Ordinära differentialekvationer

Analys och Linjär Algebra, del B, K1/Kf1/Bt1

1 Inledning

I ett par studio-övningar ska vi lära oss hur man konstruerar och beräknar lösningar till ordinära differentialekvationer (ODE) av typen

$$u'(x) = f(x, u(x))$$

Vi börjar med det enklaste fallet då $f = f(x)$ är en funktion som bara beror på x. Sedan låter vi f bero också på u och till sist låter vi $\boldsymbol{u}$ och $\boldsymbol{f}$ vara vektorfunktioner, dvs. vi löser system av ordinära differentialekvationer.

2 Primitiv funktion

Låt $I = [a, b]$ vara ett intervall och antag att $f : I \to \mathbb{R}$ är en kontinuerlig funktion. Om funktionen u är deriverbar och uppfyller

$$u'(x) = f(x), \quad x \in I, \tag{1}$$

så säger vi att u är en *primitiv funktion* till f (antiderivative). Vi ska konstruera primitiva funktioner. Vi börjar med att påminna om att en primitiv funktion är väsentligen unik. För om två funktioner u och v är primitiva funktioner till samma funktion f så gäller

$$(u - v)'(x) = u'(x) - v'(x) = f(x) - f(x) = 0, \quad x \in I.$$

Härav följer att $u - v =$ konstant, dvs. $u(x) = v(x) + C,\ x \in I$.

För att bestämma konstanten kan man kräva att funktionen skall vara lika med ett bestämt värde i en bestämd punkt: $u(a) = u_a$. Vi kan nu precisera vår uppgift. Givet en kontinuerlig funktion $f : [a, b] \to \mathbb{R}$ och en konstant u_a söker vi en deriverbar funktion u sådan att

$$\begin{cases} u'(x) = f(x), & x \in [a, b] \\ u(a) = u_a. \end{cases} \tag{2}$$

Eftersom vi anger funktionens värde i intervallets vänstra ändpunkt så kallas problemet (2) ett *begynnelsevärdesproblem*. Enligt integralkalkylens huvudsats så är

$$F(x) = \int_a^x f(t)\, dt, \quad x \in I, \tag{3}$$

en primitiv funktion till f. Därför gäller att varje lösning u till (2) beskrivs av $u(x) = F(x) + C$ där C är en konstant. Sätter vi nu in $x = a$ så finner vi att

$$u_a = u(a) = \int_a^a f(t)\, dt + C = 0 + C = C.$$

Därför är

$$u(x) = u_a + \int_a^x f(t)\,dt, \quad x \in [a,b]$$

den entydiga lösningen till (2).

2.1 Konstruktion

Vi ska nu beskriva hur man kan konstruera en primitiv funktion för varje kontinuerlig funktion f, dvs. konstruera en unik lösning till begynnelsevärdesproblemet (2).
Vi börjar med att dela in intervallet $[a,b]$ i N stycken delintervall av längden $h = (b-a)/N$:

$$a = x_0 < x_1 < x_2 < \cdots < x_{i-1} < x_i < \cdots < x_{N-1} < x_N = b,$$
$$x_i = a + hi,\ h = (b-a)/N = x_i - x_{i-1}.$$

Vi har att

$$u(x_i) = u_a + \int_a^{x_i} f(t)\,dt = u_a + \int_a^{x_{i-1}} f(t)\,dt + \int_{x_{i-1}}^{x_i} f(t)\,dt =$$
$$= u(x_{i-1}) + \int_{x_{i-1}}^{x_i} f(t)\,dt$$

Om vi låter $U(x_i)$ beteckna approximationen av $u(x_i)$ och om vi approximerar integralen med *vänster rektangelregel* så får vi **Eulers framåtmetod**

$$U(x_i) = U(x_{i-1}) + hf(x_{i-1})$$

och med *höger rektangelregel* så får vi **Eulers bakåtmetod**

$$U(x_i) = U(x_{i-1}) + hf(x_i)$$

Med *mittpunktsregeln* så får vi **mittpunktsmetoden**

$$U(x_i) = U(x_{i-1}) + hf\left(\frac{x_{i-1}+x_i}{2}\right)$$

och med *trapetsregeln* så får vi **trapetsmetoden**

$$U(x_i) = U(x_{i-1}) + \frac{h}{2}(f(x_{i-1}) + f(x_i))$$

Genom att förbinda punkterna $(x_i, U(x_i))$ med räta linjer får vi en graf och funktionen $U(x)$ blir definierad också mellan beräkningsnoderna x_i.

Man kan visa att $U(x)$ konvergerar mot en unik lösning $u(x)$ till (2) då antalet delintervall $N \to \infty$. Detta gäller för alla metoderna ovan.

2.2 Program i MATLAB

Detta är lätt att programmera i MATLAB. Algoritmerna har likheter med de för `min_integral` som ni jobbade med förra veckan. Eftersom vi behöver både x_i och $U(x_i)$ för att till exempel plotta funktionen så måste algoritmen räkna ut även noderna x_i. Vi ser på Eulers framåtmetod.

Initiera:

$$x_0 = a, \quad U(x_0) = u_a$$

Uppdatera: För $i = 1, 2, \cdots, N$

$$x_i = x_{i-1} + h, \quad U(x_i) = U(x_{i-1}) + hf(x_{i-1})$$

I MATLAB kan index inte börja med 0, så att till exempel initieringen av kolonnvektorn x skrivs `x(1)=a`. Likaså måste $U(x_i)$ representeras av en kolonnvektor `U` med elementen `U(i)`. Man kan inte skriva `U(x(i))`.

Uppgift 1. Skriv ett program `min_primitiv` med anropet `[x,U]=min_primitiv(f,I,ua,h)` som löser begynnelsevärdesproblemet (2) med Eulers framåtmetod. Du skall använda programskalet: `min_primitiv.m` (se studiohemsidan).

Uppgift 2. Testa programmet på följande. Lös först begynnelsevärdesproblemet analytiskt (dvs. som en formel med penna och papper). Plotta både den analytiska lösningen u och den approximativa lösningen U i samma figur. Använd både ett stort steg h, till exempel, $h = 10^{-1}$, och ett litet steg h, till exempel, $h = 10^{-3}$.

(a). $\begin{cases} u'(x) = x^2, & x \in [0, 2], \\ u(0) = 3. \end{cases}$ **(b).** $\begin{cases} u'(x) = x^2, & x \in [1, 4], \\ u(1) = 3. \end{cases}$

(c). $\begin{cases} u'(x) = \cos(2x), & x \in [0, \pi], \\ u(0) = 0. \end{cases}$ **(d).** $\begin{cases} u'(x) = 1/x, & x \in [1, 10], \\ u(1) = 0. \end{cases}$

Observera att naturliga logarimen $\ln(x)$ heter `log(x)` i MATLAB.

Uppgift 3. Vi får en betydligt noggrannare approximation om vi använder trapetsmetoden. Modifiera programmet `min_primitiv` så att det även kan använda trapetsmetoden, genom att introducera en variabel `k` för att välja metod. Testa det nya programmet på samma exempel som ovan.

3 Allmän ODE

Vi ska nu behandla differentialekvationer där $f(x, u)$ är en allmän funktion av både x och u. Till exempel,

$$\begin{cases} u'(x) = -xu(x), \ x \in [0, 3], \\ u(0) = 1, \end{cases}$$

med analytisk lösning $u(x) = \exp(-x^2/2)$. Här ges alltså $f : \mathbb{R}^2 \to \mathbb{R}$ av formeln $f(x, u) = -xu$. Vi ska även studera system av sådana ekvationer. I kemi-tillämpningarna är den oberoende variabeln oftast tiden, så vi kommer beteckna den med t istället för x nedan.

3.1 Konstruktion

Vi ska nu beskriva hur man kan konstruera en lösning u för varje kontinuerlig funktion f, dvs. konstruera en unik lösning till begynnelsevärdesproblemet

$$\begin{cases} u'(t) = f(t, u(t)), & t \in [a, b], \\ u(a) = u_a. \end{cases} \tag{4}$$

För att lösa detta använder vi **Eulers framåtmetod**. Vi börjar med att dela in intervallet $[a, b]$ i N stycken delintervall av längden $h = (b - a)/N$:

$$a = t_0 < t_1 < t_2 < \cdots < t_{i-1} < t_i < \cdots < t_{N-1} < t_N = b,$$
$$t_i = a + hi,\ h = (b - a)/N = t_i - t_{i-1}.$$

Vi beräknar nu en approximativ lösning enligt

$$U(t_0) = u_a$$
$$U(t_i) = U(t_{i-1}) + hf(t_{i-1}, U(t_{i-1})).$$

Genom att förbinda punkterna $(t_i, U(t_i))$ med räta linjer får vi en graf och funktionen $U(t)$ blir definierad också mellan beräkningsnoderna t_i. På motsvarande sätt kan vi använda de andra metoderna som vi såg på för primitiva funktioner.

3.2 Program i MATLAB

Detta är lätt att programmera. Algoritmen är bara en liten förändring av den för `min_primitiv`. Vi ser på Eulers framåtmetod.

Initiera:

$$t_0 = a,\quad U(t_0) = u_a$$

Uppdatera: För $i = 1, 2, \cdots, N$

$$t_i = t_{i-1} + h,\quad U(t_i) = U(t_{i-1}) + hf(t_{i-1}, U(t_{i-1}))$$

I MATLAB måste $U(t_i)$ representeras av en vektor `U` med N komponenter. Med andra ord, `U(i)` skall innehålla approximationen av $u(t_i)$ för beräkningsnoden (tidpunkten) t_i.

Uppgift 4. Skriv ett program `min_ode` med anropet `[t,U]=min_ode(f,I,ua,h)` som löser begynnelsevärdesproblemet (4). Du skall använda programskalet: `min_ode.m` (se studiohemsidan).

Uppgift 5. Testa programmet på följande begynnelsevärdesproblem. För varje exempel måste du skriva en funktionsfil eller ett funktionshandtag som beskriver differentialekvationens högerled. Lös först begynnelsevärdesproblemet analytiskt (dvs. som en formel med penna och papper). Plotta både den analytiska lösningen u och den approximativa lösningen U i samma figur.

(a). $\begin{cases} u'(t) = t^2,\ t \in [1, 3], \\ u(1) = 1. \end{cases}$ **(b).** $\begin{cases} u'(t) = u(t),\ t \in [0, 2], \\ u(0) = 1. \end{cases}$

(c). $\begin{cases} u'(t) = -t\,u(t),\ t \in [0, 3], \\ u(0) = 1. \end{cases}$ **(d).** $\begin{cases} u'(t) = -5u(t),\ t \in [0, 1], \\ u(0) = 2. \end{cases}$

4 Allmänt system av ODE

Vi kommer vilja lösa m differentialekvationer med tillhörande begynnelsevärden

$$\begin{cases} u_1'(t) = f_1(t, u_1(t), \cdots, u_m(t)), & t \in [a, b],\ u_1(a) = u_{a1}, \\ \qquad\vdots & \qquad\qquad\vdots \\ u_m'(t) = f_m(t, u_1(t), \cdots, u_m(t)), & t \in [a, b],\ u_m(a) = u_{am}. \end{cases}$$

Om vi inför vektorerna

$$\boldsymbol{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_m(t) \end{bmatrix}, \quad \boldsymbol{f}(t, \boldsymbol{u}) = \begin{bmatrix} f_1(t, u_1(t), \cdots, u_m(t)) \\ \vdots \\ f_m(t, u_1(t), \cdots, u_m(t)) \end{bmatrix}, \quad \boldsymbol{u}_a = \begin{bmatrix} u_{a1} \\ \vdots \\ u_{am} \end{bmatrix}$$

så kan begynnelsevärdesproblemet skrivas på standardform

$$\begin{cases} \boldsymbol{u}'(t) = \boldsymbol{f}(t, \boldsymbol{u}(t)), \quad t \in [a, b], \\ \boldsymbol{u}(a) = \boldsymbol{u}_a. \end{cases} \tag{5}$$

Här är $\boldsymbol{f} : \mathbb{R}^{m+1} \to \mathbb{R}^m$ den funktion som beskriver differentialekvationen och vektorn $\boldsymbol{u}_a$ ger begynnelsevillkoret.

Vi har tittat på Eulers framåtmetod för skalär ODE och den fungerar lika bra för system av ekvationer. Man behöver bara modifiera programmet lite. Mer om detta och om de andra metoderna i en senare studio-övning. Nu skall vi se på färdiga program i MATLAB.

5 ODE-lösare i MATLAB

MATLAB har flera program som löser ODE med samma anrop som `min_ode` utom att man inte behöver ange steget; det väljs adaptivt av programmet. Till exempel,

```
>> [t,U]=ode45(f,I,ua)
>> [t,U]=ode15s(f,I,ua)
```

Här är `ode15s` avsedd för s.k. styva problem som ibland uppstår vid kemiska reaktioner. Vi återkommer till detta i en senare studio-övning.

Som exempel på hur man löser en skalär ODE med `ode45` tar vi: Beräkna och rita lösningen till differentialekvationen

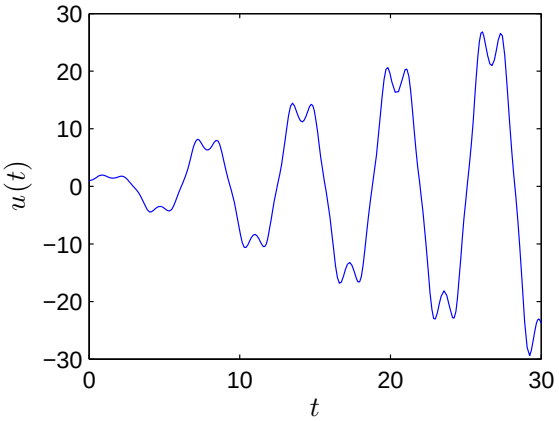
$$\begin{cases} u'(t) = t\cos(t) + \sin(4t)\, u(t), \, t \in [0, 30], \\ u(0) = 1. \end{cases}$$

Vi beskriver högerledet i differentialekvationen med ett funktionshandtag

```
f=@(t,u)t*cos(t)+sin(4*t)*u;
```

Vi löser och ritar upp med

```
>> [t,u]=ode45(f,[0,30],1);
>> plot(t,u)
```



Lägg märke till hur vi anger intervallet för t och hur vi ger begynnelsevärdet $u(0) = 1$.

Uppgift 6. Lös begynnelsevärdesproblemet

$$\begin{cases} u'(t) = \cos(3t) - \sin(5t)\, u(t), \ \ t \in [0, 15], \\ u(0) = 2, \end{cases}$$

med `ode45` och rita upp lösningen.

Vi kan lika lätt lösa system av differentialekvationer

$$\begin{cases} \boldsymbol{u}'(t) = \boldsymbol{f}(t, \boldsymbol{u}(t)), \quad t \in [a, b], \\ \boldsymbol{u}(a) = \boldsymbol{u}_a. \end{cases} \tag{6}$$

där

$$\boldsymbol{u}(t) = \begin{bmatrix} u_1(t) \\ \vdots \\ u_m(t) \end{bmatrix}, \ \ \boldsymbol{f}(t, \boldsymbol{u}) = \begin{bmatrix} f_1(t, u_1(t), \cdots, u_m(t)) \\ \vdots \\ f_m(t, u_1(t), \cdots, u_m(t)) \end{bmatrix}, \ \ \boldsymbol{u}_a = \begin{bmatrix} u_{a1} \\ \vdots \\ u_{am} \end{bmatrix}$$

Skillnaden är att nu blir `U` en $n \times m$-matris och vi finner $u_k(t)$ (för olika tidpunkter t_i) som kolonn `U(:,k)`, dvs. kolonn nr k i `U`.

Som exempel på ett system av ekvationer tar vi: *Populationsdynamik* – Vi betraktar population av bytesdjur (kaniner) som lever tillsammans med en population rovdjur (rävar). Låt $u_1(t)$ respektive $u_2(t)$ beteckna antalet kaniner respektive rävar vid tiden t. En enkel matematisk modell för populationernas utveckling ges av Volterra-Lotka-ekvationerna:

$$\begin{cases} u_1'(t) = a\, u_1(t) - b\, u_1(t)\, u_2(t) \\ u_2'(t) = -c\, u_2(t) + d\, u_1(t)\, u_2(t) \end{cases} \tag{7}$$

Koefficienterna a, b, c, d är positiva. Termen $a\, u_1(t)$ representerar netto-födelse-dödstalet i en ensam kaninpopulation. Termen $-c\, u_2(t)$ är motsvarande för rävarna. Termen $-b\, u_1(t)\, u_2(t)$ är antalet kaniner som blir uppätna per tidsenhet. Termen $d\, u_1(t)\, u_2(t)$ är antalet rävar per tidsenhet som överlever på grund av tillgång på föda. Observera teckenkombinationen i ekvationerna. Vad blir lösningen om populationerna är ensamma ($b = d = 0$)?

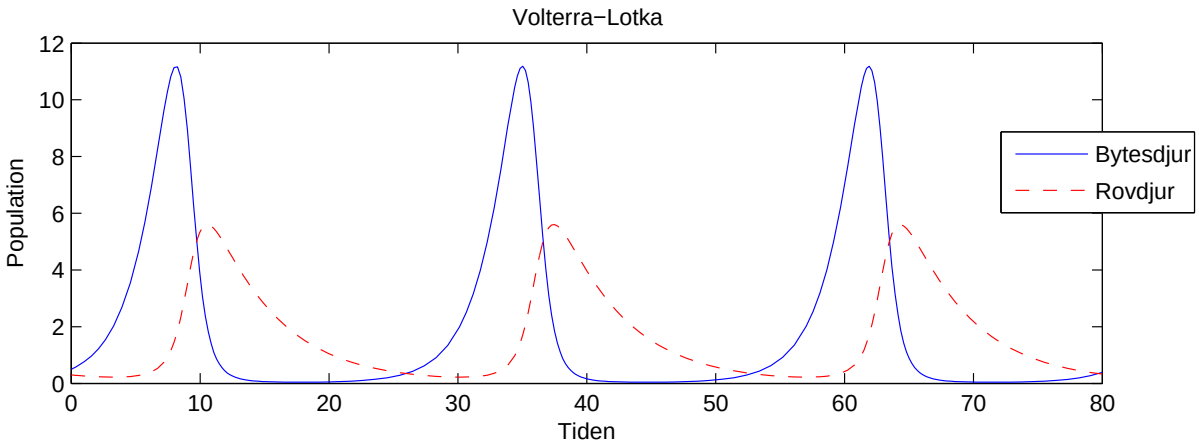
Vi beskriver högerledet i differentialekvationen med en funktion

```
function f=volterra(t,u)
a=0.5; b=0.3; c=0.2; d=0.1;
f=[ a*u(1)-b*u(1)*u(2)
   -c*u(2)+d*u(1)*u(2)];
```

Vi löser sedan med funktionen `ode45` och ritar upp enligt

```
>> ua=[0.5;0.3];
>> [t,U]=ode45(@volterra,[0,80],ua);

>> figure(1), clf
>> plot(t,U(:,1),t,U(:,2),'r--')
>> legend('Bytesdjur','Rovdjur')
>> xlabel('Tiden'), ylabel('Population')
>> title('Volterra-Lotka')
```



Uppgift 7. Lös Volterra-Lotka-ekvationerna med `ode45`. Ändra koefficienterna till $a = 0.5, b = 0.3, c = 0.2, d = 0.05$.

6 Redovisning

Denna vecka skall uppgifterna 1-7 redovisas för handledaren.

7 Inför nästa veckas studio-övning

Inför nästa veckas studio-övning är det viktigt att man i förväg läser igenom texterna för kinetik-projektet. Observera att kemister kommer till dator-salarna. Ni måste i förväg satt igång med projektet för att kunna utnyttja möjligheten att diskutera med kemisterna. Missa inte den chansen!