NTNU

Kunnskap for en bedre verden

Department of Marine Technology

TMA4106 - Matematikk 2

nO HUMOR ØVING

Author:
Tario Solberg Hanski

Date

Table of Contents

List of Figures

List of Tables

1 Introduksjon

Dette forsøket tar for seg å løse varmeligningen numerisk med metodene eksplisitt, implisitt og Crank-Nicolson. Det vil først bli sett på noen metoder for numerisk derivasjon og hvor nøyaktige det er med forskjellige steglengder h. Det vil se på hva som skjer når en øker

2 Numerisk derivasjon

Første mini-eksperiment består i å benytte følgende formler for numerisk derivasjon:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

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

Disse formlene gir tilnærminger for henholdsvis førstederiverte av en funksjon $f(x)$, basert på små endringer h. Ved å velge små verdier for h, kan man beregne tilnærmede verdier for $f'(x)$ og $f''(x)$.

I eksperimentene benyttes funksjonen $f(x) = e^x$, ettersom den har den nyttige egenskapen at alle dens deriverte også er e^x. Dette gjør det enkelt å sammenligne numeriske tilnærminger med eksakt verdi. Verdien vi sammenligner med i testene er:

$$f'(1.5) = f(2) = e^{1.5} \approx 7.3890560989$$

Koden som ble benyttet vises i Listing 1.

Listing 1: Numerisk derivasjon med fremoverdifferanse

```
import numpy as np

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

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

def f_central(x, h):
    return (f(x + h) - f(x - h)) / (2 * h)

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

x = 2
true_value = f(x)

print(r"\begin{tabular}{|c|c|c|c|c|c|c|}")
print(r"\hline")
print(r"$h$ & Fremover & Feil & Sentral & Feil & 4. ordens & Feil \\")
print(r"\hline")

for i in range(1, 17):
    h = 0.1 ** i
    fwd = f_forward(x, h)
    cent = f_central(x, h)
    fourth = f_fourth_order(x, h)

    err_fwd = abs(fwd - true_value)
    err_cent = abs(cent - true_value)
```

```
    err_fourth = abs(fourth - true_value)

    print(f"{h:.1e} & {fwd:.10f} & {err_fwd:.2e} & "
          f"{cent:.10f} & {err_cent:.2e} & "
          f"{fourth:.10f} & {err_fourth:.2e} \\\\")
print(r"\hline")
print(r"\end{tabular}")
```

Koden over evaluerer tre numeriske metoder for første derivert i punktet $x = 2$, for h-verdier som går ned til 10^{-17}. Resultatene er vist i tabellen under.

h	Fremover	Feil	Sentral	Feil	4. ordens	Feil
1.0e-01	4.7134335406	2.32e-01	4.4891622878	7.47e-03	4.4816741136	1.50e-05
1.0e-02	4.5041723976	2.25e-02	4.4817637655	7.47e-05	4.4816890688	1.49e-09
1.0e-03	4.4839306620	2.24e-03	4.4816898173	7.47e-07	4.4816890703	3.56e-13
1.0e-04	4.4819131623	2.24e-04	4.4816890778	7.47e-09	4.4816890703	3.85e-13
1.0e-05	4.4817114789	2.24e-05	4.4816890704	9.66e-11	4.4816890704	5.22e-11
1.0e-06	4.4816913114	2.24e-06	4.4816890701	2.59e-10	4.4816890700	3.33e-10
1.0e-07	4.4816893041	2.34e-07	4.4816890732	2.85e-09	4.4816890739	3.59e-09
1.0e-08	4.4816890643	6.03e-09	4.4816890199	5.04e-08	4.4816889977	7.26e-08
1.0e-09	4.4816896860	6.16e-07	4.4816892419	1.72e-07	4.4816893900	3.20e-07
1.0e-10	4.4816950151	5.94e-06	4.4816905742	1.50e-06	4.4816920545	2.98e-06
1.0e-11	4.4817483058	5.92e-05	4.4817038969	1.48e-05	4.4817186999	2.96e-05
1.0e-12	4.4826364842	9.47e-04	4.4821923950	5.03e-04	4.4825624694	8.73e-04
1.0e-13	4.4853010195	3.61e-03	4.4808601274	8.29e-04	4.4801199787	1.57e-03
1.0e-14	4.5297099405	4.80e-02	4.4853010195	3.61e-03	4.4853010195	3.61e-03
1.0e-15	5.3290705182	8.47e-01	4.8849813084	4.03e-01	5.1070259133	6.25e-01
1.0e-16	0.0000000000	4.48e+00	0.0000000000	4.48e+00	-1.4802973662	5.96e+00
1.0e-17	0.0000000000	4.48e+00	0.0000000000	4.48e+00	-7.4014868308	1.19e+01

Table 1: Numerisk tilnærming av $f'(2)$ med ulike metoder og feil mot eksakt verdi $e^2 \approx 7.3891$.

Som man ser av tabellen, fungerer de numeriske metodene godt for moderate h-verdier, men svikter når h blir for liten. Dette skyldes begrenset presisjon i datamaskinens flyttallsrepresentasjon (typisk 16 desimaler for `float64`). For fremoverdifferansen ser vi at verdien kollapser til null når $h < 10^{-15}$, og sentraldifferansen følger etter. Fjerdeordensmetoden gir svært høy nøyaktighet for mellomstore h, men påvirkes også av avrundingsfeil når h blir ekstremt lite.

Feilens oppførsel kan forklares analytisk ved hjelp av Taylor-rekker, som drøftes nærmere i neste avsnitt.

2.1 Feilanalyse med Taylor-rekker

For å forstå nøyaktigheten til de ulike numeriske metodene, benytter vi Taylor-rekker for å utvikle funksjonsverdiene rundt punktet x. Vi ønsker å se hvordan feilen avhenger av h, og hvor raskt metodene konvergerer mot den sanne verdien når $h \to 0$.

Fremoverdifferanse

Fremoverdifferansemetoden er gitt ved:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Vi Taylor-utvikler $f(x+h)$ rundt x:

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

Setter vi dette inn i formelen får vi:

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

Dette viser at feilen er proporsjonal med h, altså:

$$\text{Feil}_{\text{fremover}} = \mathcal{O}(h)$$

Metoden har dermed førsteordens nøyaktighet.

Sentraldifferanse

Sentraldifferansen er gitt ved:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Vi Taylor-utvikler $f(x+h)$ og $f(x-h)$:

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

Trekker vi disse og deler på $2h$:

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

Dette gir en feil proporsjonal med h^2:

$$\text{Feil}_{\text{sentral}} = \mathcal{O}(h^2)$$

Altså er metoden andreordens nøyaktig.

Fjerdeordens differanse

Fjerdeordens formelen er:

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

Ved å bruke Taylor-rekker for alle fire funksjonsverdier rundt x, og sette det inn, får man:

$$f'(x) + \mathcal{O}(h^4)$$

Detaljert utledning kan vises, men for formålet her holder det å vite at denne metoden kansellerer ut de lavere ordens leddene i Taylor-rekken, og gir en **fjerdeordens nøyaktighet**:

$$\text{Feil}_{\text{fjerde}} = \mathcal{O}(h^4)$$

Dette betyr at når h reduseres med en faktor 10, reduseres feilen med en faktor $10^4 = 10\,000$, så lenge vi holder oss over maskinens avrundingsgrense.

Oppsummering

Taylor-analyse gir en analytisk forklaring på hvorfor sentraldifferanse og spesielt fjerdeordens metode er mer presise enn fremoverdifferanse. Resultatene i tabellen i forrige seksjon bekrefter dette: for moderate h-verdier oppnår sentraldifferanse betydelig bedre presisjon enn fremover, og fjerdeordens metode gir nær maskinpresisjon før avrundingsfeil gir mindre pressisjon.

3 Numerisk løsning av varmelikningen med feilanalyse

I denne delen skal vi løse den én-dimensjonale varmelikningen

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2},$$

med randbetingelser

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

og initialbetingelse

$$u(x,0) = f(x), \quad x \in [0,1],\ t \geq 0.$$

3.1 Diskretisering

For numerisk løsning benytter vi et gitter i både tid og rom:

- Rom: Del intervallet $x \in [0,1]$ inn i n delintervaller med gitterstørrelse $h = \frac{1}{n}$. Gitterpunktene defineres som $x_i = ih$.
- Tid: Del tidsintervallet inn i steg med størrelse k, og la $t_j = jk$.

Vi approksimerer løsningen ved hvert gitterpunkt med $u_i^j \approx u(x_i, t_j)$.

Romdiskretisering: Den andrederiverte med hensyn på x tilnærmes med en sentraldifferanse:

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) \approx \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{h^2}.$$

Tidsdiskretisering: For den tidslige derivert kan vi benytte ulike tilnærmingsmetoder, for eksempel:

- Eksplisitt Euler:
$$\frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u_i^{j+1} - u_i^j}{k},$$
- Implisitt Euler, eller
- Crank–Nicolson.

Ved å kombinere tid- og romdiskretiseringen oppnår vi et system som beskriver utviklingen av u_i^j over tid.

3.2 Det diskretiserte skjemaet

For eksempel gir den eksplisitte Euler-metoden følgende oppdateringsformel:

$$\frac{u_i^{j+1} - u_i^j}{k} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{h^2} \quad \Rightarrow \quad u_i^{j+1} = u_i^j + \frac{k}{h^2}\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j\right).$$

Denne formelen brukes for å beregne løsningen for hvert tidssteg. (Her brukes selvsagt også de tilsvarende skjemaene for implisitt Euler og Crank–Nicolson i implementasjonen.)

3.3 Numerisk løsning med feilanalyse

Vi løser varmelikningen med tre ulike metoder:

- Eksplisitt Euler-metode,
- Implisitt Euler-metode, og
- Crank–Nicolson-metode.

Her bruker vi initialbetingelsen

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

og de samme randbetingelsene som nevnt. Den numeriske løsningen sammenlignes med en analytisk tilnærming som er basert på en Fourier-sinuserie med Fourier-koeffisientene

$$a_n = \frac{\sin(1 - n\pi)}{1 - n\pi} - \frac{\sin(1 + n\pi)}{1 + n\pi}.$$

For hver metode beregnes den maksimale absolutte feilen – definert som det største avviket mellom den numeriske løsningen og den analytiske løsningen – som en funksjon av tid.

Implementasjon

Listing 2 er koden for de ulike numeriske metodene,

Listing 2: Numerisk derivasjon med fremoverdifferanse

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation


# Parametere for domene og gitter
L = 1.0
nx = 50                  # Antall intervaller => (nx+1) gitterpunkter
h = L / nx
x = np.linspace(0, L, nx + 1)

# Tidsparametre
T = 0.05                 # Sluttid

# For eksplisitt metode     stabilitetsbetingelsen: dt <= hˆ 2/2
dt_explicit = 0.5 * h**2
nt_explicit = int(T / dt_explicit)

# For implisitt metode kan vi bruke et annet tidssteg
dt_implicit = 0.005
nt_implicit = int(T / dt_implicit)

# For Crank-Nicolson bruker vi samme dt som for eksplisitt
dt_cn = 0.5 * h**2
nt_cn = int(T / dt_cn)
```

```
# Initialbetingelse: u(x,0) = sin(x)
u0 = np.sin(x)
u0[0] = 0
u0[-1] = 0  # Tvinger randbetingelsene


# Eksplisitt metode
def explicit_scheme(u, dt, h, nt):
    u_all = [u.copy()]
    for n in range(nt):
        u_next = u.copy()
        for i in range(1, len(u)-1):
            u_next[i] = u[i] + dt/(h**2) * (u[i+1] - 2*u[i] + u[i-1])
        u_next[0] = 0
        u_next[-1] = 0
        u = u_next.copy()
        u_all.append(u.copy())
    return np.array(u_all)

# Implisitt metode
def implicit_scheme(u, dt, h, nt):
    u_all = [u.copy()]
    r = dt / h**2
    n_interior = len(u) - 2
    A = np.zeros((n_interior, n_interior))
    for i in range(n_interior):
        A[i, i] = 1 + 2*r
        if i > 0:
            A[i, i-1] = -r
        if i < n_interior - 1:
            A[i, i+1] = -r
    for n in range(nt):
        u_inner = u[1:-1]
        u_inner_new = np.linalg.solve(A, u_inner)
        u[1:-1] = u_inner_new
        u[0] = 0; u[-1] = 0
        u_all.append(u.copy())
    return np.array(u_all)

# Crank-Nicolson metode
def crank_nicolson(u, dt, h, nt):
    u_all = [u.copy()]
    r = dt / h**2
    n_interior = len(u) - 2
    A = np.zeros((n_interior, n_interior))
    B = np.zeros((n_interior, n_interior))
    for i in range(n_interior):
        A[i, i] = 1 + r
        B[i, i] = 1 - r
        if i > 0:
            A[i, i-1] = -r/2
            B[i, i-1] = r/2
        if i < n_interior - 1:
            A[i, i+1] = -r/2
            B[i, i+1] = r/2
    for n in range(nt):
        u_inner = u[1:-1]
        rhs = B.dot(u_inner)
        u_inner_new = np.linalg.solve(A, rhs)
        u[1:-1] = u_inner_new
        u[0] = 0; u[-1] = 0
        u_all.append(u.copy())
    return np.array(u_all)


# Analytisk l sning med Fourier sinuserie
def analytic_solution(x, t, n_terms=50):
    u = np.zeros_like(x)
    # For initialbetingelsen f(x)=sin(x):
    # a_n = (sin(1-n*pi)/(1-n*pi)) - (sin(1+n*pi)/(1+n*pi))
    for n in range(1, n_terms+1):
        a_n = (np.sin(1 - n*np.pi)/(1 - n*np.pi) - np.sin(1 + n*np.pi)/(1 + n*np.pi
```

```
            ))
        u += a_n * np.sin(n * np.pi * x) * np.exp(- (n*np.pi)**2 * t)
    return u


# Beregn de numeriske l sningene
solution_explicit = explicit_scheme(u0.copy(), dt_explicit, h, nt_explicit)
solution_implicit = implicit_scheme(u0.copy(), dt_implicit, h, nt_implicit)
solution_cn = crank_nicolson(u0.copy(), dt_cn, h, nt_cn)

# Opprett tidstabeller for hver metode
time_explicit = np.linspace(0, T, len(solution_explicit))
time_implicit = np.linspace(0, T, len(solution_implicit))
time_cn = np.linspace(0, T, len(solution_cn))


# Beregn feil over tid for de tre metodene (maks absolutt feil)
errors_explicit = []
for i, t in enumerate(time_explicit):
    u_an = analytic_solution(x, t, n_terms=50)
    error = np.max(np.abs(solution_explicit[i] - u_an))
    errors_explicit.append(error)

errors_implicit = []
for i, t in enumerate(time_implicit):
    u_an = analytic_solution(x, t, n_terms=50)
    error = np.max(np.abs(solution_implicit[i] - u_an))
    errors_implicit.append(error)

errors_cn = []
for i, t in enumerate(time_cn):
    u_an = analytic_solution(x, t, n_terms=50)
    error = np.max(np.abs(solution_cn[i] - u_an))
    errors_cn.append(error)


# Plotting: Sammenlign sluttl sningene med den analytiske l sningen ved t = T
u_analytic = analytic_solution(x, T, n_terms=50)
plt.figure(figsize=(10,6))
plt.plot(x, solution_explicit[-1], 'o-', label='Eksplisitt')
plt.plot(x, solution_implicit[-1], 's-', label='Implisitt')
plt.plot(x, solution_cn[-1], 'x-', label='Crank-Nicolson')
plt.plot(x, u_analytic, 'k--', label='Analytisk')
plt.xlabel('x')
plt.ylabel('u(x, T)')
plt.title(f'Sammenligning av numeriske l sninger og analytisk l sning ved t = {T}
    ')
plt.legend()
plt.show()


# Plotting: Feil over tid for de tre metodene
plt.figure(figsize=(10,6))
plt.plot(time_explicit, errors_explicit, 'o-', label='Eksplisitt feil')
plt.plot(time_implicit, errors_implicit, 's-', label='Implisitt feil')
plt.plot(time_cn, errors_cn, 'x-', label='Crank-Nicolson feil')
plt.xlabel('Tid (t)')
plt.ylabel('Maks absolutt feil')
plt.title('Utvikling av maks absolutt feil over tid')
plt.legend()
plt.show()


# Eksempel: Animasjon av Crank-Nicolson l sningen (test)
fig, ax = plt.subplots()
line, = ax.plot(x, solution_cn[0], lw=2)
ax.set_xlim(0, L)
ax.set_ylim(-1.1, 1.1)
ax.set_xlabel('x')
ax.set_ylabel('u(x,t)')
ax.set_title('Crank-Nicolson metode')

def animate_cn(n):
```

```
    line.set_ydata(solution_cn[n])
    ax.set_title(f'Crank-Nicolson, t = {n*dt_cn:.4f}')
    return line,

ani = animation.FuncAnimation(fig, animate_cn, frames=len(solution_cn), interval
    =50)
plt.show()
```

Resultater

Figur 1 viser sluttløsningen ved tidspunkt $t = T$ for de tre metodene, sammenlignet med den analytiske løsningen.

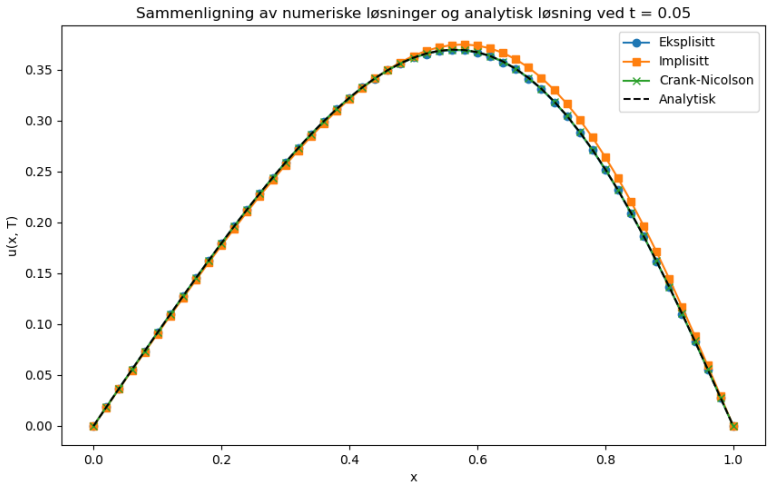

Figure 1: Sluttløsning ved $t = T$ for eksplisitt, implisitt og Crank–Nicolson-metodene, sammenlignet med den analytiske løsningen.

Figur 2 viser utviklingen av den maksimale absolutte feilen over tid for de tre metodene.

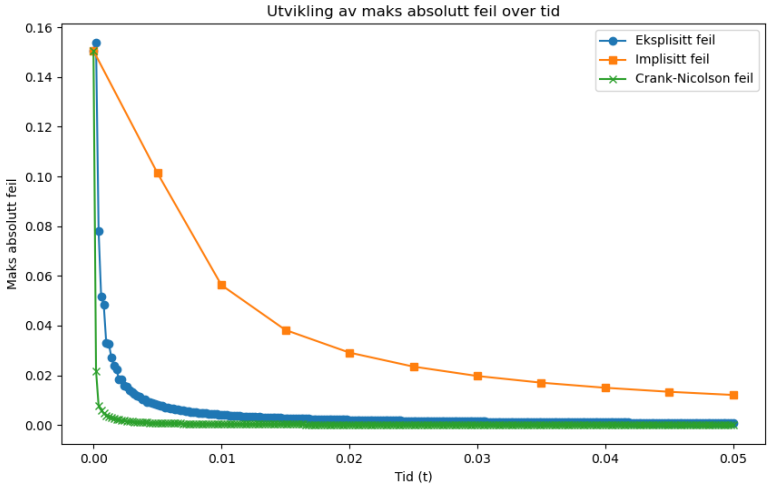

Figure 2: Utviklingen av maks absolutt feil over tid for de tre metodene.

4 Konklusjon

I denne rapporten har vi undersøkt tre metoder for å løse den én-dimensjonale varmelikningen med randbetingelsene u(0,t)=u(1,t)=0u(0,t)=u(1,t)=0 og initialbetingelsen u(x,0)=sin(x)u(x,0)=sin(x). Vi har implementert og sammenlignet eksplisitt Euler-, implisitt Euler- og Crank–Nicolson-metoden.

Resultatene viser at eksplisitt Euler-metoden er enkel, men krever svært små tidssteg for å holde løsningen stabil. Implisitt Euler-metoden er stabil ved større tidssteg, men oppnår lavere nøyaktighet. Crank–Nicolson-metoden gir den mest nøyaktige løsningen, med lavere feil sammenlignet med den analytiske løsningen. Feilanalysen bekrefter at Crank–Nicolson-metoden konvergerer raskere og med mindre feil over tid.

Sammendrag: Valget av numerisk metode og de tilhørende diskretiseringsparametrene har stor betydning for nøyaktigheten og stabiliteten til løsningen, og Crank–Nicolson-metoden fremstår som den beste kompromissløsningen i dette prosjektet.

Fremtidig arbeid kan blant annet se på utvidelser til høyere dimensjoner og forbedringer i visualiseringen av løsningen, for eksempel ved bruk av 3D-plott for å vise den fulle dynamikken i både tid og rom.

NTNU 2025 NTNU 2021 NTNU 2015

Bibliography

NTNU (2015). *Crank-Nicolson Method*. https://www.math.ntnu.no/emner/TMA4135/2015h/notater/crank-nicolson/cn.pdf. Accessed: 12.04.2025.

— (2021). *Numerical Methods for the Heat Equation*. https://www.math.ntnu.no/emner/TMA4125/2021v/lectures/NumericalMethodsHeatEquation.pdf. Accessed: 12.04.2025.

— (2025). *OBLIG - TMA4106*. Øving 7.