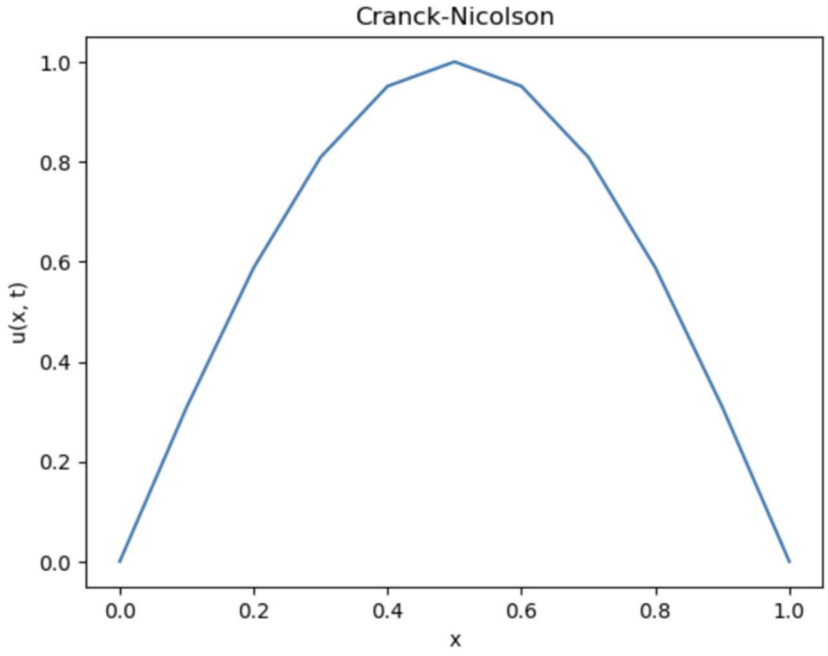
Oppgave 5

```
for i in range(0, m):
    for j in range(1, n):
        u[j, i+1] = u[j, i] + (k/h**2) * (u[j+1, i+1] -2* u[j, i+1] + u[j-1, i+1])
plt.plot(x, u[:, 0])
plt.xlabel('x')
plt.ylabel('u(x, t)')
plt.title('Implisitt Euler-metode')
plt.show()
#mer stabil enn eksplisitt euler, mer nøyaktig for små k
```



Oppgave 6

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

plt.plot(x, u[:, 0])
plt.xlabel('x')
plt.ylabel('u(x, t)')
plt.title('Cranck-Nicolson')
plt.show()
#den mest nøyaktige metoden, spesielt for små k
```



Dette er den mest nøyaktige metoden. Denne fungerer veldig bra for små k.