Partielle differensialligninger

Anne Kværnø

Problemstilling

I dette notatet skal vi diskutere hvordan partielle differensialligninger kan diskretiseres ved hjelp av en endelig differansemetode, og litt om hvilke utfordringer som dukker opp. Vi bruker varmeledningsligningen

$$u_t = u_{xx}$$

som et eksempel på en tidsavhenging ligning, og Poissons ligning

$$u_{xx} + u_{yy} = f(x, y)$$

som eksempel på en statisk ligning. Begge disse ligningen ble diskutert i første del av dette kurset, så jeg antar at dere vet hvordan definisjonsområdet velges for disse ligningene, og hva slags randbetingelser man kan sette opp.

Følgende differanseformler vil bli brukt:

$$f'(x) = \begin{cases} \dfrac{f(x+h)-f(x)}{h} - \dfrac{h}{2}f''(\xi) & \text{Foroverdifferanse} \\ \dfrac{f(x)-f(x-h)}{h} + \dfrac{h}{2}f''(\xi) & \text{Bakoverdifferanse} \\ \dfrac{f(x+h)-f(x-h)}{2h} - \dfrac{h^2}{6}f'''(\xi) & \text{Sentraldifferanse} \end{cases}$$

$$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} - \frac{h^2}{12}f^{(4)}(\xi), \qquad \text{Sentraldifferanse}$$

Diskretiseringen og numerisk løsning av en ligning består av følgende punkter.

1. Tegn defininsjonsområdet, og tegn inn randbetingelsene.
2. Diskretiser definisjonsområdet (lag et rutenett over området)
3. Sett opp en diskret ligning i hvert gridpunkt hvor løsningen ikke er kjent. Dette gjøres ved å erstatte de deriverte med en differanseformel i hvert gitterpunkt. Vi ender da opp med et system av algebraiske ligninger.
4. Løs disse på en eller annen måte.

Selv om framgangsmåten i og for seg er den samme, vil denne teknikken anvendt på forskjellige ligninger ha sine spesielle utfordringer, hvilket her blir demonstrert på varmeledningsligningen og Poissons ligning.

Poissons ligning

Gitt Poissons ligning

$$u_{xx} + u_{yy} = f(x, y)$$

på enhetskvadratet $0 \leq x, y \leq 1$ med randbetingelser

$$u(x, 0) = g_s(x), \quad u(x, 1) = g_n(x), \quad u(0, y) = g_v(y) \quad \text{og} \quad u(1, y) = g_o(y).$$

Indeksene på randbetingelsene står for hhv. sør, nord, vest og øst.

Denne ligningen kan diskretiseres som følger:

1. Tegn enhetskvadratet og skriv inn randbetingelsene.
2. Velg en N slik at $h = 1/N$. Gitterpunktene (x_i, y_j) er gitt av

   $$x_i = ih, \quad y_j = jh, \qquad i, j = 0, \ldots, N.$$

   Tegn dette inn i figuren.
3. Bruk sentraldifferanser i hhv. x- og y-retningen for å tilnærme u_{xx} og u_{yy} i et tilfeldig valgt gitterpunkt (x_i, y_j) (ikke et randpunkt):

   $$u_{xx}(x_i, y_j) \approx \frac{u(x_i + h, y_j) - 2u(x_i, y_j) + u(x_i - h, y_j)}{h^2},$$
   $$u_{yy}(x_i, y_j) \approx \frac{u(x_i, y_j + h) - 2u(x_i, y_j) + u(x_i - h, y_j - h)}{h^2}.$$

   Erstatt $u(x_i, y_j)$ med en tilnærmelsen $U_{i,j}$ og sett dette inn i differensialligningen. Differanseapproksimasjonen til differensialligningen blir da

   $$\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} = f(x_i, y_j), \qquad i, j = 1, \ldots, N-1,$$

   eller

   $$U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j} = h^2 f(x_i, y_j).$$

   Denne diskretiseringen kalles ofte *fempunktsformelen*. Denne sammen med randbetingelsene

   $$U_{i,0} = g_s(x_i), \quad U_{i,N} = g_n(x_i), \quad U_{0,j} = g_v(y_j), \quad U_{N,j} = g_o(y_j)$$

   danner et lineært ligningssystem

   $$A\mathbf{U} = \mathbf{b}$$

   med $N - 1 \times N - 1$ ligninger.
4. Løs ligningssystemet.

La oss demonstrere hvordan ligningsystemet blir seende ut for $N = 4$.

La $\mathbf{U} = [U_{1,1}, U_{2,1}, U_{3,1}, U_{1,2}, U_{2,2}, U_{3,2}, U_{3,1}, U_{3,2}, U_{3,3}]^T$ (radvis nummerering) være vektoren med tilnærmelsene til løsningen i gitterpunktet. Sett opp ligningene i samme rekkefølge, og flytt alle randverdiene, der løsningen er kjent over på høyresiden. Ligningssystemet blir

$$
\begin{aligned}
-4U_{1,1} + U_{2,1} + U_{1,2} &= h^2 f(x_1, y_1) - U_{0,1} - U_{1,0}, \\
U_{1,1} - 4U_{2,1} + U_{3,1} + U_{2,2} &= h^2 f(x_2, y_1) - U_{2,0}, \\
U_{2,1} - 4U_{3,1} + U_{3,2} &= h^2 f(x_3, y_1) - U_{3,0} - U_{1,4}, \\
&\vdots \\
U_{2,2} + U_{1,3} - 4U_{2,3} + U_{3,3} &= h^2 f(x_2, y_3) - U_{4,2}, \\
U_{3,2} + U_{2,3} - 4U_{3,3} &= h^2 f(x_3, y_3) - U_{3,4} - U_{4,3},
\end{aligned}
$$

som kan skrives som

$$
\begin{bmatrix}
-4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4
\end{bmatrix}
\begin{bmatrix}
U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ U_{1,3} \\ U_{2,3} \\ U_{3,3}
\end{bmatrix}
=
\begin{bmatrix}
h^2 f(x_1, y_1) - U_{0,1} - U_{1,0} \\
h^2 f(x_2, y_1) - U_{2,0} \\
h^2 f(x_3, y_1) - U_{3,0} - U_{1,4} \\
h^2 f(x_1, y_2) - U_{0,2} \\
h^2 f(x_2, y_2) \\
h^2 f(x_3, y_2) - U_{4,2} \\
h^2 f(x_1, y_3) - U_{0,3} - U_{1,4} \\
h^2 f(x_2, y_3) - U_{2,4} \\
h^2 f(x_3, y_3) - U_{3,4} - U_{4,3}
\end{bmatrix}
$$

For en generell N vil A bli en *blokk-tridiagonal matrise*, gitt ved:

$$
A = \begin{bmatrix}
B & I_{N-1} & & \\
I_{N-1} & \ddots & \ddots & \\
& \ddots & \ddots & I_{N-1} \\
& & I_{N-1} & B
\end{bmatrix} \in \mathbb{R}^{(N-1)^2 \times (N-1)^2},
$$

$$
B = \begin{bmatrix}
-4 & 1 & & \\
1 & \ddots & \ddots & \\
& \ddots & \ddots & 1 \\
& & 1 & -4
\end{bmatrix} \in \mathbb{R}^{(N-1) \times (N-1)}
$$

og I_{N-1} er identitetsmatrisen. A er et typisk eksempel på en *glissen (sparse) matrise*, en matrise hvor bare et fåtall elementer er forskjellig fra 0. I vårt tilfelle vil det aldri ble mer enn 5 elementer forskjellig fra 0 i hver rad. Dette er utnyttet i koden under, der A er konstruert som en glissen matrise, og ligningssystemet $A\mathbf{U} = \mathbf{b}$ er løst med en glissen lineær ligningssløser. Fordelen med dette er demonstrert i seksjonen etter implementasjonen.

Varianter

- Neumann-betingelser på deler av randa håndteres med falske render som beskrevet i notatet om to punkts-randverdiproblemer.
  For andre definisjonsområder Dersom randene gitteret ikke stemmer overens med randbetingelsene

Implementasjon

En implementasjon av en numerisk løser for Poissons ligning på enhetskvadratet består av følgende:

1. Definer kildefunksjonen f, og randbetingelsene g_s, g_v, g_n, g_o.
2. Velg N, og sett opp gitteret.
3. Lag A-matrisen
4. Lag b-vektoren.
5. Løs ligningssystemet $A\mathbf{U} = \mathbf{b}$.
6. Presenter løsningen i et plott.

For å teste implementasjonen velger vi en differensialligning med eksakt løsning

$$u(x, y) = x^3 + 2y,$$

dvs.

$$u_{xx} + u_{yy} = 3x + 4 \qquad 0 \leq x, y \leq 1$$

med randbetingelser gitt av den eksakte løsningen. I dette tilfellet vil den numeriske løsningen være lik den eksakte i gitterpunktene (hvorfor?), så det er lett å undersøke om implementasjonen er riktig.

1. Velg først $N = 4$ og sammelign numerisk og eksakt løsning. Skriv ut A-matrisen og b-vektoren.
2. Velg deretter $N = 100$. Tar det lang tid å løse ligningen?

In [ ]:

```
# Importer nødvendige moduler, og sett parametre for plotting.
# Dette må alltid kjøres først!
%matplotlib inline
from numpy import *               # Matematiske funksjoner og lin.alg.
from scipy.linalg import solve
import scipy.sparse as sparse     # Glisne matriser
from scipy.sparse.linalg import spsolve   # Lineær løser for glisne matriser
from matplotlib.pyplot import *   # Grafikk
newparams = {'figure.figsize': (8.0, 4.0), 'axes.grid': True,
             'lines.markersize': 8, 'lines.linewidth': 2,
             'font.size': 14}
rcParams.update(newparams)
from mpl_toolkits.mplot3d import Axes3D     # For 3-d plott
from matplotlib import cm
```

In [ ]:

```
# Sett opp randbetingelser og kildefunksjonen f
def gs(x):                  # y=0
    return x**3
def gn(x):                  # y=1
    return 2+x**3
def gv(y):                  # x=0
    return 2*y**2
def go(y):                  # x=1
    return 1+2*y**2
def f(x,y):
    return 6*x+4

N = 4  # Antall intervaller

# Lag gitteret
x = linspace(0, 1, N+1)
y = linspace(0, 1, N+1)
h = 1/N

# De indre punktene i gitteret
xi = x[1:-1]
yi = y[1:-1]
Xi, Yi = meshgrid(xi, yi)
Ni = N-1       # Antall indre gitterpunkter i hver retning
Ni2 = Ni**2    # Antall indre gitterpunkter (og ukjente) totalt

# Sett opp A-matrise (glissen)
B = sparse.diags([1,-4,1],[-1,0,1],shape=(Ni, Ni), format="lil")
A = sparse.kron(sparse.eye(Ni), B)
C = sparse.diags([1,1],[-Ni,Ni],shape=(Ni2, Ni2), format="lil")
A = (A+C).tocsr() # Konverter til csr-format (nødvendig for spsolve)

# Sett opp b-vektoren
b = zeros(Ni2)
# Legg til randbetingelsene
b[0:Ni] = b[0:Ni]-gs(xi)                          # y=0
b[Ni2-Ni:Ni2] = b[Ni2-Ni:Ni2] - gn(xi)            # y=1
b[0:Ni2:Ni] = b[0:Ni2:Ni] - gv(yi)                # x=0
b[Ni-1:Ni2:Ni] = b[Ni-1:Ni2:Ni] - go(yi)          # x=1
# Legg til kildefunksjonen
b = b+h**2*f(Xi,Yi).flatten()

# Løs ligningssystemet.
Ui = spsolve(A, b)      # Løs det glisne lineære ligningssystemet

# Lag et (N+1)x(N+1) array for å lagre løsningen i gitterpunktene,
# inkludert rendene
U = zeros((N+1, N+1))

# Gjør om vektoren Ui til en Ni x Ni-matrise og
# plasser den i de indre punktene av U
U[1:-1,1:-1] = reshape(Ui, (Ni,Ni))

# Legg på randverdiene
U[0,:] = gs(x)
U[N,:] = gn(x)
U[:,0] = gv(y)
U[:,N] = go(y)
```

In [ ]:

```
# Skriv ut A og b
print('A =\n ', A.toarray())           # A konverteres til en full matrise først
print('\nb=\n', b)
```

In [ ]:

```
# Skriv ut numerisk og eksakt løsning
X, Y = meshgrid(x,y)
U_eksakt = X**3+2*Y**2
print('\nU=\n', U)
print('\nu_eksakt=\n', U_eksakt)
```

In [ ]:

```
# Plott løsningen
fig = figure()
X, Y = meshgrid(x,y)
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, U, cmap=cm.coolwarm)            # Surface-plot
#ax.plot_wireframe(X, Y, U)         # Mesh-plot
xlabel('x')
ylabel('y')
title('Løsning av Poissons ligning');
```

Fordelen med glisne ligningssystemer

Vi demonstrerer dette med et lite eksempel:

Velg $N = 100$ (litt mindre om du har en langsom PC med lite minne, større om du har en rask med mye minne). Vi har altså $99^2 = 9801$ ligninger som skal løses. Kjør koden over, slik at en glissen matrise A og en korresponderende $\mathbf{b}$-vektor blir konstruert.

Løs deretter $A\mathbf{U} = \mathbf{b}$ både med en full og med en glissen ligningsløser og mål tidsbruken.

Moralen er: Bruk glisne matriser for store systemer, det er mye CPU-tid og minne å spare på det.

In [ ]:

```
import time
# Fullt system
Af = A.toarray()            # Lag en full matrise
start = time.time()
U = solve(Af, b)            # Løs systemet med en vanlig ligningsløser
ferdig = time.time()
print('Tid brukt for en full ligningsløser:', ferdig-start)
```

In [ ]:

```
# Glissent system
start = time.time()
U = spsolve(A, b)            # Løs systemet med en glissen ligningsløser
ferdig = time.time()
print('Tid brukt for en glissen ligningsløser:', ferdig-start)
```

Varmeledningsligningen

Gitt varmeledningsligningen

$$\begin{aligned} u_t &= u_{xx}, && 0 \leq x \leq 1 \\ u(0,t) &= g_0(t), \quad u(1,t) = g_1(t), && \text{Randbetingelser} \\ u(x,0) &= f(x) && \text{Startbetingelse} \end{aligned}$$

I praksis må vi stoppe et sted, f.eks. når $t = t_{end}$.

Semi-diskretisering

Det er ingenting som hindrer oss i å bruke teknikken for diskretisering vist ovenfor. Men for tidsavhengige ligninger er det mulig å bruke et alternativ, kalt semidiskretisering: Gjør en diskretisering i rom (i x-retningen). Dette gir oss en ordinær differensialligning i hvert gitterpunkt. Løs dette systemet av ordinære differensialligninger med en av metodene diskutert tidligere i kurset.

- Diskretisering i x-retningen: Velg M, la $\Delta x = 1/M$, og gitterpunktene $x_i = i\Delta x$.

For et fast gitterpunkt x_i er $u(x_i, t)$ en funksjon av t alene.

- Bruk en sentraldifferanse for u_{xx} i gitterpunktet x_i for et vilkårlig tidspunkt t:

$$\frac{\partial u}{\partial t}(x_i, t) \approx \frac{u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)}{\Delta x^2}.$$

  La $U_i(t) \approx u(x_i, t)$, slik at $\dot{U}_i(t) = \frac{dU_i}{dt} \approx u_t(x_i, t)$. Bruk dette i uttrykket over, slik at vi får følgende system av ordinære differensialligninger

$$\dot{U}_i(t) = \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{\Delta x^2}, \qquad i = 1, 2, \ldots, M-1,$$

  sammen med randbetingelsene $U_0(t) = g_0(t)$ og $U_M(t) = g_1(t)$ og startbetingelsen $U_i(0) = f(x_i)$, $i = 1, 2, \ldots, M-1$.
- Løs denne med en eller annen teknikk for å løse ordinære differensialligninger. For eksempel eksplisitt Eulers metode med steglengde Δt, slik at $t_n = n\Delta t$, og $U_{i,n} \approx U_i(t_n) \approx u(x_i, t_n)$.

$$U_{i,n+1} = U_{i,n} + \frac{\Delta t}{\Delta x^2}\Big(U_{i+1,n} - 2U_{i,n} + U_{i-1,n} \Big).$$

  Dette er for øvrig akkurat det samme som du ville ha kommet fram til ved å bruke diskretiseringsteknikken beskrevet i introduksjonen, med en sentraldifferanse i x og en foroverdifferanse i t.

Implementasjon

Test algoritmen over på ligningen $u_t = u_{xx}$ på intervallet $[0, 1]$. Løs denne fra $t = 0$ til $t = 0.5$.

Eksempel 1.

Start- og randbetingelser er gitt av:

$$u(x, 0) = \sin(\pi x), \qquad g_0(t) = g_1(t) = 0.$$

Til sammenligning er den eksakte løsningen gitt ved

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x).$$

Eksempel 2.

Start- og randbetingelser er gitt av:

$$u(x,0) = \begin{cases} 2x & 0 \le x \le 0.5, \\ 2(1-x) & 0.5 < x \le 2-2x, \end{cases} \qquad g_0(t) = 0, \quad g_1(t) = 0.$$

I begge tilfellene: prøv med

- M=4, N=20
- M=8, N=40
- M=16, N=80

In [ ]:

```
# Gi start og randbetingelser:
def f1(x):                # Eksempel 1
    return sin(pi*x)

def f2(x):                # Eksempel 2
    y = 2*x
    y[x>0.5] = 2-2*x[x>0.5]
    return y

def g0(t):
    return 0
def g1(t):
    return 0
```

In [ ]:

```
# Løs varmeledningsligningen med en foroverdifferanse
#
# Diskretisering av definisjonsområdet:
M = 4
Dx = 1/M
x = linspace(0,1,M+1)
tend = 0.5
N = 20
Dt = tend/N
t = linspace(0,tend,N+1)

# Sett opp et array for å lagre løsningene U_{i,j}
U = zeros((M+1,N+1))
U[:,0] = f1(x)              #  Startbetingelsen U_{i,0} = f(x_i

r = Dt/Dx**2
print('r =',r)

# Hovedløkke
for n in range(N):
    U[1:-1, n+1] = U[1:-1,n] + r*(U[2:,n]-2*U[1:-1,n]+U[0:-2,n])
    U[0, n+1] = g0(t[n+1])
    U[M, n+1] = g1(t[n+1])
```

In [ ]:

```
# Plott numerisk løsning
fig = figure()
ax = fig.gca(projection='3d')
T, X = meshgrid(t,x)
# ax.plot_wireframe(T, X, U)
ax.plot_surface(T, X, U, cmap=cm.coolwarm)
ax.view_init(azim=30)              # Roter figuren
xlabel('t')
ylabel('x')
title('Løsning av varmemledningsligningen');
```

In [ ]:

```
# Plott feilen for eksempel 1
def u_eksakt(x,t):
    return exp(-pi**2*t)*sin(pi*x)

fig = figure()
ax = fig.gca(projection='3d')
feil = u_eksakt(X, T) - U
ax.plot_wireframe(T, X, feil, color = 'r')
ax.view_init(azim=30)
xlabel('t')
ylabel('x')
print('Maksimal feil:', max(abs(feil.flatten())))        # Maksimal feil over he
le arrayet
```

Løsningen er stabil for $M = 4, N = 20$, mens den er tydeligvis ustabil for $M = 16, N = 80$. Hvorfor?

Stabilitetsanalyse

Det semi-diskretiserte ligningssystemet

$$\dot{U}_i(t) = \frac{U_{i+1}(t) - 2U_i(t) + U_{i-1}(t)}{\Delta x^2}, \qquad i = 1, 2, \ldots, M-1, \qquad U_0(t) = g_0(t),$$
$$U_M(t) = g_1(t)$$

er en lineær differensialligning på formen

$$\dot{\mathbf{U}} = \frac{1}{\Delta x^2}\Big(A\mathbf{U} + \mathbf{g}(t)\Big),$$

der

$$\mathbf{U} = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{M-1} \end{bmatrix}, \qquad A = \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix} \qquad \text{og} \qquad \mathbf{g}(t) = \begin{bmatrix} g_0(t) \\ 0 \\ \vdots \\ 0 \\ g_1(t) \end{bmatrix}$$

Stabilitetsegenskapene for lineære ligninger ble diskutert i notatet `StiveODL`. For Eulers metode ble det vist at steglengden Δt må velges slik at $-2 \leq \Delta t \lambda_k \leq 0$ for alle egenverdiene λ_k til $\frac{1}{\Delta x^2}A$. Matrisen A har egenverdiene (se vedlegg):

$$\lambda_k = -4\sin^2\Big(\frac{k\pi}{M}\Big), \qquad k = 1, \cdots, M-1.$$

Alle egenverdiene til $\frac{1}{\Delta x^2}$ er negative og mindre enn $4/\Delta x^2$ i absoluttverdi. Vi konkluderer med at den numeriske løsningen er stabil dersom

$$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}.$$

Dette er et typisk eksempel på en stiv differensialligning, og den eneste måten å unngå en slik stabilitetsbegrensning er å benytte en $A(0)$-stabil metode, for eksempel implisitt Euler eller trapesmetoden.

Implisitt Euler:

Implisitt Euler for det diskretiserte systemet $\dot{\mathbf{U}} = \frac{1}{\Delta x^2}\Big(A\mathbf{U} + \mathbf{g}(t)\Big)$ er

$$\mathbf{U}_{n+1} = \mathbf{U}_n + rA\mathbf{U}_{n+1} + r\,\mathbf{g}(t_{n+1}), \qquad \text{med} \qquad r = \frac{\Delta t}{\Delta x^2}.$$

Element i vektoren $\mathbf{U}_n$ er altså $U_{i,n} \approx u(x_i, t_n)$. For hvert tidssteg må følgende ligningssystem løses:

$$(I_{M-1} - rA)\mathbf{U}_{n+1} = \mathbf{U}_n + r\,\mathbf{g}(t_{n+1}).$$

Metoden kan også konstrueres ved å bruke en bakoverdifferanse i tid og en sentraldifferanse i rom. Differanseapproksimasjonen til varmeledningsligningen i punkt (x_i, t_{n+1}) blir

$$\frac{U_{i,n+1} - U_{i,n}}{\Delta t} = \frac{U_{i+1,n+1} - 2U_{i,n+1} + U_{i-1,n+1}}{\Delta x^2}.$$

Crank-Nicolson (trapesmetoden)

Vi kan også løse metoden over med trapesmetoden, som også er ubetinget stabil og i tillegg av orden 2 i Δt. Anvendt på den semidiskretiserte varmeledningsligningen er metoden mer kjent under navnet Crank-Nicolson. Den er gitt av

$$\mathbf{U}_{n+1} = \mathbf{U}_n + \frac{\Delta t}{2\Delta x^2} A\Big(\mathbf{U}_{n+1} - \mathbf{U}_n\Big) - \frac{\Delta t}{2\Delta x^2}\Big(\mathbf{g}(t_n) + \mathbf{g}(t_{n+1})\Big).$$

For hvert tidssteg må ligningssystemet

$$(I_{M-1} - \frac{r}{2}A)\mathbf{U}_{n+1} = (I_{M-1} + \frac{r}{2}A)\mathbf{U}_n - \frac{r}{2}\Big(\mathbf{g}(t_n) + \mathbf{g}(t_{n+1}\Big), \qquad r = \frac{\Delta t}{\Delta x^2}$$

løses med hensyn på $\mathbf{U}_{n+1}$.

Metoden kan også konstrueres ved å anvende en sentraldifferanse for u_t i punktet $x_i, t_{n+1/2}$ med steglengde $\Delta t/2$, og middelverdien a sentraldifferansene for u_{xx} i punktene (x_i, t_n) og $x_i, t_{n+1})$.
Differanseapproksimasjonen til ligningen blir

$$\frac{U_{i,n+1} - U_{i,n}}{\Delta t} = \frac{1}{2}\left(\frac{U_{i+1,n+1} - 2U_{i,n+1} + U_{i-1,n+1}}{\Delta x^2} + \frac{U_{i+1,n+1} - 2U_{i,n+1} + U_{i-1,n+1}}{\Delta x^2}\right).$$

Feilen er $\mathcal{O}(\Delta x^2 + \Delta t^2)$.

Implementasjon

Det er ingenting i veien for å løse ligningen ved hjelp av metodene utviklet i notatet om stive differensialligninger. Men den implementasjonen er ikke spesielt effektiv, så metodene slik de er beskrevet over er implementert i det følgende.

For hvert tidssteg må det løses et ligningssystem:

$$K\mathbf{U}_{n+1} = \mathbf{b}$$

der:

- Implisitt Euler:

$$K = I_{M-1} - rA, \qquad \mathbf{b} = \mathbf{U}_n + r[g_0(t_n), 0, \dots, 0, g_1(t_n)]^T$$

- Crank-Nicolson:

$$K = I_{M-1} - \frac{r}{2}A,$$

$$\mathbf{b} = (I_{M-1} + \frac{r}{2}A)\mathbf{U}_n + r\left[\frac{1}{2}(g_0(t_n) + g_0(t_{n+1})), 0, \dots, 0, \frac{1}{2}(g_1(t_n) + g_1(t_{n+1}))\right]^T.$$

Metodene testes på et eksempel med ikke-trivielle randbetingelser:

Eksempel

Løs ligningen

$$u_t = u_{xx}, \qquad u(0,t) = e^{-\pi^2 t}, \quad u(1,t) = -e^{-\pi^2 t}, \quad u(x,0) = \cos(\pi x).$$

Den eksakte løsningen er $u(x,t) = e^{-\pi^2 t}\cos(\pi x)$. Løs ligningen til $t_{end} = 0.2$.

Bruk $M = N$ for $M = 10$ og 100. Legg merke til at det ikke oppstår noe stabilitetsproblem selv om faktoren $r = \Delta t/\Delta x^2$ blir stor.

In [ ]:

```
def tridiag(v, d, w, N):
    # Hjelpefunksjon.
    # Lager en tridiagonal matrise A=tridiag(v, d, w) av dimensjon N x N.
    e = ones(N)        # array [1,1,...,1] av lengde N
    A = v*diag(e[1:],-1)+d*diag(e)+w*diag(e[1:],1)
    return A
```

In [ ]:

```
# Implementasjon av implisitt Euler og Crank-Nicolson
# for varmeledningsligningen u_t=u_{xx} på [0,1].


def f3(x):
    return cos(pi*x)
def g0(t):
    return exp(-pi**2*t)
def g1(t):
    return -exp(-pi**2*t)
def u_eksakt(x,t):
    return exp(-pi**2*t)*cos(pi*x)

f = f3


# metode = 'iEuler'
metode = 'CrankNicolson'
# Diskretisering av definisjonsområdet:
M = 10
Dx = 1/M
x = linspace(0,1,M+1)
tend = 0.2
N = 10
Dt = tend/N
t = linspace(0,tend,N+1)

# Sett opp et array for å lagre løsningene U_{i,j}
U = zeros((M+1,N+1))
U[:,0] = f(x)              #  Startbetingelsen U_{i,0} = f(x_i)

# Sett opp matrisa på høyresiden:
A = tridiag(1, -2, 1, M-1)
r = Dt/Dx**2
print('r = ', r)
if metode is 'iEuler':
    K = eye(M-1) - r*A
elif metode is 'CrankNicolson':
    K = eye(M-1) - 0.5*r*A

Utmp = U[1:-1,0]          # Løsning for de indre punktene i tidssteg n.

# Hovedløkke
for n in range(N):
    if metode is 'iEuler':
        b = copy(Utmp)                    # Kopierer arrayet, ikke bare pekeren
        b[0] = b[0] + r*g0(t[n+1])
        b[-1] = b[-1] + r*g1(t[n+1])
    elif metode is 'CrankNicolson':
        b = dot(eye(M-1)+0.5*r*A, Utmp)
        b[0] = b[0] + 0.5*r*(g0(t[n])+g0(t[n+1]))
        b[-1] = b[-1] + 0.5*r*(g1(t[n])+g1(t[n+1]))

    Utmp = solve(K,b)           # Løser ligningen K*Utmp = b

    U[1:-1,n+1] = Utmp        # Lagrer løsningen
    U[0, n+1] = g0(t[n+1])    # Legg inn randvilkårene
    U[M, n+1] = g1(t[n+1])
```

In [ ]:

```
# Plott eksakt og numerisk løsning
fig = figure()
ax = fig.gca(projection='3d')
T, X = meshgrid(t,x)

ax.plot_surface(T, X, U, cmap=cm.coolwarm)
ax.view_init(azim=30)
xlabel('t')
ylabel('x');
```

In [ ]:

```
# Plott feilen for eksempel 1
fig = figure()
ax = fig.gca(projection='3d')
feil = u_eksakt(X, T) - U
ax.plot_wireframe(T, X, u_eksakt(X, T)-U,color = 'r')
ax.view_init(azim=30)
xlabel('t')
ylabel('x')
print('Maksimal feil={:.2e}'.format(max(abs(feil.flatten()))))        #
```

Vedlegg

Egenverdier for tridiagonale matriser

En tridiagonal $n \times n$-matrise $A = \text{tridiag}\{v, d, w\}$ har egenverdier

$$\lambda_k = d + 2\sqrt{vw}\cos\left(\frac{\pi k}{n+1}\right).$$