Ickelinjära ekvationer 1

Ickelinjär reaktion: Har tidigare betraktat reaktions-diffusionsekvationer av typen

$$\dot{u} - \nabla \cdot a\nabla u = f - c\,u,$$

med givna begynnelse och randvillkor för $u = u(x,t)$, och med $a = a(x,t)$, $f = f(x,t)$ och $c = c(x,t)$ givna data. Skall närmast generalisera till ekvationer av typen

$$\dot{u} - \nabla \cdot a\nabla u = f$$

med $f = f(x,t,u(x,t))$, där f är en given funktion som nu även tillåts *bero på u*, dvs $f : \Omega \times R_+ \times R \to R$. Notera att *linjära* reaktions-diffusionsekvationer här ingår som specialfall.

Ickelinjära ekvationer 2

För att förenkla beteckningarna betraktar vi inledningsvis ekvationer med $f = f(u)$, dvs utan *givet* beroende av x och t, men där f alltså beror på $u = u(x,t)$ (och därmed indirekt på x och t). Exempel skulle kunna vara $f(u) = u^2$ och $f(u) = e^u$. Exempel av den mera allmänna typen med givet beroende även på x och/eller t skulle kunna vara $f = x\,u^2$ och $f = e^{-t/u}$.

Ickelinjära ekvationer 3

Modell ekvation: Som tidigare betraktar vi även *modellekvationen*

$$\dot{u} + a\,u = f,$$

där $u = u(t)$, a är konstant, och, här, $f = f(u)$ (eller ev $f = f(t,u)$).

Explicit Euler för denna ekvation reduceras till

$$U_n = U_{n-1} - k\,a\,U_{n-1} + k\,f_{n-1}.$$

där nu $f_{n-1} = f(U_{n-1})$ och (förhoppningsvis) $U_n \approx u(t_n)$.

Ickelinjära ekvationer 4

Metoden kan förstås generaliseras till den betraktade pde-modellen, dvs reaktions-diffusionsekvationen, genom

$$M\,U_n = M\,U_{n-1} - k\,A\,U_{n-1} + k\,F_{n-1},$$

där F_{n-1} nu är vektorn med element $\int_\Omega \phi_i\,f(U_{n-1})$.

Genom att ersätta massmatriserna M med motsvarande *lumpade* matris $\bar{M}$ och sedan multiplicera med dess invers erhålls en fullt explicit metod även för det givna ickelinjära problemet:

$$U_n = U_{n-1} - k\,\tilde{a}\,U_{n-1} + k\,\tilde{f}_{n-1},$$

där $\tilde{a} = \bar{M}^{-1}A$ och $\tilde{f}_{n-1} = \bar{M}^{-1}F_{n-1}$.

Ickelinjära ekvationer 5

Implicit Euler: Vi betraktar nu motsvarande *implicita* Eulermetod

$$U_n + k\,a\,U_n = U_{n-1} + k\,f(U_n).$$

Vi noterar att vi nu inte längre kan lösa ut U_n genom en enkel division med $1 + k\,a$ som i fallet med givet $f = f(t)$. Istället har vi nu en *icklinjär* ekvation att lösa.

Ickelinjära ekvationer 6

I "undantagsfall" kan vi tänkas kunna lösa ut U_n "analytiskt", dvs genom formelmanipulation. Om t.ex. $f(u) = u^2$, så har vi ju

$$(1 + k\,a)\,U_n - k\,U_n^2 = U_{n-1},$$

med lösning

$$U_n = \frac{1 + k\,a}{2\,k} \pm \sqrt{(\frac{1 + k\,a}{2\,k})^2 - \frac{U_{n-1}}{k}}.$$

Sedan kvarstår förstås att beräkna roten, och välja rätt tecken på densamma.

Ickelinjära ekvationer 7

Än mera problematiskt att lösa de resulterande ekvationerna blir det förstås för motsvarande pde:

$$M\,U_n + k\,A\,U_n = M\,U_{n-1} + k\,F(U_n),$$

där $F(U_n) = F_n$ betecknar vektorn med komponenter $\int_\Omega \phi_i\,f(U_n)$.

Ickelinjära ekvationer 8

Fixpunktsiteration: Redan i ALA-a löste vi denna typ av ickelinjära ekvationssystem först med *fixpunktsiteration* och sedan med *Newtons metod*. Vi erinrar oss att fixpunktsiteration tillämpas på ekvationer på formen $u = g(u)$ och skriver därför om vår ekvation som

$$U_n = (1 + k\,a)^{-1}\,(U_{n-1} + k\,f(U_n)) =: g(U_n).$$

För att hitta lösningen U_n till denna ekvation startar vi från lämplig punkt/vektor $U_n^{(0)}$ och beräknar succesivt

$$U_n^{(j+1)} = g(U_n^{(j)}) = (1 + k\,a)^{-1}\,(U_{n-1} + k\,f(U_n^{(j)})).$$

Ickelinjära ekvationer 9

För motsvarande pde beräknar vi förstås

$$U_n^{(j+1)} = (M + k\,A)^{-1}\,(U_{n-1} + k\,F(U_n^{(j)})).$$

Ickelinjära ekvationer 10

Nu inställer sig en rad frågor.

- Hur väljer man lämplig $U_n^{(0)}$?
- Kan man vänta sig konvergens?
- Hur många iterationssteg kan behövas?

Den första frågan är ganska lätt att besvara. Det är förstås högst rimligt att välja $U_n^{(0)} = U_{n-1}$!

I vissa fall kan man kanske hitta ännu bättre utgångspunkter, men den föreslagna är i alla fall klart överlägsen nollvektorn eller en slumpmässigt vald startvektor.

Ickelinjära ekvationer 11

Nästa fråga gällde konvergens. Vi erinrar oss att fixpunktsiteration konvergerar under förutsättning att g är *kontraktiv*, dvs att g är Lipschitz med Lipschitzkonstant är mindre än 1.

Turligt nog kan denna typ av g väntas ha just denna (något speciella) egenskap. I alla fall om man väljer tillräckligt korta tidssteg, vilket man ofta ändå vill göra av noggrannhetsskäl.

Ickelinjära ekvationer 12

För att “se” detta noterar vi att

$$\begin{aligned} g(v) - g(w) &= (1 + k\,a)^{-1}\,(U_{n-1} + k\,f(v)) \\ &\quad -(1 + k\,a)^{-1}\,(U_{n-1} + k\,f(w)) \\ &= (1 + k\,a)^{-1}\,(k\,f(v) - k\,f(w)), \end{aligned}$$

dvs för $a \geq 0$

$$|g(v) - g(w)| \leq k\,L_f\,|v - w|,$$

där L_f är en Lipschitz konstant för f inom aktuellt intervall. Vi ser här att om bara $k < 1/L_f$ så blir g en kontraktion och vi kan förvänta oss konvergens. Detta låter sig också direkt generaliseras till pde-fallet.

Ickelinjära ekvationer 13

Så till frågan om lämpligt antal iterationer.

Naturligtvis kan vi i varje tidssteg iterera “till konvergens”, men detta kan vara ganska onödigt med tanke på att vi ändå måste räkna med ett resulterande *diskretiseringsfel* av storleksordning k^2 i varje tidssteg (eller mera bestämt $k^2\,\dot{u}/2$).

Ickelinjära ekvationer 14

Om vi nu jämför den lösning $U_n^{(1)}$ vi erhållit redan efter *en* iteration startande från $U_n^{(0)} = U_{n-1}$, dvs $\tilde{U}_n = g(U_{n-1})$, och jämför denna med den “färdigitererade” lösningen $U_n = g(U_n)$ så fås

$$\begin{aligned} |\tilde{U}_n - U_n| &= |g(U_{n-1}) - g(U_n)| \\ &\le k\,L_f\,|U_{n-1} - U_n| \le k\,L_f\,k\,|f - a\,U_n|, \end{aligned}$$

Ickelinjära ekvationer 15

dvs det kvarvarande *iterationsfelet* i $U_n^{(j)}$ är redan efter en iteration reducerat till storleksordning k^2, som för *diskretiseringsfelet*, dvs felet orsakat av Eulerapproximationen i sig. Om vi därför, för säkerhets skull, genomför *två* iterationer i varje tidssteg, dvs beräknar först $\tilde{U}_n$ och sedan $U_n = g(\tilde{U}_n)$ så borde iterationsfelet säkert domineras av diskretiseringsfelet i de flesta fall.

I mera komplicerade fall, eller beräkning baserad på en mera noggrann diskretiseringsmetod som t.ex. CN, bör man förstås även iterera noggrannare.

Ickelinjära ekvationer 16

Newtons metod: Vi erinrar oss att vi även kan lösa ickelinjära ekvationssystem med den mera sofistikerade *Newtons metod*. Vi skriver då ekvationen vi vill lösa som

$$G(U_n) := U_n + k\,a\,U_n - U_{n-1} - k\,F(U_n) = 0,$$

där vi alltså söker U_n. Vi ersätter här $G(U)$ med den *linjära tangentplansapproximationen* $G(U_n^{(j)}) + G'(U_n^{(j)})\,(U - U_n^{(j)})$ och söker alltså $U_n^{(j+1)}$ så att

$$G(U_n^{(j)}) + G'(U_n^{(j)})\,(U_n^{(j+1)}) - U_n^{(j)}) = 0,$$

dvs

Ickelinjära ekvationer 17

löser

$$G'(U_n^{(j)})\,(U - U_n^{(j)}) = -G(U_n^{(j)}),$$

och sätter $U_n^{(j+1)} = U$.

Återigen kan vi ta $U_n^{(0)} = U_{n-1}$ som utgångspunkt, och förutsatt att tidssteget är litet kan vi räkna med “konvergens i ett steg”, dvs tillräcklig noggrannhet redan efter ett iterationssteg i många fall.

Ickelinjära ekvationer 18

Om vi nu generaliserar till pde-fallet så vill vi hitta vektorn U_n så att

$$G(U_n) := M\,U_n + k\,A\,U_n - M\,U_{n-1} - F(U_n) = 0,$$

och varje Newtonsteg reduceras till att lösa

$$G'(U_n^{(j)})\,X = -G(U_n^{(j)}),$$

och sedan sätta $U_n^{(j+1)} = U_n^{(j)} + X$, där G' är Jacobianmatrisen med gradienten av komponent i i G i rad i.

Ickelinjära ekvationer 19

Vi noterar att i motsvarande linjära fall med $G(U) = M\,U + k\,A\,U - M\,U_{n-1} - F$ så reduceras G' till $M + k\,A$, vilket anknyter till den ekvationslösning vi hamnade i vid fixpunktsiteration.

Ickelinjära ekvationer 20

Resulterande kod: För explicit Euler räcker det att vi i varje tidssteg beräknar vektorn $F_{n-1} = F(U_{n-1})$. Detta kan göras genom att skicka med aktuell lösningsvektor U_{n-1} som indata till din `MyLoadVectorAssembler.m` som modifieras så att den med `F = MyLoadVectorAssembler(p,e,t,U)` returnerar den önskade vektorn $F(U)$, för att sedan användas i tidsloopen:

```
while time < finaltime
 F=MyLoadVectorAssembler(p,t,e,U);
 U=U-k*A*U+k*F;
 time=time+k;
end
```

Ickelinjära ekvationer 21

För implicit Euler krävs alltså någon form av ekvationslösning i varje tidssteg. Enklast “fuskar” vi helt enkelt med att iterera “till konvergens”, och nöjer oss med två fixpunktsiterationer i varje tidssteg:

```
while time < finaltime
 W=U;
 time=time+k;
 FU=MyLoadVectorAssembler(p,t,e,U);
 U=(M+k*A)\(M*W+k*FU);
 FU=MyLoadVectorAssembler(p,t,e,U);
 U=(M+k*A)\(M*W+k*FU);
end
```

Notera att vi här kallat U_{n-1} för W för att undvika sammanblandning med “iteranderna” $U = U_n^{(j)}$, $j = 1, 2$.

Ickelinjära ekvationer 22

Generalisering till CN/cG1: Vi påminner oss om CN för modellekvationen $\dot{u} + a\,u = f$:

$$(1 + \frac{k}{2}\,a)\,U_n = (1 - \frac{k}{2}\,a)\,U_{n-1} + \frac{k}{2}\,f_{n-1} + \frac{k}{2}\,f_n,$$

som på samma sätt som implicit Euler kan generaliseras till fallet $f = f(u)$ genom att sätta $f_{n-1} = f(U_{n-1})$ och $f_n = f(U_n)$.

Motsvarande för pde-fallet blir förstås

$$(M + \frac{k}{2}\,A)\,U_n = (M - \frac{k}{2}\,A)\,U_{n-1} + \frac{k}{2}\,F(U_{n-1}) + \frac{k}{2}\,F(U_n).$$

Ickelinjära ekvationer 23

Matlab kod för detta skulle kunna se ut som följer, i princip:

```
while time < finaltime
 W=U;
 time=time+k/2;
 FW=MyLoadVectorAssembler(p,t,e,W);
 U=(M+k/2*A)\((M-k/2*A)*W+k/2*FW+k/2*FW);
 FU=MyLoadVectorAssembler(p,t,e,U);
 U=(M+k/2*A)\((M-k/2*A)*W+k/2*FW+k/2*FU);
 FU=MyLoadVectorAssembler(p,t,e,U);
 U=(M+k/2*A)\((M-k/2*A)*W+k/2*FW+k/2*FU);
 time=time+k/2;
end
```