Semesteroppgave i TMA4106

Varmelikning

Innholdsfortegnelse

[**Introduksjon** 1](#_Toc165120026)

[**Resultater** 1](#_Toc165120027)

[**Konklusjon** 11](#_Toc165120028)

[**Referanser** 12](#_Toc165120029)

[**Appendix** 13](#_Toc165120030)

[***Kode brukt for approksimasjon av derivert*** 13](#_Toc165120031)

[***Kode brukt for å regne ut varmelikningen*** 16](#_Toc165120032)

# **Introduksjon**

Varmelikningen er en fundamental differensiallikning som beskriver hvordan temperaturen i et materiale endrer seg over tid og rom. Denne likningen brukes for å modellere varmeledning i faste stoffer til å beskrive dynamikken til væsker og gasser. I denne oppgaven skal jeg utforske tre numeriske metoder for å løse denne likningen på; den eksplisitte Eulers metode, implisitt Eulers metode og Crank-Nicholsen metoden. Hver av disse metodene har sine styrker og svakheter, og det er dette som skal undersøkes. Eksplisitt Eulers metode: En enkel og intuitiv tilnærming som tar et skritt fremover i tid basert på nåværende temperatur og dens deriverte. Implisitt Eulers metode: Tar et skritt fremover i tid ved å løse en ligning som inkluderer nåværende og fremtidige temperaturverdier. Crank-Nicholson metoden: Tar en midtpunktstilnærming og kombinerer de to tidligere nevnte metodene.[2]

# **Resultater**

Når man skal sette i gang å begynne og løse partielle differensiallikninger numerisk er varmelikningen lettest å starte med. Den deriverte til funksjon f er gitt ved

dersom den grenseverdien eksisterer.

er stigningen til sekanten til f mellom x og .[1] For små h er denne sekanten en grei tilnærming:

Det antas at desto mindre h er, jo bedre vil denne tilnærmingen være.[1] For eksempel for f(x) = ex og h=0.1 til h=1e-09

**h Approksimert derivert Error**

0.1 4.713433540570504 0.2317444702324396

0.01 4.5041723976187775 0.022483327280713006

0.001 4.483930662008362 0.0022415916702973604

0.0001 4.481913162264206 0.00022409192614158968

1e-05 4.4817114789097445 2.2408571680010425e-05

1e-06 4.48169131139764 2.2410595752475615e-06

1e-07 4.481689304114411 2.3377634672527847e-07

1e-08 4.481689064306238 6.0318265937553406e-09

1e-09 4.481689686031132 6.156930671963323e-07

Ser at feilen er lineær. Deler man h på 10 så deler man feilen på 10. Dette gjelder helt fram til h=1e-07. Der er ikke feilen lengere lineær, og etter h=1e-08 vil feilen bli større igjen.

Gjentar man det samme med denne formelen får man:

**h Approksimert derivert Error**

0.1 4.489162287752202 0.007473217414137423

0.01 4.481763765529401 7.469519133618263e-05

0.001 4.481689817286139 7.469480740596168e-07

0.0001 4.48168907780655 7.468485385686563e-09

1e-05 4.481689070434669 9.660450217552352e-11

1e-06 4.481689070079398 2.5866686570452657e-10

1e-07 4.481689073188022 2.8499576032459117e-09

1e-08 4.481689019897317 5.04407475787616e-08

1e-09 4.481689241941922 1.716038573462697e-07

Her er også feilen lineær, men deler man h på 10 så deler man feilen på 100. Dette stemmer helt fra til h=1e-05, for etter det begynner feilen å øke igjen

Dette ble igjen gjort med

Det gav svært nøyaktige svar ved høye h

**h Approksimert derivert Error**

0.1 4.481674113579637 1.4956758427331351e-05

0.01 4.481689068844186 1.4938787984419832e-09

0.001 4.481689070337709 3.552713678800501e-13

0.0001 4.481689070338449 3.8458125573015423e-13

1e-05 4.481689070390259 5.219469301209756e-11

1e-06 4.481689070005383 3.326814379533971e-10

1e-07 4.481689073928171 3.590106878448296e-09

1e-08 4.481688997692856 7.264520895944315e-08

1e-09 4.481689389971658 3.196335933708383e-07

men ettersom h ble lavere enn 1e-04 ble feilen større igjen

A graph with different colored lines

Description automatically generated

**Figur 1: Eksplisitt Eulers metode med h = 0.025 og k = 0.025**

A graph with a line

Description automatically generated

**Figur 2: Eksplisitt Eulers metode med h= 1 og k = 0.025**

A graph with lines and numbers

Description automatically generated with medium confidence

**Figur 3: Implisitt Eulers metode med h = 0.025 og k = 0.025**

A graph with lines and numbers

Description automatically generated with medium confidence

**Figur 4: Implisitt Eulers metode med h = 0 og k = 0.025**

A graph of a function

Description automatically generated

**Figur 5: Crank-Nicholsen sammenlignet med analytisk løsning med h = 0.025 og k = 0.025**

Vet ikke helt hvorfor bare den ene linjen stemmer, men det den viser er at den analytiske metoden og Crank-Nicholsen vil gi det samme

Beregningene gjort for analytisk løsning for varmelikningen

A paper with writing on it

Description automatically generated

A piece of paper with writing on it

Description automatically generated A piece of paper with writing on it

Description automatically generated  
A piece of paper with writing on it

Description automatically generated

# **Konklusjon**

Fant ut at den eksplisitte Eulers metode er lett å implementere og forstå, men kan være noe begrenset av kravet til små tidssteg for å sikre stabilitet. Implisitt Eulers metode er derimot mer stabil men noe mer beregningsintensiv på grunn av at det er nødvendig å løse ligninger for hver tidsiterasjon. Crank-Nicholsen metoden viste seg å være en god kompromissløsning, som kombinerer fordelene fra eksplisitt- og implisitt Eulers metode. Crank-Nicholsen stemte også godt overens med analytiske beregninger, selv om python koden ikke stemte helt.

# **Referanser**

1. Norges Tekniske Naturvitenskapelige Universitet. *Standardprosjekt*. Hentet 24.04.2024. Tilgjengelig fra: <https://folk.ntnu.no/mortano/prosjekt/standard/standard.pdf>
2. Trench, William. F., *The Heat Equation*. Hentet 23.04.2024. Tilgjengelig fra: <https://math.libretexts.org/Bookshelves/Differential_Equations/Elementary_Differential_Equations_with_Boundary_Value_Problems_(Trench)/12%3A_Fourier_Solutions_of_Partial_Differential_Equations/12.01%3A_The_Heat_Equation#title>

# **Appendix**

## ***Kode brukt for approksimasjon av derivert***

import math

def appDer(f, x, h):

return (f(x + h) - f(x)) / h

def samDer(f, x, h\_values):

der = math.exp(x) # Analytisk derivert av e^x er e^x

results = []

for h in h\_values:

approksimert = appDer(f, x, h)

error = abs(der - approksimert)

results.append((h, approksimert, error))

return results

def main():

x = 1.5

h\_values = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001] # Liste med forskjellige h-verdier

results = samDer(math.exp, x, h\_values)

print("h\tApproksimert derivert\tError")

for h, approksimert, error in results:

print(f"{h}\t{approksimert}\t{error}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

def sentralDiff(f, x, h):

return (f(x + h) - f(x - h)) / (2 \* h)

def sammenlign\_sentralDiff(f, x, h\_values):

der = math.exp(x) # Analytisk derivert av e^x er e^x

results = []

for h in h\_values:

approksimert = sentralDiff(f, x, h)

error = abs(der - approksimert)

results.append((h, approksimert, error))

return results

def main():

x = 1.5

h\_values = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001] # Liste med forskjellige h-verdier

results = sammenlign\_sentralDiff(math.exp, x, h\_values)

print("h\tApproksimert derivert\tError")

for h, approksimert, error in results:

print(f"{h}\t{approksimert}\t{error}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

def femPunktDiff(f, x, h):

return (f(x - 2\*h) - 8\*f(x - h) + 8\*f(x + h) - f(x + 2\*h)) / (12 \* h)

def sammenlignFemPunktDiff(f, x, h\_values):

der = math.exp(x) # Analytisk derivert av e^x er e^x

results = []

for h in h\_values:

approksimert = femPunktDiff(f, x, h)

error = abs(der - approksimert)

results.append((h, approksimert, error))

return results

def main():

x = 1.5

h\_values = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001] # Liste med forskjellige h-verdier

results = sammenlignFemPunktDiff(math.exp, x, h\_values)

print("h\tApproksimert derivert\tError")

for h, approksimert, error in results:

print(f"{h}\t{approksimert}\t{error}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## ***Kode brukt for å regne ut varmelikningen***

import numpy as np

import matplotlib.pyplot as plt

#eksplisitt euler

h=0.025

k=0.025

x = np.arange(0, 1+h, h)

t = np.arange(0, 0.1+k, k).round(3)

boundaryConditions = [0, 0]

initialConditions = np.sin(np.pi\*x)

factor = k/h\*\*2

n = len(x)

m = len(t)

T = np.zeros((n,m))

T[0,:] = boundaryConditions[0]

T[-1, :] = boundaryConditions[1]

T[:,0] = initialConditions

T.round(3)

for j in range(1,m):

for i in range(1,n-1):

T[i,j] = factor\*T[i-1, j-1]+(1-2\*factor)\*T[i,j-1] + factor\*T[i+1,j-1]

T.round(3)

plt.plot(T)

plt.xlabel('Avstand')

plt.ylabel('Tid')

#implisitt euler

A = np.diag([1+2\*factor]\*(n-2),0) + np.diag([-factor]\*(n-3), -1) + np.diag([-factor]\*(n-3), 1)

for j in range(1, m):

b = T[1:-1,j-1].copy()

b[0] = b[0] + factor\*T[0 , j]

b[-1] = b[-1] + factor\*T[-1, j]

solution = np.linalg.solve(A,b)

T[1:-1,j] = solution

print(solution)

R = np.linspace(1,0,m)

B = np.linspace(0,1,m)

G = 0

for j in range(m):

plt.plot(x,T[:,j], color= [R[j-2],G,B[j-2]])

plt.xlabel('Avstand')

plt.ylabel('Tid')

#Crank-Nicholsen

B = np.diag([2+2\*factor]\*(n-2),0) + np.diag([-factor]\*(n-3),-1)+ np.diag([-factor]\*(n-3), 1)

C = np.diag([2-2\*factor]\*(n-2),0) + np.diag([factor]\*(n-3),-1)+ np.diag([factor]\*(n-3), 1)

for j in range(0, m-1):

c = T[1:-1,j].copy()

c = np.dot(C, c)

c[0] = c[0] + factor\*(T[0,j]+T[0,j+1])

c[-1] = c[-1] + factor\*(T[-1,j] + T[-1,j+1])

solution = np.linalg.solve(B,c)

T[1:-1,j+1]=solution

T.round(2)

R = np.linspace(1,0,m)

B = np.linspace(0,1,m)

G = 0

#analytisk

def analytisk(x,t):

return np.exp(-np.pi\*\*2\*t/k)\*np.sin(np.pi\*x)

for j in range(m):

plt.plot(x, T[:,j], color= [R[j],G,B[j]])

plt.plot(x, analytisk(x, t[j]),'--')

plt.xlabel('Avstand')

plt.ylabel('Tid')

plt.legend(t)