Uppsala universitet
Institutionen för informationsteknologi
Beräkningsvetenskap

Lösningar till tentamen i *Numeriska metoder och simulering* 5.0 hp, 2019-10-25

Uppgifter som testar måluppfyllelse för betyg 3

1. (a) Skriv om differentialekvationen på den form som `ode45` förutsätter: $y'(t) = -\sin(t+y(t))^3/\sqrt{t+1}$ Skriv sedan en matlabfunktion som beräknar ODE-problemets högerled:

```
function yprim = myODE( t, y)
      yprim = -sin(t + y)^3/sqrt(t + 1);
end
```

   Exempel på ett kort program som löser problemet:

```
T = input("Sluttid?  ");
a = input("Begynnelsevärde?  ");
[t, y] = ode45(@myODE, [0 T], a);
```

   (b) Exempel på ett program enligt uppgiften:

```
N = input('Antal upprepade simuleringar?  ');
resultat = zeros(1,N);
for i = 1:N
      resultat(i) = dygnslangd(0.16);
end
medeldygnslangd = mean(resultat)
```

2. (a) Kancellation
   (b) Adaptivt steglängdsval
   (c) Konsistens
   (d) Stokastisk modell

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

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

Explicita Eulers metod uppställd för denna ekvation blir:

$$y_{k+1} = y_k - h\frac{2y_k}{1+0.1y_k}.$$

Vi har $y_0 = 10$ och med $h = 0.1$ kommer y_1 att vara ett närmevärde till $y(0.1)$. För att räkna ut y_1 sätter vi $k = 0$ i ovanstående formel. Med de kända värdena på y_0 och h får vi:

$$y_1 = 10 - 0.1\frac{20}{1+1} = 9.$$

(b) *Inverse Transform Sampling* innebär i den diskreta versionen att r, numret på nästa utfall, ska väljas som det minsta tal x så att $F(x) \geq u$, där $F(x)$ är den kumulativa fördelningsfunktionen. Med de givna värdena kan $F(x)$ representeras av vektorn

$$F = \begin{bmatrix} 0.45 \\ 0.45 + 0.10 \\ 0.45 + 0.10 + 0.45 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0.55 \\ 1.0 \end{bmatrix}.$$

Vi ser att $F(3) = 1.0$ är det första värde i F som är större än $u = 0.61$. Slutsatsen blir att $r = 3$. Det blir alltså färgen grön som slumpas fram.

4. (a) Lokala trunkeringsfelet τ för implicita Eulers metod är:

$$\tau = y(t_{i+1}) - y(t_i) - hf(t_{i+1}, y(t_{i+1})) \tag{1}$$

Notera att vår ODE är $y'(t) = f(t, y(t))$. Observera även att $t_{i+1} = t_i + h$. Dessa observationer insatta i (1) ger:

$$\tau = y(t_i + h) - y(t_i) - hy'(t_i + h) \tag{2}$$

Taylorutveckling kring t_i ger:

$$\begin{aligned} y(t_i + h) &= y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \mathcal{O}(h^3) \\ y'(t_i + h) &= y'(t_i) + hy''(t_i) + \mathcal{O}(h^2) \end{aligned}$$

Insättning i (2) ger:

$$\begin{aligned} \tau = \; & y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + \mathcal{O}(h^3) \\ & -y(t_i) \\ & -h\left(y'(t_i) + hy''(t_i) + \mathcal{O}(h^2)\right) \end{aligned}$$

Gruppering av termer efter h-potens ger:

$$\begin{aligned}\tau &= y(t_i) - y(t_i) + \\ &\quad h\left(y'(t_i) - y'(t_i)\right) + \\ &\quad h^2\left(\frac{1}{2}y''(t_i) - y''(t_i)\right) + \\ &\quad \mathcal{O}(h^3)\end{aligned}$$

Lokala trunkeringsfelet för implicita Eulers metod är alltså $-\frac{h^2}{2}y'''(t_i) + \mathcal{O}(h^3) = \mathcal{O}(h^2)$. Det globala felet är då $\mathcal{O}(h)$. Slutsats: noggrannhetsordningen är 1.

(b) Exekveringstiden är direkt proportionell mot antalet upprepade simuleringar. Fördubbling av antal simuleringar medför alltså ungefär fördubblad exekveringstid. Slutsats: exekveringstiden blir ca 2 minuter.

5. (a) Heuns metod är explicit och Trapetsmetoden är implicit. Eftersom ODE-modellen inte är styv kommer den explicita metoden 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. Båda metoderna har noggrannhetsordning 2, så om man ser till trunkeringsfelet behöver båda metoderna använda ungefär samma antal beräkningspunkter för att uppfylla noggrannhetskravet. Men Trapetsmetoden, som är implicit, kommer att utföra mera arbete per beräkningspunkt än Heuns metod, som är explicit. Under antagandena ovan kommer därför Heuns metod att behöva kortare exekveringstid än Trapetsmetoden för att genomföra simuleringen med önskad noggrannhet. Slutsatsen blir att Heuns metod är lämpligare än Trapetsmetoden i detta fall.

(b) Om man tänker på de modeller som vi har använt i kursen, så är exekveringstiden för att simulera kemiska reaktioner kortare om man använder en deterministisk ODE-modell än om man gör stokastisk simulering. Om det slumpmässiga inslaget är försumbart är därför en deterministisk modell för de kemiska reaktionerna lämpligast. Under de förutsättningar som anges i uppgiften kan man anta att det vid varje tidpunkt sker många reaktioner mellan de olika kemikalierna (och inte enstaka reaktioner vid slumpmässiga tidpunkter). Därför kommer slumpmässigheten att vara försumbar i det scenario som uppgiften avser. Slutsatsen blir att det är lämpligast med en deterministisk modell i detta fall.

Uppgifter som testar måluppfyllelse för högre betyg

6. *Program för betyg 4:* Nedanstående är ett exempel på hur `heun` skulle kunna utformas. Så som det fungerar i Matlabs ODE-lösare (`ode45` etc) så förutsätts `odefun` returnera en kolonnvektor om det är ett system av ODE som ska lösas. Vidare förutsätts att inparametern `y` till `odefun` är en kolonnvektor, så nedanstående förutsätter att även `y0` är en kolonnvektor. Utparametern `yout` ska vara en matris, med lika många rader som det finns tidpunkter i vektorn `t`. Rad nummer `i` i `yout` ska innehålla den numeriska lösningen för tidpunkten `t(i)`. Eftersom den numeriska lösningen är en kolonnvektor måste den transponeras innan den läggs in i `yout`.

```
function [t, yout] = heun(odefun, tspan, y0, h)
   % Initialisering
     t = tspan(1):h:tspan(2);
     m = length(t);
     n = length(y0);

     yout = zeros(m,n);

     yout(1,:)  = y0';

   % Begynnelsevärde
     y = y0;

   % Tidsstegning
     for i = 1:m-1
          ti = t(i);
          yi = y;

          K1 = odefun(ti,yi);
          K2 = odefun(ti+h,yi+h*K1);

          y = yi + 0.5*h*(K1 + K2);
          yout(i+1,:)  = y';
     end
end
```

Testprogrammet kan se ut så här:

```
[t, y] = heun(@(t, y) -y, [0 1], 1, 0.1);

% Plotta den numeriska lösningen som diskreta punkter
plot(t, y, '*')

% Plotta den exakta lösningen y(t) = exp(-t) som heldragen
% kurva i samma plot
hold on
tplot = linspace(0, 1);
yexact = exp(-tplot);
plot(tplot, yexact);

xlabel('t')
ylabel('y')
legend('Numerisk lösning','Exakt lösning')
```

Analys för betyg 5: Eftersom ODE-systemet inte är styvt är det rimligt att anta att den steglängd som krävs för att felet ska uppfylla toleransen även uppfyller stabilitetsvillkoret. Vi har nu för en steglängd h_1 fått att fel$(h_1) \approx 10^{-4}$ och söker en kortare steglängd $h_2 = \alpha h_1$ så att fel$(h_2) \approx 10^{-6}$. Eftersom Heuns metod har noggrannhetsordning 2 är fel$(h) \approx ch^2$. Därför gäller följande:

$$10^{-6} \approx c \cdot (\alpha h_1)^2 = \alpha^2 c h_1^2 \approx \alpha^2 10^{-4}.$$

Vi får att α ska väljas så att $\alpha^2 = 10^{-2}$, det vill säga att $\alpha = 0.1$. Exekveringstiden är proportionell mot antalet tidssteg och därmed omvänt proportionell mot steglängden. Vi har att:

$$\frac{1}{h_2} = \frac{1}{0.1 h_1} = 10 \frac{1}{h_1}.$$

När vi minskar steglängden från h_1 till h_2 så ökar alltså antalet tidssteg med en faktor 10 och därmed ökar även exekveringstiden med denna faktor. Exekveringstiden för fel i storleksordningen 10^{-4} blev 20 sekunder. Exekveringstiden för fel i storleksordningen 10^{-6} blir alltså, under de antaganden som vi har gjort, ca $10 \cdot 20 = 200$ sekunder.

7. *Analys för betyg 4*: Ensemblesimuleringen är en Monte Carlo-metod. För en sådan är övre gränsen för felet med N upprepningar proportionell mot $1/\sqrt{N}$. Vi ökar antalet upprepningar till kN så att felgränsen halveras. Då ska alltså k väljas så att:

$$\frac{1}{\sqrt{kN}} = \frac{1}{2} \frac{1}{\sqrt{N}}.$$

Detta innebär att $k = 4$. Exekveringstiden med 500 upprepningar blev 2 minuter. Eftersom exekveringstiden för en Monte Carlo-metod är proportionell mot antalet upprepningar, så kommer exekveringstiden med $4 \cdot 500$ upprepningar att bli ca $4 \cdot 2 = 8$ minuter. (En mera detaljerad motivering av att exekveringstiden är proportionell mot antalet upprepade simuleringar: I `ensemble` är övriga operationers exekveringstid försumbar jämfört med exekveringstiden för en enstaka simulering. Exekveringstiden för `ensemble` blir alltså ca Nt_s, där t_s är tiden per simulering.)

Program för betyg 5: Nedanstående är ett exempel på hur `ensemble` skulle kunna utformas. Eftersom funktionen skulle vara lämplig för problem som inte är styva använder vi `ode45`. I kursens laboration 2, om hur det går till att lösa ODE i Matlab, ingick att använda `odeset` för att föreskriva den tolerans `ode45` ska använda för det relativa felet. Det kommer till användning i `ensemble`.

```
function ymean = ensemble(odefun, tspan, y0, tol,...
                          nsim, eps)
   % Initialisering
     ymean = 0;
     choice = odeset('RelTol', tol);
     twoeps = 2*eps;

   % Upprepade simuleringar
     for i = 1:nsim
          epsi = -eps + rand()*twoeps;

          [t, y] = ode45(odefun, tspan, y0 + epsi,...
                         choice);

          ymean = ymean + y(end);
     end

     ymean = ymean/nsim;
end
```