Obligatorisk prosjekt 1, INF5620

Av: Laila Andersland, brukernavn:lailaea

19. september 2017

1 Hensikt

Hensikten med dette prosjektet er å lage et allsidig kode i python som løser differensiallikninger, for så å bruke den i forskjellige situasjoner.

2 Hoveddel av prosjektet

2.2 Diskretisering

Vi har en 2-dimensjonal, standard, lineær bølgelikning med demping:

$$\frac{\partial^2 u}{\partial t^2} + b\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}(q(x,y)\frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(q(x,y)\frac{\partial u}{\partial y}) + f(x,y) \tag{1}$$

Diskretisert PDE:

$$[D_t D_t u + bD_t u = D_x(qD_x u) + D_y(qD_y u) + f]^n_{i,j} \tag{2}$$

Som gir

$$\frac{u^{n+1}_{i,j} - 2u^n_{i,j} + u^{n-1}_{i,j}}{\Delta t^2} + b\frac{u^{n+1}_{i,j} - u^{n-1}_{i,j}}{2\Delta t} = \frac{1}{\Delta x^2}(q_{i+1/2,j}(u^n_{i+1,j} - u^n_{i,j}) - q_{i-1/2,j}(u^n_{i,j} - u^n_{i-1,j})) + \frac{1}{\Delta y^2}(q_{i,j+1/2}(u^n_{i,j+1} - u^n_{i,j}) - q_{i,j-1/2}(u^n_{i,j} - u^n_{i,j-1})) + f^n_{i,j}$$

Løser for $u^{n+1}_{i,j}$ og får:

$$u^{n+1}_{i,j} = \frac{1}{2}\Big(2u^n_{i,j} - u^{n-1}_{i,j} + \frac{b\Delta t}{2}u^{n-1}_{i,j} + \frac{\Delta t^2}{\Delta x^2}(q_{i+1/2,j}(u^n_{i+1,j} - u^n_{i,j}) - q_{i-1/2,j}(u^n_{i,j} - u^n_{i-1,j})) + \frac{\Delta t^2}{\Delta y^2}(q_{i,j+1/2}(u^n_{i,j+1} - u^n_{i,j}) - q_{i,j-1/2}(u^n_{i,j} - u^n_{i,j-1})) + \Delta t^2 f^n_{i,j}\Big)$$

Fra initalbetingelsene har vi at:

$$u^0_{i,j} = I_{i,j}$$

$$D_{2t}u^0_{i,j} = V_{i,j} \Rightarrow u^{-1}_{i,j} = u^1_{i,j} - 2\Delta t V_{i,j}$$

Regner så ut algoritmen for første steg, n=0, og setter inn for initialbetingelsene:

$$u^1_{i,j} = \frac{1}{2}\Big(2I_{i,j} + 2\Delta t V_{i,j} - \frac{b\Delta t}{2} 2\Delta t V_{i,j} + \\ \frac{\Delta t^2}{\Delta x^2}(q_{i+1/2,j}(u^0_{i+1,j} - I_{i,j}) - q_{i-1/2,j}(I_{i,j} - u^0_{i-1,j})) + \\ \frac{\Delta t^2}{\Delta y^2}(q_{i,j+1/2}(u^0_{i,j+1} - I_{i,j}) - q_{i,j-1/2}(I_{i,j} - u^1_{i,j-1})) + \Delta t^2 f^0_{i,j}\Big)$$

Går videre med å beregne modifisert algoritme for grenseverdiene. Vi har:

$$\frac{\partial u}{\partial x} = 0, \quad \frac{\partial u}{\partial y} = 0$$

Og får at i grensene så gjelder:

$$u^n_{-1,j} = u^n_{1,j}, \quad u^n_{i,-1} = u^n_{i,1}$$

Setter dette inn i $u^1_{0,0}$:

$$u^1_{0,0} = \frac{1}{2}\Big(2I_{0,0} + 2\Delta t V_{0,0} - \frac{b\Delta t}{2} 2\Delta t V_{0,0} + \\ \frac{\Delta t^2}{\Delta x^2}(q_{1/2,0}(u^0_{1,0} - I_{0,0}) - q_{-1/2,0}(I_{0,0} - u^0_{-1,0})) + \\ \frac{\Delta t^2}{\Delta y^2}(q_{0,1/2}(u^0_{0,1} - I_{0,0}) - q_{0,-1/2}(I_{0,0} - u^0_{0,-1})) + \Delta t^2 f^0_{0,0}\Big)$$

$$= \frac{1}{2}\Big(2I_{0,0} + 2\Delta t V_{0,0} - \frac{b\Delta t}{2} 2\Delta t V_{0,0} + \\ \frac{\Delta t^2}{\Delta x^2}(q_{1/2,0}(u^0_{1,0} - I_{0,0}) - q_{-1/2,0}(I_{0,0} - u^0_{1,0})) + \\ \frac{1}{2}(q_{0,1/2}(u^0_{0,1} - I_{0,0}) - q_{0,-1/2}(I_{0,0} - u^0_{0,1})) + \Delta t^2 f^0_{0,0}\Big)$$

2.3 Implementasjon

Se kode i project1.py, hvor funksjonen solver er skalar-løsningen, mens solver_vec er den vektoriserte løsningen.

3 Verifikasjon

3.1 Se python funksjon test_case(), ved å kjøre:

```
if __name__=="__main__":
#    solver(I=lambda x,y:x,V=lambda x,y:0, q=lambda x,y:1 , b=10, f=lambda x,y,t
#    solver_vec(I=lambda x,y:x,V=lambda x,y:0, q=lambda x,y:1, b=10, f=lambda x,
    test_case()
#    plug_wave_test()
#    standing_undamped()
#    physical_problem('gaussian',True)
#    physical_problem('cosine_hat',True)
#    physical_problem('box',True)
```

3.2 Vi skal vise at den konstante løsningen av likning (1), $u(x, y, t) = c$ også stemmer for den diskretiserte likningen.

Vi har:

$$\frac{\partial^2 u}{\partial t^2} + b\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}(q(x,y)\frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(q(x,y)\frac{\partial u}{\partial y}) + f(x,y)$$

Og vi ser at hvis vi setter inn $u(x, y, t) = c$ så får vi:

$$0 = f(x, y)$$

Dersom jeg setter inn $u(x, y, t) = c$ i den diskretiserte løsningen:

$$[D_t D_t u + bD_t u = D_x(qD_x u) + D_y(qD_y u) + f]^n_{i,j}$$

Får jeg:

$$\frac{c - 2c + c}{\Delta t^2} + b\frac{c - c}{2\Delta t} = \frac{1}{\Delta x^2}(q_{i+1/2,j}(c - c) - q_{i-1/2,j}(c - c)) +$$
$$\frac{1}{\Delta y^2}(q_{i,j+1/2}(c - c) - q_{i,j-1/2}(c - c)) + f^n_{i,j}$$
$$=$$

$$\frac{}{\Delta t^2} + b\frac{0}{2\Delta t} = \frac{1}{\Delta x^2}(q_{i+1/2,j}(0) - q_{i-1/2,j}(0)) + \frac{1}{\Delta y^2}(q_{i,j+1/2}(0) - q_{i,j-1/2}(0)) + f^n_{i,j}$$

$$0 = f_{i,j}^n$$

Og dermed ser vi at den konstante løsningen er en løsning for den diskretiserte likningen.

3.3 Et kjøreeksempel av nosetest for test_case():

```
(C:\Program Files\Anaconda2) C:\Users\laila\INF5620\Mandatory project 1>nosetest
.
----------------------------------------------------------------------
Ran 1 test in 22.666s

OK
```

3.4 Selvkostruerte bugs

1)Endrer funksjonen for den eksakte løsningen:

```
u_const = lambda x,y,time:c*2
```

2)Endrer initialfunksjonen til:

```
I=lambda x,y:c*2
```

3)Hvis jeg endrer initialhastigheten til å retunere en ikke-null konstant:

```
V=lambda x,y:2
```

4)Det samme med funksjonen for f:

```
f=lambda x,y,t:2
```

5)Hvis jeg endrer c til å bli for stor:

```
 c = 5
```

Alle disse gir dermed følgende feil:

```
(C:\Program Files\Anaconda2) C:\Users\laila\INF5620\Mandatory project 1>python p
Traceback (most recent call last):
  File "project1.py", line 380, in <module>
    test_case()
  File "project1.py", line 215, in test_case
    assert diff<tol, "Error in contant solution"
AssertionError: Error in contant solution
```

3.3 Eksakt 1D plug-bølge løsning i 2D

Se python funksjonen: plug_wave_test()

Her får jeg feil, ettersom den numeriske og eksakte ikke stemmer.

3.4 Stående, udempede bølger

$$u_e(x,y,t) = A\cos(k_x x)cos(k_y y)cos(\omega t),\ \ k_x = \frac{m_x\pi}{L_x}, k_x = \frac{m_y\pi}{L_y}$$

Se funskjonen standing_undamped() i project1.py

Jeg får følgende resultater:

```
E: [1.5005287285416491, 1.5005287285416491, 1.5005287285416491, 1.500528728541649
r: [-0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
h [0.1, 0.05, 0.025, 0.0125, 0.00625, 0.003125, 0.0015625]
```

4 Undersøk et fysisk problem

For å kjøre funskjonen for en Gaussisk bakke under vannet må man kjøre:

```
if __name__=="__main__":
#    solver(I=lambda x,y:x,V=lambda x,y:0, q=lambda x,y:1 , b=10, f=lambda x,y,t
#    solver_vec(I=lambda x,y:x,V=lambda x,y:0, q=lambda x,y:1, b=10, f=lambda x,
#    test_case()
#    plug_wave_test()
#    standing_undamped()
    physical_problem('gaussian',True)
#    physical_problem('cosine_hat',True)
#    physical_problem('box',True)
```

5 Konklusjon

Siden jeg ikke fikk til å animere i 3D har jeg animasjoner som viser for konstant y=1, og ser dermed kun på hvordan x oppfører seg over tid. Jeg kunne også gjøre motsatt; se hvordan y oppfører seg over tid med en konstant x, men ikke begge deler.

Det største problemet var å lage en solver som kunne brukes i alle de forskjellige oppgavene. Solveren kunne fungere for en oppgave men ikke for en annen.