Uppsala universitet
Institutionen för informationsteknologi
Beräkningsvetenskap

Lösningar till tentamen i *Numeriska metoder och simulering* 5.0 hp, 2016-10-17

Uppgifter som testar måluppfyllelse för betyg 3

1. (a) Skriv om differentialekvationen på den form som `ode45` förutsätter: $y'(t) = -2\sin(y(t)) + t^2$. Skriv sedan en matlabfunktion som beräknar ODE-problemets högerled:

```
function yprim = myODE( t, y)
   yprim = -2*sin(y)+t^2;
end
```

Exempel på ett skript som löser problemet med de indata som angavs i uppgiften:

```
[t, y] = ode45(@myODE, [0 10], 0.5);
plot(t,y)
```

   (b) Exempel på ett program enligt uppgiften:

```
N = input('Antal simuleringar av dygnsrytm?  ');
resultat = zeros(1,N);

for i = 1:N
      resultat(i) = dygnsrytm(200);
end

slutresultat = mean(resultat);
disp(['Genomsnittlig dygnslängd:  '...
      num2str(slutresultat) ' timmar' ])
```

2. (a) Underflow

   (b) Lokalt trunkeringsfel

(c) Implicit metod

(d) Stabilitet

3. (a) Skriv om differentialekvationen på den form som förutsätts i implicita Eulers metod:

$$y'(t) = -2\sin(y(t)) + t^2$$

Implicita Eulers metod tillämpad på denna ekvation blir:

$$y_{i+1} = y_i + h\left(-2\sin(y_{i+1}) + t_{i+1}^2\right)$$

(b) Inverse Transform Sampling innebär i detta fall att r, numret på nästa reaktion, ska väljas som det minsta tal x så att $F(x) \geq u$. Med givna värden kan $F(x)$ representeras av vektorn

$$\begin{aligned} F &= [0.1 \quad 0.1+0.4 \quad 0.1+0.4+0.2 \quad 0.1+0.4+0.2+0.3] \\ &= [0.1 \quad 0.5 \quad 0.7 \quad 1.0]. \end{aligned}$$

Vi ser att $F(3) = 0.7$ är det första värde i F som är större än $u = 0.62$. Slutsatsen blir att $r = 3$.

4. (a) Se föreläsningsanteckningar.

(b) När N ökar minskar den övre gränsen för felet proportionellt mot $1/\sqrt{N}$. Om antalet punkter fördubblas från 10000 till 20000 kommer alltså felet att ungefär minska med faktorn $1/\sqrt{2}$.

5. (a) Under de förutsättningar som anges i uppgiften är det glest mellan molekylerna. Molekylerna rör sig i cellen och reaktion kan bara ske när molekyler råkar träffa på varandra. Det blir alltså enstaka reaktioner i slumpmässig ordning och vid slumpmässiga tidpunkter. Med andra ord är slumpmässigheten betydelsefull under dessa förhållanden. Slutsatsen blir att det är lämpligast med en stokastisk modell i detta fall.

(b) Heuns metod och klassiska Runge-Kuttas metod är explicita metoder. Eftersom ODE-modellen inte är styv kommer de explicita metoderna inte att behöva ta extremt korta steg av stabilitetsskäl. Vi kan därför anta att det är toleransen som kommer att avgöra hur korta stegen behöver vara. Heuns metod har noggrannhetsordning 2, klassiska Runge-Kutta har noggrannhetsordning 4. Tack vare den högre noggrannhetsordningen kan klassiska Runge-Kutta

uppfylla toleransen med väsentligt färre beräkningspunkter än Heun. Detta uppväger med råge att klassiska Runge-Kutta kräver dubbelt så mycket beräkningsarbete som Heun per beräkningspunkt. Under antagandena i uppgiften kommer därför klassiska Runge-Kuttas metod att behöva kortare exekveringstid än Heuns metod för att genomföra simuleringen med önskad noggrannhet. Slutsatsen blir att klassiska Runge-Kuttas metod är lämpligare än Heuns metod i detta fall.

Uppgift som testar måluppfyllelse för betyg 4

6. Nedan visas ett program som genomför simulering enligt uppgiften.

   Högerledet i ODE-systemet beskrivs i funktionen `populationODE`:

```
function yprim = populationODE(t, y)
  global r1 r2 k1 k2 b12 b21
  yprim = [r1*y(1)*(k1-y(1)-b12*y(2))/k1;
           r2*y(2)*(k2-y(2)-b21*y(1))/k2];
```

   Följande skript genomför simuleringen:

```
% Sätt start- och sluttid för simuleringen
starttid = 0;
sluttid = 365;
tidsintervall = [starttid sluttid];

% Sätt begynnelsevärden
N1 = input('Begynnelsevärde på N1 ?  ');
N2 = input('Begynnelsevärde på N2 ?  ');
y0 = [N1 N2];

% Sätt värden på modellparametrarna
global r1 r2 k1 k2 b12 b21
r1 = input('Värde på r1?  ');
r2 = input('Värde på r2?  ');
k1 = input('Värde på k1?  ');
k2 = input('Värde på k2?  ');
b12 = input('Värde på b12?  ');
b21 = input('Värde på b21?  ');

% Lös ODEn genom anrop av ode45
```

```
[t, y] = ode45(@populationODE, tidsintervall, y0);

% Rita upp lösningen
plot(t,y);
xlabel('Tid');
ylabel('Populationstäthet');
title('Populationsutveckling');
legend('Art 1', 'Art 2');
```

Eftersom ODE-systemet inte är styvt kommer det att vara lämpligt med en explicit metod för att lösa ODE-problemet numeriskt. Om ODE-systemet hade varit styvt skulle explicita metoder ha behövt ta mycket korta tidssteg av stabilitetsskäl. En implicit metod hade då varit att föredra eftersom implicita metoder har bättre stabilitetsegenskaper. När problemet är icke-styvt kan man anta att metoder av samma noggrannhetsordning kan ta ungefär lika långa steg för att uppfylla toleransen. Explicita metoder är då att föredra framför implicita, eftersom de implicita metoderna kräver mera beräkningsarbete per beräkningssteg. Programmet ovan anropar Matlabs inbyggda ODE-lösare `ode45`. Den har hög noggrannhetsordning, använder en explicit metod och är därför lämplig när ODE-systemet inte är styvt.

Uppgift som testar måluppfyllelse för betyg 5

7. Vi börjar med att skriva om modellen som ett system av första ordningens ODE, eftersom de ODE-lösare vi har använt i Matlab förutsätter att systemet har denna form. För att åstadkomma omskrivningen inför vi nya variabler:

$$Y_1(t) = x(t), Y_2(t) = y(t), Y_3(t) = x'(t), Y_4(t) = y'(t).$$

Vi får då:

$$\begin{pmatrix} Y_1'(t) \\ Y_2'(t) \\ Y_3'(t) \\ Y_4'(t) \end{pmatrix} = \begin{pmatrix} x'(t) \\ y'(t) \\ x''(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} x'(t) \\ y'(t) \\ -Cx(t)/(\sqrt{x(t)^2 + y(t)^2})^3 \\ -Cy(t)/(\sqrt{x(t)^2 + y(t)^2})^3 \end{pmatrix} =$$

$$\begin{pmatrix} Y_3(t) \\ Y_4(t) \\ -CY_1(t)/(\sqrt{Y_1(t)^2 + Y_2(t)^2})^3 \\ -CY_2(t)/(\sqrt{Y_1(t)^2 + Y_2(t)^2})^3 \end{pmatrix}.$$

För att lösa detta system i Matlab behöver vi skriva en funktion som beräknar högerledet i ovanstående system:

```
function fvalue = satellitODE(t, y)
  global C
  expr = sqrt(y(1)^2+y(2)^2)^3;
  fvalue = [y(3);
            y(4);
            -C*y(1)/expr;
            -C*y(2)/expr];
```

Följande skript genomför simuleringen:

```
% Sätt värden på modellparametrarna
global C
C = input('Värde på C: ');

% Sätt begynnelsevärden
y0 = zeros(4,1);
y0(1) = input('Startkoordinat i x-led:  ');
y0(2) = input('Startkoordinat i y-led:  ');
y0(3) = input('Starthastighet i x-led:  ');
y0(4) = input('Starthastighet i y-led:  ');

% Sätt sluttid
tmax = input('Sluttid:  ');
tspan = [0 tmax];

% Lös ODEn genom anrop av ode45
[t, y] = ode45(@satellitODE,tspan,y0);

% Plotta planetens position och satellitbanan
plot(0,0,'r*')
hold on
plot(y(:,1),y(:,2))
hold off
```

För att praktiskt avgöra om ODE-problemet är styvt eller ej kan man köra programmet ovan i två olika versioner, en där ode45 används och en där ode15s används, och mäta exekveringstiden. Eftersom ode45 använder en explicit numerisk metod kommer den att av stabilitetsskäl behöva ta mycket korta steg om problemet är styvt. I så fall kommer

exekveringstiden för ode45 att bli väsentligt längre än för ode15s. Om däremot exekveringstiden för ode45 blir jämförbar med eller lägre än för ode15s, så har inte stabilitetsvillkoret för ode45 varit begränsande för steglängdsvalet och vi kan då dra slutsatsen att problemet inte är styvt.