Oblig: TMA4106

Julian Martinsen

April 2025

1 Introduksjon

Slik som elektrisitet har jeg ved liten kreativitet valgt minste motstands vei. Jeg gjør eksempel-oblig i dag. Vi skal ta for oss et parr numeriske likningsløsere for å se hvordan de oppfører ved ulike parameterverdier, og til slutt sammenlikne med den analytiske løsningen.

2 Første del

Først blir vi bedt om å ta for oss numeriske derivasjonsmetoder. Det er disse sammenhengende som skal utforskes:

$$f'(x) = \frac{f(x+h) - f(x)}{h} \tag{1}$$

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

For den første likningen kunne jeg komputere meg til ca. $h = 10^{-8}$ før løsningen begynte å skli i feil retning, men jeg oppdaget at jeg ikke kom meg særlig lengre ved den andre likningen. Da gikk jeg inn for å plotte forskjellen fra den analytiske løsningen til begge metodene ved forskjellig h. Plottet ble slikt:

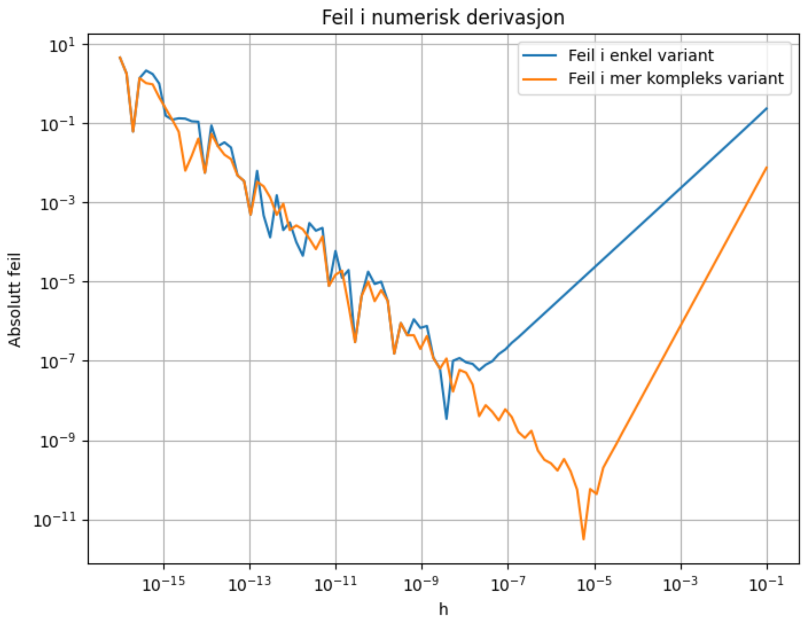

Figure 1: Plott av feildifferanse

Vi ser at (2) når sin laveste feilverdi mye tidligere enn (1). D.v.s. at den er mer effektiv ved en høyere h verdi enn definisjonen av den deriverte. (2), sin mer kompliserte utregning fører til desimalfeil ved høyere h enn (1), men oppnår alikevell en mye bedre tilnærming.

Vi blir bedt om å taylorutvikle (2), som gjøres slik:

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

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(x)}{3}h^3 + ... \tag{4}$$

Så tar vi differansen og deler på $2h$:

$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{f'''(x)}{6}h^2 + \frac{f'''''(x)}{120}h^4 + ... \tag{5}$$

Vi ser at h leddet påvirker feilen med h^2 for hver del av utviklingen. Derfor sier vi at (2) har kvadratisk feil.

Jeg slår ikke på stortrommen da jeg tror konseptet er greit.

3 Andre del

Nå skal vi ta for oss numerisk løsning av varmelikningen. Den ser slik ut:

$$\dot{u}(x,t) = u''(x,t) \tag{6}$$

Vi har randkrav

$$u(0,t) = u(1,t) = 0 \tag{7}$$

og initialkrav

$$u(x,0) = f(x). \tag{8}$$

Jeg skrev kode for den eksplisitte løsningen (alle løsningene ligger i samme vedlegg, da vi uansett måtte sammenlikne til slutt.(ikke animasjonsfunksjonaliteten, der stappet jeg inn min kode inn i vår venn deepseek for hjelp, og hjelp fikk jeg)). Jeg fant fort ut at når h og k var like eksploderte den eksplisitte løseren. Jeg begynte så å google og fant et gøyalt konsept kalt stabilitet. Jeg fant ut at den eksplisitte metoden var stabil for alle verdier av $r = \frac{k}{h^2} \leq \frac{1}{2}$ og der finner man forklaringen.

Den implisitte metoden klarte jeg ikke å få til å eksplodere på likt vis. Litt søking der gjorde at jeg fant ut at denne metoden var stabil for alle k og h. Jeg oppdaget da derimot at koden krasjet programmet ved radikale verdier med for mange 0-er.

For Crank-Nicolson klarte jeg ikke å framprovosere samme eksplosjon som eksplisitt. Mine søk oppdaget at det kan forekomme en oscillering ved høye r-verdier, men min pc ga seg før animasjonen viste tegn til å kapitulere.

I animasjonen av alle løsningene inkludert analytisk, ser vi at ved en h og k verdi som tilfredstiller stabilitetskravet til eksplisitt, følger vi den analytisk ganske tett med alle metodene.

Ved 'Pillow' biblioteket kunne jeg eksportere animasjonen til en gif som ligger i repo-en.

4 Konklusjon

For varmelikningen ved våre krav, vil alle 3 numeriske metodene være brukbare, men eksplisitt kan eksplodere internt, mens implisitt og Crank-Nicolson kan eksplodere pc-en din.

5 Vedlegg

Kode for feildifferanseplott:

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


def f(x):
```

```
    return np.exp(x)

sann_derivert = np.exp(1.5)

x = 1.5
h_values = np.logspace(-16, -1, 100)

feiler_enkel = []
feiler_værre = []

for h in h_values:
    derivert_enkel = (f(x + h) - f(x)) / h
    derivert_værre = (f(x + h) - f(x - h)) / (2*h)

    feil_enkel = abs(derivert_enkel - sann_derivert)
    feil_værre = abs(derivert_værre - sann_derivert)

    feiler_enkel.append(feil_enkel)
    feiler_værre.append(feil_værre)


plt.figure(figsize=(8, 6))
plt.loglog(h_values, feiler_enkel, label='Feil i enkel variant')
plt.loglog(h_values, feiler_værre,
                label='Feil i mer kompleks variant')
plt.xlabel("h")
plt.ylabel("Absolutt feil")
plt.title("Feil i numerisk derivasjon")
plt.grid(True)
plt.legend()
plt.show()
```

Kode for numerisk løsning og plotting:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from scipy.linalg import solve_banded

L = np.pi
T = 1.0
h = 0.05
k = 0.001

x = np.arange(0, L + h, h)
t = np.arange(0, T + k, k)
```

```
N = len(x)
r = k / h**2

u0 = np.sin(x)
u0[0] = 0
u0[-1] = 0

def solve_explisitt():
    u = u0.copy()
    solutions = [u.copy()]
    for _ in t[1:]:
        u_new = u.copy()
        u_new[1:-1] = u[1:-1] + r * (u[2:] - 2*u[1:-1] + u[:-2])
        u = u_new
        solutions.append(u.copy())
    return solutions

def solve_implisitt():
    a = -r * np.ones(N-3)
    b = (1 + 2*r) * np.ones(N-2)
    c = -r * np.ones(N-3)

    ab = np.zeros((3, N-2))
    ab[0,1:] = c
    ab[1,:] = b
    ab[2,:-1] = a

    u = u0.copy()
    solutions = [u.copy()]

    for _ in t[1:]:
        d = u[1:-1]
        u_inner = solve_banded((1,1), ab, d)
        u[1:-1] = u_inner
        solutions.append(u.copy())
    return solutions

def solve_crank_nicolson():
    alpha = r / 2

    a = -alpha * np.ones(N-3)
    b = (1 + alpha*2) * np.ones(N-2)
    c = -alpha * np.ones(N-3)

    ab = np.zeros((3, N-2))
    ab[0,1:] = c
```

```
    ab[1,:] = b
    ab[2,:-1] = a

    u = u0.copy()
    solutions = [u.copy()]

    for _ in t[1:]:
        d = alpha*u[:-2] + (1 - 2*alpha)*u[1:-1] + alpha*u[2:]
        u_inner = solve_banded((1,1), ab, d)
        u[1:-1] = u_inner
        solutions.append(u.copy())
    return solutions

def solve_analytisk(x, t, alpha=1.0):
    Nt = len(t)
    Nx = len(x)
    U = np.zeros((Nt, Nx))
    for n in range(Nt):
        U[n, :] = np.sin(x) * np.exp(-alpha * t[n])
    return U

def animate(solutions, title):
    fig, ax = plt.subplots()
    line, = ax.plot(x, solutions[0])
    ax.set_ylim(-1.1, 1.1)
    ax.set_title(title)

    def update(frame):
        line.set_ydata(solutions[frame])
        ax.set_xlabel(f"tid = {frame * k:.3f} s")
        return line,

    ani = FuncAnimation(fig, update, frames=len(solutions), interval=30)
    plt.show()



animate(solve_explisitt(), "Eksplisitt skjema")
animate(solve_implisitt(), "Implisitt skjema")
animate(solve_crank_nicolson(), "Crank-Nicolson skjema")


def animate_comparison():
    u_exp = solve_explisitt()
    u_imp = solve_implisitt()
    u_cn = solve_crank_nicolson()
```

```
    u_ana = solve_analytisk(x,t)
    fig, ax = plt.subplots()
    line_exp, = ax.plot(x, u_exp[0], label="Eksplisitt", linestyle="--")
    line_imp, = ax.plot(x, u_imp[0], label="Implisitt", linestyle="-")
    line_cn, = ax.plot(x, u_cn[0], label="Crank-Nicolson", linestyle=":")
    line_ana, = ax.plot(x, u_ana[0], label="Analytisk")
    ax.set_ylim(-1.1, 1.1)
    ax.set_title("Sammenlikning av metoder")
    ax.legend()

    def update(frame):
        line_exp.set_ydata(u_exp[frame])
        line_imp.set_ydata(u_imp[frame])
        line_cn.set_ydata(u_cn[frame])
        line_ana.set_ydata(u_ana[frame])
        ax.set_xlabel(f"tid = {frame * k:.3f} s")
        return line_exp, line_imp, line_cn

    ani = FuncAnimation(fig, update, frames=len(u_exp), interval=30)
    plt.show()

animate_comparison()
```