

System 1

System av reaktions-diffusionsekvationer:

Betraktar inledningsvis **system** av (två) ekvationer av typ

$$\dot{u}_i - \nabla \cdot a_i \nabla u_i = f_i(u) \quad i = 1, 2,$$

där $u_i = u_i(x, t)$ och $u = (u_1, u_2)$, dvs

$$\begin{cases} \dot{u}_1 - \nabla \cdot a_1 \nabla u_1 = f_1(u_1, u_2) \\ \dot{u}_2 - \nabla \cdot a_2 \nabla u_2 = f_2(u_1, u_2), \end{cases}$$

med givna $a_i = a_i(x) > 0$ och $f_i = f_i(u)$, **begynnelsevillkor** $u_i = u_{i,0}$ för $t = 0$, och **randvillkor**

$$-a_i \partial_n u_i = \gamma_i (u_i - g_{i,D}) + g_{i,N} \quad i = 1, 2.$$

-- P.1/19

System 3

Diffusivetskoefficienterna a_i modellerar förstås arternas benägenhet att vilja "flytta" eller "sprida sig".

Randvillkoren kan vara av olika typ. Längs en rand t.ex. mot ett "stort vatten" bör gälla att $\gamma_i = 0$ och $g_{i,N} = 0$, eftersom varken rävar eller harar (inte ens jagade) gärna ger sig ut på långa simturer. Längs en rand t.ex. mot en hårt trafikerad väg (utan viltstängsel, och med attraktiv mark på andra sidan) kanske snarare gäller att $u_i = 0$, dvs alla djur som kommer ut på vägen stryker med, vilket ju kan modelleras med $\gamma_i \gg 1$, $g_{i,D} = 0$, och $g_{i,N} = 0$.

-- P.3/19

System 2

Exempel rovdjur-byte: Låt u_1 och u_2 vara antal **harar** respektive **rävar**, per hektar, i ett (tvådimensionellt) "skogsområde" Ω . Notera att $u_i = u_i(x, t)$ här representerar **lokala** densiteter av respektive djurart. En första (grov) modell för samspelet/växelverkan mellan de två djurpopulationerna skulle kunna vara :

$$\begin{cases} \dot{u}_1 - \nabla \cdot a_1 \nabla u_1 = c_1 u_1 (\bar{u}_2 - u_2) \\ \dot{u}_2 - \nabla \cdot a_2 \nabla u_2 = c_2 u_2 (u_1 - \bar{u}_1), \end{cases}$$

där $c_i > 0$ och $\bar{u}_i > 0$ är givna konstanter, och där (något förenklat) \bar{u}_2 kan ses som den **kritiska rävtäthet** vid vilken hararna lyckas reproducera sig i samma takt som de blir konsumerade, och \bar{u}_1 som den **kritiska harstäthet** vid vilken de precis utgör tillräcklig föda för rävarna.

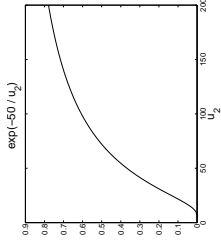
-- P.2/19

System 4

Exempel förbränning: Låt u_1 beteckna **koncentrationen** av ett ämne som kan konsumeras genom **förbränning**, och låt u_2 beteckna **temperaturen**. Förbränningshastigheten kan förstås antas proportionell mot tillgänglig mängd/koncentration av det brännbara ämnet, dvs mot u_1 . Proportionalitetsfaktorn är oftast temperaturberoende, och växer med växande temperatur, från 0 vid "noll-temperatur". Ett kvantitet som har denna egenskap skulle kunna vara u_2^r med lämpligt $r > 0$. En mera verklighetsnära modell erhålls med proportionalitetsfaktorn $c_1 \exp(-\alpha/u_2)$ för lämpliga $c_1 > 0$ och $\alpha > 0$.

-- P.4/19

System 5



Värmeproduktionen bör förstås också vara proportionell mot förbränningsintensiteten, och därmed erhålls systemet

$$\begin{cases} u_1 - \nabla \cdot a_1 \nabla u_1 = -c_1 \exp(-\alpha/u_2) u_1 \\ u_2 - \nabla \cdot a_2 \nabla u_2 = c_2 \exp(-\alpha/u_2) u_1. \end{cases}$$

-- p.5/19

System 8

Matlab implementering: En implementering i Matlab skulle nu i princip kunna se ut som följer:

```
while time < finaltime
    F1=MyLoadVectorAssembler1(p,t,e,U1,U2);
    F2=MyLoadVectorAssembler2(p,t,e,U1,U2);
    U1=M \ ( (M-k*A1) *U1+k*F1 );
    U2=M \ ( (M-k*A2) *U2+k*F2 );
    time=time+k;
end
```

där vi påminner om möjligheten att **effektivisera** genom att **förfaktorisera** massmatrisen M som $L U$, och ersätta $M \setminus$ med $L \setminus U \setminus$, eller hellre $\bar{M} \setminus$, med den masslumpade approximationen \bar{M} av M .

-- p.7/19

System 6

Diskretisering: Vi diskretiserar förstås varje ekvation för sig med ansatser $U_{1n}(x) = \sum_j U_{1n,j} \phi_j(x) \approx u_1(x, t_n)$ resp $U_{2n}(x) = \sum_j U_{2n,j} \phi_j(x) \approx u_2(x, t_n)$. Koefficienterna $U_{1n,j}$ och $U_{2n,j}$ bestäms med explicit Euler-tidsstegning av

$$\begin{cases} M U_{1n} = (M - k A1) U_{1n-1} + k F1(U_{1n-1}, U_{2n-1}) \\ M U_{2n} = (M - k A2) U_{2n-1} + k F2(U_{1n-1}, U_{2n-1}), \end{cases}$$

där A_i är respektive diffusionsmatriser, i vilka även kan tänkas ingå randdiffusionsmatriserna K_i , och där $F1$ och $F2$ är resp lastvektorer beroende på såväl U_{1n-1} som U_{2n-1} , och i vilka även kan tänkas ingå randlastvektorena härlhörande från $g_{i,D}$ och $g_{i,N}$ -termerna i de fall dessa är $\neq 0$.

-- p.6/19

System 9

För att undvika den uppenbara nackdelen att behöva **dubbla** assemblerkoden kan vi istället skicka med namnet på funktionsfilerna där respektive $f_i(U)$ beräknas som indata. Koden ersätts då med

```
while time < finaltime
    F1=MyLoadVectorAssembler(p,t,e,f1,U1,U2);
    F2=MyLoadVectorAssembler(p,t,e,f2,U1,U2);
    U1=M \ (MU1-k*A1 *U1+k*F1 );
    U2=M \ (MU2-k*A2 *U2+k*F2 );
    time=time+k;
end
```

-- p.8/19

System 10

där `MyLoadVectorAssembler` kan tänkas ha formen

```
function
[F]=MyLoadVectorAssembler(p,t,e,file,U1,U2)
for el=1:size(p,2)
    ·
    u1=..
    u2=..
    f=feval(file,u1,u2);
    ·
end
```

där `u1` och `u2` är kvadraturpunktvärden beräknade från `U1` och `U2`, och ..

-- p.9/19

System 12

Implicit Euler: Motsvarande kod för *implicit Euler* skulle kunna se ut som följer:

```
while time < finaltime
    W1=U1;
    W2=U2;
    time=time+k;
    F1U=MyLoadVectorAssembler(p,t,e,f1,U1,U2);
    F2U=MyLoadVectorAssembler(p,t,e,f2,U1,U2);
    U1=(M+k*A1)\(M*W1+k*F1U);
    U2=(M+k*A2)\(M*W2+k*F2U);
    F1U=MyLoadVectorAssembler(p,t,e,f1,U1,U2);
    F2U=MyLoadVectorAssembler(p,t,e,f2,U1,U2);
    U1=(M+k*A1)\(M*W1+k*F1U);
    U2=(M+k*A2)\(M*W2+k*F2U);
end
```

-- p.11/19

System 11

där `f1` motsvaras av en fil `f1.m`, som i "kaninekvationen", för att ta ett konkret exempel, skulle kunna innehålla

```
function f=f1(u1,u2)
    u1*(3-u2);
```

där vi antagit att $c_1 = 1$ och $\bar{u}_2 = 3$, som ett exempel.

-- p.10/19

System 13

Fler ekvationer: Vi fler än c:a 3 ekvationer och obekanta u_i börjar det bli obekvämt att i koden adressera dessa med individuella namn som $U1, U2, \dots$. Istället kan man då tänka sig att se lösningen som en vektor av vektorer, dvs som en matris, och låta $U(:, 1) = U1, U(:, 2) = U2$ osv. Vi har redan i det skalära fallet lagrat Lösningsvektorns "tidsutveckling" på detta sätt. För att kunna lagra tidsutvecklingen av en matrisen $U = [U1U2\dots]$ kan man använda matlabs "tredimensionella" matrishantering, dvs lagra t.ex. $U2$ vid tid t_{n-1} som $U(:, 2, n)$.

-- p.12/19

System 14

Ickelinjär entalpi och diffusion:

Erinrar oss att värmeledningsekvationen tar formen

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

där $u = u(x, t)$ är temperaturen, och där c är värmekapacitivitet (mätt i J/m^3), a konduktiviteten, och f värmeproduktionen.

Har redan noterat att värmeproduktion f naturligen kan bero på u , dvs $f = f(u)$. I allmänhet gäller även att $c = c(u)$ och $a = a(u)$, dvs även entalpin och diffusionen är icke linjära i u .

--p.13/19

System 16

Vid tidsstegning med små tidssteg bör man ändå kunna tänka sig att c och a kan betraktas som konstanta inom respektive tidsintervall, och helt enkelt kunna approximeras med deras värden vid ingångstiden t_{n-1} . Med explicit Euler ger detta tidsstegningsmetoden

$$M_{n-1} U_n = (M_{n-1} - k A_{n-1}) U_{n-1} + k F_{n-1},$$

där M_{n-1} är massmatrisen med element $\int_{\Omega} \phi_i c(U_{n-1}) \phi_j$, A_{n-1} är diffusionsmatrisen med element $\int_{\Omega} \nabla \phi_i \cdot a(U_{n-1}) \nabla \phi_j$, och, som tidigare, F_{n-1} är lastvektorn med element $\int_{\Omega} \phi_i f(U_{n-1})$.

--p.15/19

System 15

Vi erinrar oss att entalpin $Q = Q(u)$ kan definieras som $Q = \int_0^u c(v) dv$, och att värmeledningsekvationen då tar formen

$$\underbrace{\dot{Q}}_{c(u)\dot{u}} - \nabla \cdot a(u) \nabla u = f.$$

T.ex. gäller för vatten i fast fas (dvs is) att

$$c = (0.00716u + 0.138) 920 \text{ och}$$

$$a = 0.00224 + 0.00000593 (273 - u)^{1.156}.$$

Vid små temperaturvariationer kan såväl c som a betraktas som konstanta, men vid större variationer, eller större krav på noggrannhet, kan krävas att c 's och a 's u -beroende tas med i beräkningen.

--p.14/19

System 17

Vågekvationen på systemform: **Vågekvationen** kan skrivas

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

där $u = u(x, t)$ representerar en lägeskoordinat, c densitet, a materialets "elasticitetsmodul", och f en (yttre) verkande kraft.

Till detta kommer begynnelsevillkor för såväl u som \dot{u} , och randvillkor som vi, för enkelhets skull antar har formen $-a \partial_n u = 0$.

--p.16/19

System 18

Vi inför hastigheten $v = \dot{u}$ som en ny (beroende) variabel och skriver ekvationen på systemform som

$$\begin{cases} \dot{u} - v = 0 \\ c \dot{v} - \nabla \cdot a \nabla u = f. \end{cases}$$

--p.17/19

System 20

För att kunna beräkna U_n och V_n skriver vi systemet på blockmatrisformen

$$= \underbrace{\begin{bmatrix} M & -\frac{k}{2}M \\ \frac{k}{2}A & M_c \end{bmatrix}}_H \underbrace{\begin{bmatrix} U_n \\ V_n \end{bmatrix}}_W + \underbrace{\begin{bmatrix} U_{n-1} \\ V_{n-1} \end{bmatrix}}_B + \begin{bmatrix} 0 \\ \frac{k}{2}(F_{n-1} + F_n) \end{bmatrix}$$

ur vilket vi kan lösa ut W_n och därmed U_n och V_n .

--p.19/19

System 19

Diskretisering: Vi diskretiserar med cG1 och får:

$$\begin{cases} M(U_n - U_{n-1}) - \frac{k}{2}M(V_{n-1} + V_n) = 0 \\ M_c(V_n - V_{n-1}) + \frac{k}{2}A(U_{n-1} + U_n) = \frac{k}{2}(F_{n-1} + F_n), \end{cases}$$

där M_c är massmatrisen med element $\int_{\Omega} \phi_i c \phi_j$, och M motsvarande matris med $c = 1$.

--p.18/19