1)

Jeg bruker dette programmet til å regne.

```python
import math as m
def estdf(x,h):
    return (m.exp(x+h)-m.exp(x))/h

def df(x):
    return m.exp(x)

h = [10**-i for i in range(0,10)]

print(df(1.5))
for i in h:
    print(i, "estimert derivert: ", estdf(1.5, i), "forskjell", estdf(1.5, i) -df(1.5))
```

```
4.4816890703380645
1 estimert derivert:  7.700804890365409 forskjell 3.219115820027344
0.1 estimert derivert:  4.713433540570504 forskjell 0.2317444702324396
0.01 estimert derivert:  4.5041723976187775 forskjell 0.022483327280713006
0.001 estimert derivert:  4.483930662008362 forskjell 0.0022415916702973604
0.0001 estimert derivert:  4.481913162264206 forskjell 0.00022409192614158968
1e-05 estimert derivert:  4.4817114789097445 forskjell 2.2408571680010425e-05
1e-06 estimert derivert:  4.48169131139764 forskjell 2.2410595752475615e-06
1e-07 estimert derivert:  4.481689304114411 forskjell 2.3377634672527847e-07
1e-08 estimert derivert:  4.481689064306238 forskjell -6.0318265937553406e-09
1e-09 estimert derivert:  4.481689686031132 forskjell 6.156930671963323e-07
```

Vi ser at når h deles på 10, deles også feilen på 10. Det blir avrundingsfeil på de mindre tallene.

2)

Hvis vi bruker den andre nye formelen får vi at feilen bli proporsjonal med h^2

```
4.4816890703380645
1 estimert derivert:  5.266886345001673 forskjell 0.7851972746636084
0.1 estimert derivert:  4.489162287752202 forskjell 0.007473217414137423
0.01 estimert derivert:  4.481763765529401 forskjell 7.469519133618263e-05
0.001 estimert derivert:  4.481689817286139 forskjell 7.469480740596168e-07
0.0001 estimert derivert:  4.48168907780655 forskjell 7.468485385686563e-09
1e-05 estimert derivert:  4.481689070434669 forskjell 9.660450217552352e-11
1e-06 estimert derivert:  4.481689070079398 forskjell -2.5866686570452657e-10
1e-07 estimert derivert:  4.481689073188022 forskjell 2.8499576032459117e-09
1e-08 estimert derivert:  4.481689019897317 forskjell -5.04407475787616e-08
1e-09 estimert derivert:  4.481689241941922 forskjell 1.716038573462697e-07
```

Vi kan forklare dette med taylorrekker:

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

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

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

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

Det største feilleddet er proporsjonalt med h^2

3)

```
4.4816890703380645
1 estimert derivert:  4.313438351753924 forskjell -0.1682507185841402
0.1 estimert derivert:  4.481674113579637 forskjell -1.4956758427331351e-05
0.01 estimert derivert:  4.481689068844186 forskjell -1.4938787984419832e-09
0.001 estimert derivert:  4.481689070337709 forskjell -3.552713678800501e-13
0.0001 estimert derivert:  4.481689070338449 forskjell 3.8458125573015423e-13
1e-05 estimert derivert:  4.481689070390259 forskjell 5.219469301209756e-11
1e-06 estimert derivert:  4.481689070005383 forskjell -3.326814379533971e-10
1e-07 estimert derivert:  4.481689073928171 forskjell 3.590106878448296e-09
1e-08 estimert derivert:  4.481688997692856 forskjell -7.264520895944315e-08
1e-09 estimert derivert:  4.481689389971658 forskjell 3.196335933708383e-07
```

4)

Jeg brukte følgende kode

```python
import math as m
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
tider = []

h = 0.01
k = 0.00005
T = 0.1

def f(x):
    return m.sin(m.pi*x)

t0 = [f(i*h) for i in range(0,int(1/h)+1)]
tider.append(t0)

for j in range(1,int(1/k) -1):
    punkter = []
    for i in range(0,int(1/h)+1):
        if i == 0 or i == int(1/h):
            punkter.append(0)
        else:
            punkter.append(tider[j-1][i]+((tider[j-1][i+1] -2*tider[j-1][i] + tider[j-1][i-1])/h**2)*k )
    tider.append(punkter)
print(tider)

fig, ax = plt.subplots()
x = [i*h for i in range(0, int(1/h)+1)]
line, = ax.plot(x,tider[0])
ax.set_ylim(-1,1)
def update(frame):
    y = tider[frame]
    line.set_ydata(y)
    return line,

ani = FuncAnimation(fig, update, frames=int(1/k)-1, interval=10, blit=True)
plt.show()
```

Da jeg satt h og k lik hverandre, steg verdiene veldig fort, men da jeg satt k til å bli mye mindre enn h, fikk jeg en mye roligere animasjon

5)

```python
import math as m
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Parametere
h = 0.01
k = 0.001
L = 1
T = 0.1
nx = int(L / h)
nt = int(T / k)
r = k / h**2


def f(x):
    return m.sin(m.pi * x)

u = [f(i*h) for i in range(nx + 1)]
tider = [u.copy()]


def thomas_algorithm(a, b, c, d):
    n = len(b)
    cp = c.copy()
    bp = b.copy()
    dp = d.copy()

    for i in range(1, n):
        m = a[i-1] / bp[i-1]
        bp[i] = bp[i] - m * cp[i-1]
        dp[i] = dp[i] - m * dp[i-1]

    x = [0] * n
    x[-1] = dp[-1] / bp[-1]
    for i in range(n-2, -1, -1):
        x[i] = (dp[i] - cp[i] * x[i+1]) / bp[i]

    return x

```

```python
n_indre = nx - 1
for t in range(1, nt):

    a = [-r] * (n_indre - 1)
    b = [1 + 2*r] * n_indre
    c = [-r] * (n_indre - 1)
    d = u[1:-1]

    u_inner = thomas_algorithm(a, b, c, d)
    u = [0] + u_inner + [0]
    tider.append(u.copy())


fig, ax = plt.subplots()
x = [i*h for i in range(nx + 1)]
line, = ax.plot(x, tider[0])
ax.set_ylim(-1, 1)

def update(frame):
    line.set_ydata(tider[frame])
    ax.set_title(f"Tid = {frame * k:.4f}")
    return line,

ani = FuncAnimation(fig, update, frames=nt, interval=30, blit=True)
plt.show()
```

Koden løser likningssettet og finner den neste varmespredningen. Denne metoden er mye mer stabil og eksploderer ikke når h er mindre enn k eller når de er like.

6)

Vi finner Crank-Nicolson-metoden ved å ta gjennomsnittet av den implisitte og ekplisitte metoden. Når vi setter alle tre inn i samme graf ser det ut til at den implisitte metoden følger den analytiske løsningen best. Dette er fordi den eksplisitte metoden har en tendens til å eksplodere ved feil verdier av h og k og siden Crank-Nicolsen inneholder den eksplisitte metoden vil denne også gi et galt svar.