Uppsala universitet
Institutionen för informationsteknologi
Beräkningsvetenskap

Lösningar till tentamen i *Numeriska metoder och simulering* 5.0 hp, 2018-10-25

Uppgifter som testar måluppfyllelse för betyg 3

1. (a) Skriv om differentialekvationen på den form som `ode45` förutsätter: $P'(t) = (k + 0.1\sin(t))P(t)(C - P(t))$. Skriv sedan en matlabfunktion som beräknar ODE-problemets högerled:

```
function Pprim = myODE( t, P)
   global k C
   Pprim = (k + 0.1*sin(t))*P*(C-P);
end
```

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

```
global k C
k = input("k? ");
C = input("C? ");
[t, P] = ode45(@myODE, [0 100], 10);
plot(t,P)
```

(b) Exempel på ett program enligt uppgiften:

```
a = input("Integrationsintervallets nedre gräns, a?  ");
b = input("Integrationsintervallets övre gräns, b?  ");
N = input('Antal punkter?  ');
resultat = zeros(1,N);
for i = 1:N
      x = a + (b-a)*rand;
      resultat(i) = f(x);
end
I = (b-a)*mean(resultat)
```

Ett kortare program som löser samma uppgift:

```
a = input("Integrationsintervallets nedre gräns, a?  ");
b = input("Integrationsintervallets övre gräns, b?  ");
N = input('Antal punkter?  ');
I = (b-a)*mean(f(a + (b-a)*rand(1,N)))
```

2. (a) Underflow
   (b) Monte Carlo-metod
   (c) Konvergens
   (d) Deterministisk metod

3. (a) Skriv om differentialekvationen på den form som förutsätts i Trapetsmetoden:

$$y'(t) = -\frac{2y(t)}{1 + 0.1y(t)}.$$

Trapetsmetoden uppställd för denna ekvation blir:

$$y_{k+1} = y_k + \frac{h}{2}\left(-\frac{2y_k}{1 + 0.1y_k} - \frac{2y_{k+1}}{1 + 0.1y_{k+1}}\right)$$

Vi har $y_0 = 1$ och med $h = 0.1$ kommer y_1 att vara ett närmevärde till $y(0.1)$. För att räkna ut y_1 flyttar vi över alla termer till vänsterledet och sätter in de kända värdena på y_0 och h:

$$y_1 - 1 - \frac{0.1}{2}\left(-\frac{2}{1 + 0.1} - \frac{2y_1}{1 + 0.1y_1}\right) = 0$$

Ovanstående är en ickelineär ekvation som vi kan lösa numeriskt (t ex med kommandot `fzero` i Matlab) för att få fram ett värde på y_1.

   (b) *Inverse Transform Sampling* innebär i det kontinuerliga fallet att x ska väljas som det värde som uppfyller $F(x) = u$, där u är ett slumptal mellan 0 och 1, med likformig fördelning. I vårt fall ska vi alltså lösa ekvationen:

$$\frac{x - a}{b - a} = u$$

och man får då formeln $x = a + u(b - a)$.

4. (a) Se föreläsningsanteckningar i foldern `Kursmaterial/Kurslitteratur-etc` i Studentportalen.

   (b) Den övre gränsen för felet är proportionell mot $1/\sqrt{N}$. För att halvera felet behöver vi därför ta fyra gånger så många punkter ($1/\sqrt{4N} = 1/(2\sqrt{N})$). Exekveringstiden är proportionell mot antalet punkter, så med fyra gånger så många punkter får vi ungefär fyra gånger så lång exekveringstid. Exekveringstiden kommer alltså att bli ca 8 sekunder.

5. (a) Båda metoderna är implicita och har därför goda stabilitetsegenskaper (de är båda ovillkorligt stabila). Det gör dem lämpliga för lösning av styva problem. Valet av steglängd kommer att avgöras av toleransen. Trapetsmetoden har högre noggrannhetsordning än Euler bakåt och kan därför uppfylla toleransen med betydligt större steglängd (och därmed färre steg) än Euler bakåt. Detta uppväger mer än väl att Trapetsmetoden kräver mer arbete per beräkningspunkt. Slutsatsen blir att Trapetsmetoden är lämpligare än Euler bakåt.

   (b) En stokastisk modell skulle vara lämpligare än en ODE-modell om reaktionerna mellan de tre kemikalierna skulle äga rum i en cell, där antalet molekyler av varje slag är litet. I en sådan situation kommer inte molekylerna i kontakt med varandra oavbrutet, utan det går en slumpmässig tid mellan två reaktioner. Vilken reaktion som kommer att äga rum vid en viss tidpunkt beror dessutom på vilka molekyler som råkar komma i kontakt med varandra då. I den beskrivna situationen går det alltså inte att bortse från slumpens betydelse. En deterministisk ODE-modell är därför olämplig i det fallet.

Uppgift som testar måluppfyllelse för betyg 4

6. Under antagandet att problemet *inte* är styvt skulle en explicit metod vara lämpligare än en implicit metod. Att problemet inte är styvt innebär att en explicit metod inte behöver begränsas av en onödigt kort steglängd av stabilitetsskäl. Då kommer en explicit metod, för att uppfylla toleransen, att kunna ta en steglängd som är jämförbar med den som behövs för en implicit metod av samma noggrannhetsordning. Det blir då ungefär lika många beräkningspunkter för båda typerna av metod. I det läget kommer den explicita metoden att lösa problemet med kortast exekveringstid, eftersom den implicita metoden

kräver väsentligt mera arbete per beräkningspunkt. Av nämnda skäl väljer vi att i det här fallet använda Matlabs `ode45`, som bygger på en explicit metod av hög noggrannhetsordning.

Nedanstående är exempel på ett program som använder `ode45` för att simulera hur encelliga bakterier konsumerar glukos.

Högerledet i ODE-systemet beskrivs som en funktion (där `km1` står för k_{-1}):

```
function yprime = BacteriaODE(t, y)
global r km1 k1 k2
c = y(1);
x = y(2);
yprime = [-k1*r*c + (km1 + k1*c)*x;
           k1*r*c - (km1 + k2 + k1*c)*x];
```

Följande huvudprogram utför simuleringen:

```
% Läs in begynnelsevärden och sluttid
c = input('Begynnelsevärde för c?  ');
x = input('Begynnelsevärde för x?  ');
T = input('Sluttid?  ');

% Läs in värden på modellparametrarna
global r km1 k1 k2
r = input('Värde på r?  ');
km1 = input('Värde på km1?  ');
k1 = input('Värde på k1?  ');
k2 = input('Värde på k2?  ');

% Genomför simuleringen
[t, y] = ode45(@BacteriaODE,[0 T], [c; x]);

% Plotta koncentrationer som funktion av tiden
plot(t, y)
title('Bakteries konsumtion av glukos');
xlabel('Tid');
ylabel('Koncentration');
legend('Koncentration av fritt glukos','Koncentration av receptorer
med glukos');
```

Uppgift som testar måluppfyllelse för betyg 5

7. I kursen har vi översiktligt gått igenom hur det automatiska steglängdsvalet i `ode23` och `ode45` fungerar och även visat hur det formelmässigt ser ut i `ode23` (se Kursmaterial/Kurslitteratur-etc/Automatiskt-adaptivt-steglängdsval.pdf). I Molers avsnitt 7.5 och 7.6 finns ytterligare detaljer om `ode23`. Idén är att man har ett par av Runge-Kutta metoder, av noggrannhetsordning q och $q+1$ där beräkningarna i metoden av högre noggrannhetsordning återanvänds i formlerna för metoden av lägre noggrannhetsordning. I `ode23` och `ode45` används det resultat som beräknas med metoden av högre noggrannhetsordning. Som feluppskattning används skillnaden mellan resultaten från de två metoderna. *Resultatet av metoden med lägre noggrannhetsordning beräknas inte explicit*, utan man tar på förhand, en gång för alla, fram formeln för skillnaden mellan de två metoderna och använder sedan den formeln i feluppskattningen. För våra två metoder, Heuns metod och metod (1), får vi:

$$\begin{aligned} y_{k+1}^{(\text{Heun})} - y_{k+1}^{(1)} &= h\left(\left(\frac{1}{2}-\frac{2}{3}\right)s_1 + \left(\frac{1}{2}-\frac{1}{6}\right)s_2 - \frac{1}{6}s_3\right) \\ &= h\left(-s_1+2s_2-s_3\right)/6. \end{aligned}$$

Notera också att `s1` för en tidpunkt är detsamma som `s3` för närmast föregående tidpunkt. Genom att utnyttja detta får man en mycket effektiv algoritm, där antalet räkneoperationer väsentligen är detsamma som om man enbart hade använt Heuns metod.

Mot denna bakgrund skulle `ode12` kunna implementeras på nedanstående sätt. Denna funktion går att exekvera och ger rimliga resultat, men den ska ses som en skiss av hur denna typ av metoder i princip är konstruerade. I de ”riktiga” funktionerna `ode23` och `ode45` är många detaljer mera genomarbetade och ”sofistikerade”. En sådan detalj är valet av inledande steglängd. I min lösningsskiss har jag använt att metoden har noggrannhetsordning 2. Med den mycket grova uppskattningen att $fel(h_i) \approx h_i^2$ får man då att `tol^(1/2)` är en lämplig steglängd att börja med. En annan sådan detalj är valet av ny steglängd. Vi har beräknat uppskattningen att $fel(h_i) \approx$ `err` och söker nu nästa h-värde, $h_{i+1} = ch_i$, så att $fel(h_{i+1}) \approx$ `tol`. Om vi kombinerar dessa formler med samma mycket grova uppskattning som nämndes ovan får vi att $c \approx$ `(tol/err)^(1/2)`, vilket är vad jag har använt i nedanstående skiss. Notera att detta medför att om `err` $>$ `tol` kommer steget att minska och om `err` $<$ `tol` kommer steget att öka.

```
function [tout, yout] = ode12(odefun,tspan,y0,tol)

% Initiering av variabler
t = tspan(1);
tfinal = tspan(2);
y = y0;

tout = [t];
yout = [y'];

% Sätt inledande steglängd och beräkna första s1-värdet
h = tol^(1/2);
s1 = odefun(t,y);

% Stega framåt i tiden
while t < tfinal

   if h > tfinal-t
     h = tfinal-t; % Se till att sista t-värde blir tfinal
   end

   s2 = odefun(t+h,y+h*s1);
   ynew = y + (h/2)*(s1 + s2); % Detta är Heun-värdet
   tnew = t + h;

   s3 = odefun(tnew,ynew);

   err = norm(h*(-s1 + 2*s2 - s3)/6); % Uppskattning av felet

   if err <= tol % Spara resultatet om toleransen uppfylldes
     t = tnew;
     y = ynew;
     tout(end+1,1) = tnew;
     yout(end+1,:)  = ynew';
     s1 = s3; % Utnyttja att nya s1 är senaste s3
   end
   h = h*(tol/err)^(1/2); % Beräkna nästa steglängd

end
```