TMV136 Matematisk analys i en variabel E, 2010

MATLAB — NUMERISK LÖSNING AV ORDINÄRA DIFFERENTIALEKVATIONER

Beskrivning och mål.

Den här laborationen syftar till att ge en grundläggande förståelse för numerisk approximation av ordinära differentialekvationer samt kännedom om de viktigaste färdiga ODE-lösarna i Matlab. Efter genomförd laboration skall du

- känna till och kunna använda några av Matlabs inbyggda ODE-lösare.
- kunna förstå viktiga delar av hjälptexten i dessa.
- därifrån dra slutsatser om vilken funktion du lämpligen bör använda på den aktuella differentialekvationen.
- kunna beskriva hur enkla approximationsmetoder så som framåt-Euler, bakåt-Euler och mittpuktsmetoden fungerar.
- Kunna skriva om högre ordningens differentialekvationer som system av första ordningens och definiera dessa i funktionsfiler som Matlabs ODE-lösare kann hantera.
- kunna skriva program som använder sig av dessa för att approximera lösningen till en differentialekvation.
- förstå och gärna skriva `while`-loopar.
- kunna diskutera begreppet effektivitet hos ett program i termer av tid, minne och antal operationer med handledaren.

Arbeta gärna två och två och diskutera er fram till lösningar. Se till att båda har förstått vad ni gör. Spara alla filer på bådas användarkonton och kommentera programmen på egen hand efter laborationstillfället för att repetera. Det kan vara lämpligt att den som har minst programmeringsvana sitter vid tangentbordet, eller att ni turas om.

Deklarera alla matematiska funktioner som explicita Matlab-funktioner, dvs i funktions*filer*. Hela laborationen ska inte redovisas utan valda delar som vi meddelar i slutet av det fjärde laborationstillfället.

1. Uppgifter

1.1. **Primitiva funktioner.** Det enklaste tänkbara begynnelsevärdesproblemet

$$\begin{aligned} u'(t) &= f(t),\ t \in (0,T], \\ u(0) &= u_0 \end{aligned} \tag{1}$$

har en lösning som är en primitiv funktion $F(x)+C$ till $f(x)$ där C väljes så att begynnelsevärdet är uppfyllt. Vi kan approximera det genom att dela upp intervallet $[0,T]$ i delintervall med ändpunkter i $0 = t_0 < t_1 < \ldots t_{N-1} < t_N = T$ approximera derivatan i punkt t_i enligt formeln

$$\frac{u(t_n) - u(t_{n-1})}{t_n - t_{n-1}} \approx u'(t_{n-1}) = f(t_{n-1})$$

Här kan vi lösa ut $u(t_n)$ och får med $k = t_n - t_{n-1}$ att

$$u(t_n) = u(t_{n-1}) + kf(t_{n-1}). \tag{2}$$

Eftersom vi vet $u(t_0)$ kan vi räkna ut $u(t_1)$ och sedan $u(t_2)$ och så vidare. Metoden ovan, då man använder $f(t_{n-1})$ i högerledet kallas för "Eulers framåtmetod", "Euler framåt", "Eulers explicita metod" eller bara "Eulers metod".

Date: 2010–11–22, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

Uppgift 1. Skriv en funktionsfil med namnet `minPrim1.m` som tar emot ett funktionshandtag f, en vektor I med ett intervalls början och slut (två värden) samt en variabel X_0 med ett begynnelsevärde och en variabel N med antal delintervall. Det ska lösa differentialekvationen (1). Du ska alltså färdigställa följande program:

```
function [t,U]=minPrim(?,?,?,?)
%Vi börjar testa antalet inargument och tillåter användaren
%att inte bry sig om hur många delintervall det ska vara.
NrIn=nargin; %Returnerar antalet inargument som användaren angett.
if NrIn==?  %Gör ingenting
elseif NrIn=?
   N=100;
else
   error('Wrong number of input arguments!')
end
U=zeros(1,N+1); %Programmet blir snabbare om vi skapar en
                %vektor som har rätt storlek innan vi börjar
                %loopa.
k=? %steglängden.
t=? %Vektor med alla t_n
U(1)=U0;
for k=2:?
   U(?)=U(?)+k*f(?);
end
```

Använd programmet för att lösa differentialekvationerna nedan. Använd olika antal steg. Jämför med den analytiska lösningen.

$$(a)\quad \begin{aligned} &u'(x) = x^2, \quad x \in [0,2],\\ &u(0) = 3, \end{aligned}$$

$$(b)\quad \begin{aligned} &u'(x) = x^2, \quad x \in [1,4],\\ &u(1) = 3, \end{aligned}$$

$$(c)\quad \begin{aligned} &u'(x) = \cos(2x), \quad x \in [0,\pi],\\ &u(0) = 0, \end{aligned}$$

$$(d)\quad \begin{aligned} &u'(x) = 1/x, \quad x \in [1,10],\\ &u(1) = 0, \end{aligned}$$

Uppgift 2. Lös samma problem med Matlabs `ode45.m` som är standardlösaren i Matlab. För att det ska gå måste du skriva om funktionsfilerna som definierar f en liten, liten smula. Läs hjälptexten till `ode45`!

Uppgift 3. Om man istället gör approximationen

$$\frac{u(t_n) - u(t_{n-1})}{t_n - t_{n-1}} \approx u'(t_n) = f(t_n)$$

och löser ut $u(t_n)$ får man en annan approximationsmetod som har fått namnet ”Euler bakåt” eller ”Eulers implicita metod”. Härled motsvarigheten till (2) för denna metod och skriv en funktion `minPrim2.m` så att den använder denna metod istället.

1.2. **Mer generella differentialekvationer.** Betrakta differentialekvationen

$$\begin{aligned} u'(t) + u(t) &= g(t),\ t \in (0,T], \\ u(0) &= u_0. \end{aligned} \tag{3}$$

Den går inte att lösa med ditt program som det är. Men om vi använder Euler framåt så krävs bara en liten liten ändring för att det ska fungera. Vi skriver om första raden i (3) så att $u'(t) = g(t) - u(t) =: f(t, u(t))$. Nu blir Euler framåt på formen

$$u(t_n) = u(t_{n-1}) + kf(t_{n-1}, u(t_{n-1})). \tag{4}$$

Uppgift 4. Spara om ditt program som `myODE1.m` och ändra det så att det löser godtyckliga differentialekvationer på formen

$$\begin{aligned} u'(t) &= f(t, u(t)),\ t \in (0,T], \\ u(0) &= u_0 \end{aligned} \tag{5}$$

med Eulers framåtmetod (4).

Några ODE:er. Lös problemen (b) 0ch (d) i Uppgift 7 med både `ode45` och `myODE1`.

Uppgift 5. Eulers bakåtmetod för ekvationer av typen (5) lyder

$$u(t_n) = u(t_{n-1}) + kf(t_n, u(t_n)). \tag{6}$$

Vad är problemet med Eulers bakåtmetod när man ska lösa generella ekvationer på formen (5)? Detta problem åtgärdas med t.ex. fixpunktsiteration som i följande program `backwardEuler.m`. Vi måste alltså lösa ekvationen (6) för varje steg. En variant av fixpunktsiteration är Newtos metod som kan generaliseras till flera dimensioner.

```
function [t,U]=backwardEuler(f,I,U0,N)

t=linspace(I(1),I(2),N+1); %Genererar tidsnoder
U=zeros(length(U0),size(t,2)); %Vektor där lösningen sparas
U(:,1)=U0; %Första värdet är begynnelsevärdet
k=t(2)-t(1); %Steglängden
tol=k^2; %Fixpunktstolerans
for n=2:N+1
    U(:,n)=U(:,n-1);    %Vi sätter en första gissning på U(:,n) till
                        %till värdet innan.
    U_old=U(:,n)+2*tol; %U_old sätts så att while-loopen påbörjas.
    while norm(U(:,n)-U_old)>tol    %Om U(:,n) är längre ifrån vad det
                                    %var förra varvet så uppdaterar vi.
        U_old=U(:,n); %Vi spar värdet som U(:,n) har för att kunna jämföra.
        U(:,n)=U(:,n-1)+k*f(t(n),U(:,n)); %Fixpunktssteg. Ändra här för
                                          %att få mittpunktsmetoden!
     end
end
U=U'; %Vi gör om till radvektorer
t=t'; %för att göra lika som ode45
```

Lös uppgifterna (b) och (d) även med detta program.

1.3. **Högre ordningens ODE.** Betrakta differentialekvationen

$$\begin{aligned} u''(t) &= -c^2 u(t), \quad t \in [0,b], \quad (b > 0) \\ u(0) &= u_0,\ u'(0) = u_1, \end{aligned}$$

med lösningen

$$u(t) = u_0 \cos(ct) + \frac{u_1}{c} \sin(ct). \tag{7}$$

Verifiera detta genom att derivera och genom att sätta in $t = 0$!

Denna differentialekvation är av andra ordningen. För att kunna använda algoritmen ovan skriver vi om ekvationen som *ett system av två differentialkevationer av första ordningen*. Vi inför två nya variabler

$$w_1 = u,\ w_2 = u'.$$

Vi deriverar

$$\begin{aligned} w_1' &= u' = w_2, \\ w_2' &= u'' = -c^2 u = -c^2 w_1. \end{aligned}$$

Begynnelsevillkoren blir $w_1(0) = u_0$, $w_2(0) = u_1$. Vi ser att w_1 och w_2 löser begynnelsevärdesproblemet

$$\begin{cases} w_1' = w_2, \\ w_2' = -c^2 w_1, \end{cases} \qquad \begin{cases} w_1(0) = u_0, \\ w_2(0) = u_1. \end{cases}$$

Om Eulerlösarna modifieras så att de kan ta emot vektorer så kan man nu lösa detta system om man definierar en funktionsfil `ode2solve.m` enligt

```
function Uprim=ode2solve(t,U)
c=1;
Uprim(1,1)=U(2);
Uprim(1,2)=-c^2*U(1);
```

Man behandlar högre ordningens system analogt.

1.4. **Uppgift 6.** Plotta den numeriska lösningen tillsammans med den analytiska lösningen (om du känner den analytiska) för följande problem. I synnerhet (c) och (e).

$$(a) \quad \begin{cases} u'(x) = x^2, \quad x \in [1,3], \\ u(1) = 1. \end{cases}$$

$$(b) \quad \begin{cases} u'(t) = cu(t), \quad t \in [0,2], \\ u(0) = u_0. \end{cases}$$

$$(c) \quad \begin{cases} u'' = -c^2 u, \quad t \in [0,10] \\ u(0) = u_0,\ u'(0) = u_1. \end{cases}$$

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

analytisk lösning: $u(x) = \exp(-x^2/2)$

$$(e) \quad \begin{cases} u'' + 4u' + 13u = e^{-t}, \quad t \in [0,10] \\ u(0) = 1,\ u'(0) = 2. \end{cases}$$

□

1.5. **Uppgift 7.** Lös ekvation (c) ovan både med Eulers framåt- och bakåtmetod. Jämför med den analytiska lösningen. Skriv om programmet `backwardEuler.m` så att det istället använder *mittpunktsmetoden*

$$u(t_n) = u(t_{n-1}) + kf\left(\frac{t_n + t_{n-1}}{2}, \frac{u(t_n) + u(t_{n-1})}{2}\right). \tag{8}$$

Det behövs bara en liten förändring i fixpunktssteget. Testa alla programmen med $N = 10$, 100, 1000, 10000. Blir lösningen bättre med högre N? För någon av lösarna kommer det kanske inte att fungera för $N = 10$. Funktionen $g(x) := x + kf(t,x)$ uppfyller då inte villkoren i fixpunktssatsen, sid. 221 i Adams och while-loopen bara fortsätter, det konvergerar inte mot någon lösning.

1.6. **System av ODE.** Vi kan tänka oss system av obekanta där de olika obekanta inte är varandras derivator utan fysikaliska storheter. Det kan vara olika spänningar i elektriska kretsar eller koncentrationer av kemiska ämnen som reagerar med varandra. Ett exempel är Volterra-Lotka-ekvationerna nedan. Lägg lite kraft på att förstå vad de olika termerna i differentialekvationen betyder.

Uppgift 8. 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:

$$\begin{aligned} u_1'(t) &= au_1(t) - bu_1(t)u_2(t) \\ u_2'(t) &= -cu_2(t) + du_1(t)u_2(t) \end{aligned} \tag{9}$$

Koefficienterna a, b, c, d är positiva. Termen $au_1(t)$ representerar netto-födelse-dödstalet i en ensam kaninpopulation. Termen $-cu_2(t)$ är motsvarande för rävarna. Termen $-bu_1(t)u_2$ är antalet kaniner som blir uppätna per tidsenhet. Termen $du_1(t)u_2(t)$ är antalet rävar per tidsenhet som överlever på grund av tillgång på föda.

Observera teckenkombinationen i ekvationerna. Vad blir lösningen om populationerna är ensamma ($b = d = 0$)? Den modellen passar kanske bra för en rävpopulation utan tillgång till föda (åtminstone i början) men knappast för en ensam kaninpopulation. Man kan förvänta sig att brist på mat hindrar tillväxten efter ett tag. Hur tror du en modell som tar hänsyn till det skulle kunna se ut? Inom populationsekologin jobbar man ofta med den här typen av modeller och försöker skattakoefficienterna för att bedömma populationers känslighet för störningar etc.

Lös Volterra-Lotka-ekvationerna med din mittpunktslösare. Lämpliga värden är $a = .5$, $b = 1$, $c = .2$, $d = 1$. Testa med lite olika begynnelsevärden! Försök hitta några där populationen håller sig relativt stabil. Vad har en sådan situation för ekologisk tolkning? Vad innebär det att lösningen närmar sig noll?

Uppgift 9. Skriv `doc ode45` vid Matlabs kommandoprompt. Då öppnas dokumentationen inte bara för `ode45` utan för alla(?) Matlabs ode-lösare. En bit ned finns en lista som beskriver dessa och när de ska användas. Där framgår att `ode45` är den som ska användas oftast. Har man däremot ett *styvt* problem så rekommenderas `ode15s`. Ett styvt problem är ett problem där lösningen ibland förändras snabbt och ibland långsammare. Där den förändras snabbt så vill man ta små steg för att fånga denna förändring utan att få för stora fel. Där den inte förändras lika snabbt kan det dock vara bra att ta längre steg för att snabba upp beräkningarna. Matlabs `ode45` klarar inte av det senare. Det framgår med all önskvärd tydlighet om man gör Exempel 2 i nämnda dokumentation först med `ode15s` som föreslås och sedan med `ode45`. Gör det! Vad händer? vad der Pols ekvation kommer urspungligen från studier av elektriska kretsar om man får tro http://en.wikipedia.org/wiki/Van_der_Pol_oscillator.