Björn Lindenberg 2004-02-11

me02_bli@it.kth.se

Laboration 3 del 1

Ickelinjära ekvationssystem

Uppgift 1: Van der Waals tillståndlag

$$\left(p + \frac{a}{v^2} \right)(v - b) = RT$$

beskriver relationen mellan trycket p, temperaturen T och volymen v i reala gaser, och där a och b är experimentellt bestämda konstanter. R är den allmänna gaskonstanten. För givna värden på T=300 och på p=1,10 och 100 skulle motsvarande värden på v bestämmas. Resultaten skulle jämföras med värden genom den ideala gaslagen

$$pv = RT \ .$$

Utförande: Newtons metod

$$v_{k+1} = v_k - z_k, \text{ med } z_k = \frac{f(v_k)}{f'(v_k)}$$

användes som den itererande algoritmen för att finna ett nollställe till

$$f(v_k) = \left(p + \frac{a}{v^2} \right)(v - b) - RT \ .$$

Som begynnelsevärde för iterationen användes

$$v_0 = \frac{RT}{p},$$

vilket kan anses ligga nära det korrekta värdet och en maxtolerans på 1e-6 på normen av z_k. I MATLAB användes symbolisk hantering av funktionerna för att underlätta deriveringen och hanterandet av algoritmen.

Resultat: Beräknade värden på v och v_0 för olika värden p kan ses i tabellen nedan.

Volym\ Tryck	p=1	p=10	p=100
v	24.5126	2.3545	0.0795
v_0	24.6162	2.4616	0.2462

Den relativa skillnaden mellan van der Waals lag och den ideala gaslagen blir påtaglig vid högt tryck och liten volym.

Koden:

Upp1.m

```
syms v
R=0.082054;
a=3.592; k=0;
b=0.04267;
T=300;
for p=[1,10,100]
f=(p+a/v^2)*(v-b)-R*T;
fprim=diff(f,'v');
v0=R*T/p;
vk=v0;
zk=subs(fprim,v,vk')\subs(f,v,vk');
while norm(zk,Inf)>1e-6
    vk=vk-zk;
    zk=subs(f,v,vk')/subs(fprim,v,vk');
end
vk
v0
end
```

Uppgift 2:

Ett ickelinjärt ekvationssystem av fyra okända t_1, t_2, w_1, w_2 gavs av

$$\begin{aligned} w_1 + w_2 &= 2, \\ w_1 t_1 + w_2 t_2 &= 0, \\ w_1 t_1^2 + w_2 t_2^2 &= 2/3, \\ w_1 t_1^3 + w_2 t_2^3 &= 0. \end{aligned}$$

Lösningar skulle bestämmas där $t_i \in [-1, \quad 1]$.

Utförande:

Newtons metod utökat för system av ekvationer användes med

$$\mathbf{f}(t_1, t_2, w_1, w_2) = \begin{pmatrix} w_1 + w_2 - 2 \\ w_1 t_1 + w_2 t_2 \\ w_1 t_1^2 + w_2 t_2^2 - 2/3 \\ w_1 t_1^3 + w_2 t_2^3 \end{pmatrix},$$

så att $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ sökes. Den itererande algoritmen skrivs som $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{z}_k$, där $\mathbf{z}_k$ bestäms genom matrisekvationen $\mathbf{J}_\mathbf{f}(\mathbf{x}_k)\mathbf{z}_k = \mathbf{f}(\mathbf{x}_k)$. Som startvärde användes $\mathbf{x}_0 = (-0.5, 0.5, 1, 1)$ och en maxtolerans 1e-12 på normen av $\mathbf{z}_k$. En MATLAB funktion Upp2 med inbyggda subfunktioner f(xk) och f_jakobian(xk) returnerade den approximerade lösningen.

Resultat:

Som lösning erhölls värden på -0.5774 och 0.5774 för t_i och 1 för vikterna w_1 och w_2. Symmetrin med avseende på t_1 och t_2 gjorde att olika tecken kunde erhållas för *t*-variablerna beroende på startvektorn.

Koden:

Upp2.m

```
function xk=Upp2
xk=[3,5,9,7]';
Fk=feval(@f_jakobian,xk);
fk=feval(@f,xk);
zk=Fk\fk;
k=0;
while norm(fk)>1e-12
    xk=xk-zk;
    k=k+1;
    Fk=feval(@f_jakobian,xk);
    fk=feval(@f,xk);
    zk=Fk\fk;
end

function fk=f(xk)
fk=zeros(4,1);
fk(1)=xk(3)+xk(4)-2;
fk(2)=xk(3)*xk(1)+xk(4)*xk(2);
fk(3)=xk(3)*xk(1)^2+xk(4)*xk(2)^2-2/3;
fk(4)=xk(3)*xk(1)^3+xk(4)*xk(2)^3;

function fprim_xk=f_jakobian(xk);
fprim_xk=[ 0, 0, 1, 1;
        xk(3), xk(4), xk(1), xk(2);
        2*xk(3)*xk(1), 2*xk(4)*xk(2), xk(1)^2, xk(2)^2;
        3*xk(3)*xk(1)^2, 3*xk(4)*xk(2)^2, xk(1)^3, xk(2)^3];
```

Bratuproblemet

Randvärdesproblemet

$$-u'' = \lambda e^u, \quad 0 < t < 1, \quad \lambda > 0,$$
$$\mathrm{u}(0) = \mathrm{u}(1) = 0,$$

har för $\lambda = 1$ exakt två lösningar.

Uppgift 3:

En finita differensmetod med *n* ekvidistanta noder skulle tillämpas på problemet för *n*=1,3,7 och 15.

Utförande:

Diskretiseringen över nätet gav upphov till ickelinjärt system

$$\mathbf{f}(\mathbf{u}) = \mathbf{A}\mathbf{u} + \mathbf{h}(\mathbf{u}) = \mathbf{0},$$

där

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}, \quad \mathbf{A} = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}, \quad \mathbf{h}(\mathbf{u}) = -h^2 \begin{pmatrix} e^{u_1} \\ e^{u_2} \\ \vdots \\ e^{u_n} \end{pmatrix}.$$

Lösningen beräknades med en maxtolerans på 1e-8 för normen på $\mathbf{z}_k$ genom Newtons metod:

$$\mathbf{u}_{k+1} = \mathbf{u}_k - \mathbf{z}_k, \quad \mathbf{J}_\mathbf{f}(\mathbf{u}_k)\mathbf{z}_k = \mathbf{f}(\mathbf{u}_k),$$

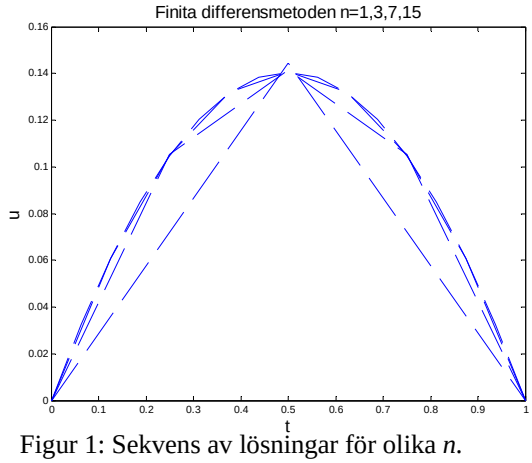
med

$$\mathbf{J}_\mathbf{f}(\mathbf{u}_k) = \mathbf{A} + \mathbf{h}'(\mathbf{u}_k), \ \mathbf{h}'(\mathbf{u}_k) = diag(\mathbf{h}(\mathbf{u}_k)).$$

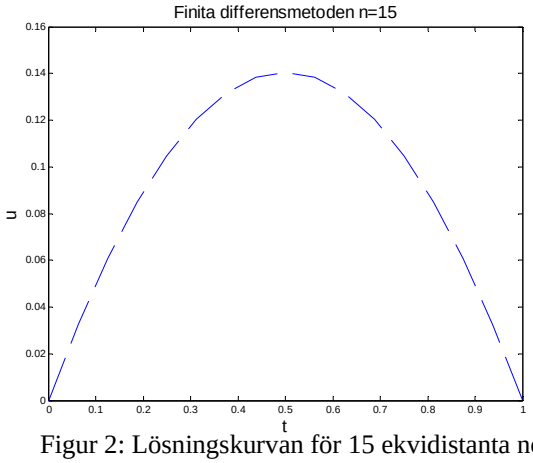
Startvärdet för iterationen sattes till $\mathbf{u}_0 = \mathbf{0}$ och sekvensen av lösningar för olika antal noder plottades.

Resultat:

Lösningskurvorna med linjärinterpolation och en graf av den diskreta lösningen för n=15 kan ses i figur 1 och 2.



Figur 1: Sekvens av lösningar för olika *n*.



Figur 2: Lösningskurvan för 15 ekvidistanta noder.

Koden:

Upp3.m

```
for n=[1,3,7,15,100]
k=0;
h=1/(n+1);
u=zeros(n,1);
s=ones(n,1);
A=spdiags([-s,2*s,-s],-1:1,n,n);
H=-h^2*exp(u);
F=A+spdiags(H,0,n,n);
f=A*u+H;
zk=F\f;
while norm(zk,Inf)>1e-8
    u=u-zk;
    k=k+1;
    H=-h^2*exp(u);
    F=A+spdiags(H,0,n,n);
    f=A*u+H;
    zk=F\f;
end
u=[0;u;0];
t=0:h:1; figure(1);
plot(t,u,'--'); hold on;
```

```
end
hold off;
title('Finita differensmetoden n=1,3,7,15','FontSize',16);
xlabel('t','FontSize',16);
ylabel('u','FontSize',16);
figure(2); plot(t,u,'--')
title('Finita differensmetoden n=15','FontSize',16);
xlabel('t','FontSize',16);
ylabel('u','FontSize',16);
```

Uppgift 4a:

En metod kallad ”Collocation method” skulle användas för Bratuproblemet med n=3,4,5,6 noder (ekvidistanta) vid $t_i = ih,\ h = \frac{1}{(n+1)}$ och basfunktioner

$$v_n(t) = t(1-t)t^{n-1},$$

så att

$$u(t) \approx v(t) = \sum_n x_n t(1-t)t^{n-1} = t(1-t)(x_1 + x_2 t + \cdots + x_n t^{n-1}) .$$

Utförande:

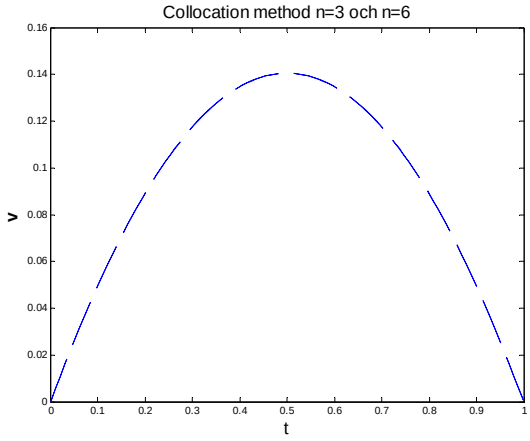
Insättning av *v(t)* i ekvationen gav för de olika noderna *n* stycken ickelinjära ekvationer, som löstes med Newton metoden genom:

$$f_i(\mathbf{x}) = v''(t_i) + e^{v(t_i)} = 0 \Leftrightarrow \mathbf{f}(\mathbf{x}) = \mathbf{0},$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{z}_k,\ \ \mathbf{J}_\mathbf{f}(\mathbf{x}_k)\mathbf{z}_k = \mathbf{f}(\mathbf{x}_k).$$

Symbolisk hantering av de okända underlättade vid beräkningen av Jakobianen och $\mathbf{x}_0$ gavs genomgående ettor för alla *n*.

Resultat:

Approximationen förbättrades inte nämnvärt av ytterligare tillägg av basfunktioner utöver n=3 vilket antyder att basfunktionernas karaktäristik för lägre *n* är väl anpassade till ena lösningens natur. En jämförelse mellan n=3 och n=6 kan ses i figur 3, tillsynes ligger de nära varandra.



Figur 3: Koefficienter för basfunktioner av lägre grad dominerar

Uppgift 4b:

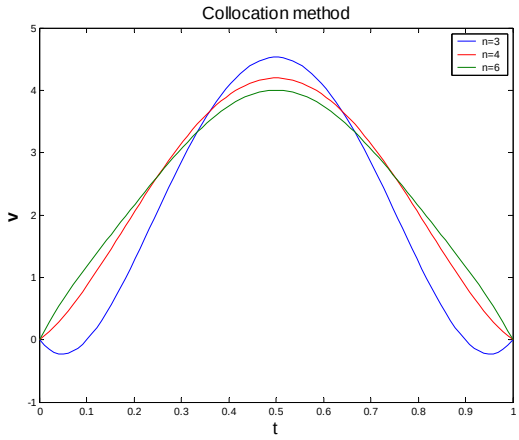
Bratuproblemet har för $\lambda = 1$ exakt två lösningar, den andra mer svårtillgängliga lösningen skulle finnas med ledtråden om att båda lösningarna har en inledande positiv derivata för små *t*.

Utförande:

Vid små *t* domineras derivatan till ansatsen av x_1 och x_2 och där första ordningen av *t* till lika stor del påverkas av de båda koefficienterna med motsatt tecken. Startvärdet för iterationen ansattes därför till $x_1 = 10$, $x_2 = 10$, vilket skulle göra derivatan positiv för små *t*.

Resultat:

Den andra lösningen hittades med ovanstående iterationsstart, förvisso med andra koefficient-värden. Lösningen kan ses i figur 4 för olika *n*. Lösningen antar mycket större värden än föregående kurva.



Figur 4: En ansats av fler basfunktioner visar att metoden konvergerar mot en annan lösning, en lösning som antar mycket större värden under intervallet.

Koden:

Upp4.m

```
syms x1 x2 x3 x4 x5 x6 t
X=[x1,x2,x3,x4,x5,x6];
dt=linspace(0,1,100);
vtot=[];
for n=[3,4,6];
h=1/(n+1);
v=0;
for i=1:n
    v=v+X(i)*t^(i-1);
end
v=v*t*(1-t);
s=v;
vbis=diff(v,'t',2);
ev=exp(v);
ti=h:h:1-h;
f=subs(vbis+ev,t,ti');
x=[];
for i=1:n
    x=[x,X(i)];
end
F=jacobian(f,x);
xk=zeros(n,1);
xk(1)=10; xk(2)=10;
zk=1e30*xk;
while norm(zk,Inf)>1e-9
zk=subs(F,x,xk')\subs(f,x,xk');
xk=xk-zk;
end
v=subs(v,x,xk');
v=subs(v,t,dt');
vtot=[vtot,v];
end
plot(dt,vtot(:,1),dt,vtot(:,2),'r',dt,vtot(:,3));
title('Collocation method','FontSize',16)
legend('n=3','n=4','n=6');
xlabel('t','FontSize',16);
ylabel('v','FontSize',16,'FontWeight','bold');
```

Naturkonstanten Pi

Det reella talet $\pi = 3.141592653...$ kan genom arctangens uttryckas med en integral

$$\int_0^1 \frac{4}{1+x^2} dx = \pi \, .$$

Integralens värde ska approximeras med numerisk integration.

Uppgift 5a:

Med hjälp av metoder baserade på polynominterpolation får man för olika antal interpolationsnoder formler för approximativa värden på integralen under intervallet. För numerisk integration används främst tre formler motsvarande antalet noder *n*=1,2,3:

$$R(f) = (b-a)\,f\left(\frac{a+b}{2}\right)$$ ger mittpunktsformeln,

$$T(f) = \frac{b-a}{2}\left(f(a)+f(b)\right)$$ ger trapetsapproximationen, och

$$S(f) = \frac{b-a}{6}\left(f(a)+f(b)+4f\left(\frac{a+b}{2}\right)\right)$$ ger Simpsons formel,

där *f* är integranden. Dess metoder ska användas för att approximera värdet på pi genom ovanstående integral. Metodernas precision ska jämföras och felet *e(h)* som en funktion av steglängden *h* ska uppskattas.

Utförande: Steglängden varierades genom halveringar 1/2, 1/4, 1/8, 1/16. Symbolisk hantering användes. Felet uppskattades vara en asymptotisk funktion $e(h) \approx ch^p$ med $p \approx \frac{\log|e(h)/e(h/q)|}{\log q}$ för två beräknade värden för *h* och *h/q*.

Resultat: I tabellen nedan ses felet för respektive metod och steglängd.

Metod\ *h*	*h=1/2*	*h=1/4*	*h=1/8*	*h=1/16*
Mittpunkts.	0.0208	0.0052	0.0013	3.2552e-004
Trapets.	0.0416	0.0104	0.0026	6.5104e-004
Simpsons	2.4026e-005	1.5113e-007	2.3650e-009	3.6957e-011

Simpsons formel visar en remarkabel precision även för stora *h*. Trapetsapproximationen, *n=2,* uppvisar däremot sämre egenskaper än mittpunktsformeln, *n*=1, för den givna integranden. Insättning av värden ifrån tabellen visar att felfunktionen för mittpunktsformeln och trapetsapproximation kan skrivas som $e(h) \approx 0.0832h^2$ och motsvarande uttryck för Simpsons formel blir $e(h) \approx 0.00062h^6$. Reduktion av steglängden tappar sin effektivitet när maskinprecisionen kommer in som en bidragande del av beräkningarna.

Uppgift 5b:

MATLABs inbyggda rutiner för numerisk integration skulle användas och tillförlitligheten i ansatt tolerans skulle utredas.

Resultat: Funktionerna quad (Simpson) och quadl (Lobatto) användes med olika toleranser. De angivna rutinerna uppfyllde varje toleranskrav.

Koden:

Upp5.m

```
syms x
h=1/16;
f=4/(1+x^2);
dx=0:h:1;
```

```
% R(f)
I=0;
for i=1:length(dx)-1
I=I+(dx(i+1)-dx(i))*subs(f,x,(dx(i+1)+dx(i))/2);
end
R=abs(I-pi)
% T(f)
I=0;
for i=1:length(dx)-1
I=I+(dx(i+1)-dx(i))/2*(subs(f,x,dx(i+1))+subs(f,x,dx(i)));
end
T=abs(I-pi)
% S(f)
I=0;
for i=1:length(dx)-1
I=I+(dx(i+1)-
dx(i))/6*(subs(f,x,dx(i+1))+subs(f,x,dx(i))+4*subs(f,x,(dx(i+1)+dx(i))/2));
end
S=abs(I-pi)
f=inline('4./(1+x.^2)');
Quad_Simpson=quad(f,0,1,1e-12)-pi
Quad_Lobatto=quadl(f,0,1,1e-12)-pi
```

Uppgift 6a: Föregående steg i uppgift 5a skulle utföras för integralen

$$\int_0^1 \sqrt{x}\log(x)dx = -\frac{4}{9}.$$

Utförande: För behandling av integrandens värde för x=0 beräknades gränsvärdet analytiskt

$$\lim_{x\to 0}\sqrt{x}\log(x) = \lim_{x\to 0}\frac{\log(x)}{x^{-1/2}} = \left[\frac{"-\infty"}{"\infty"} \Rightarrow l'Hospital\right]$$
$$= \lim_{x\to 0}\frac{1/x}{-1/2x^{-3/2}} = \lim_{x\to 0}(-2)x^{1/2} = 0.$$

Gränsvärdet substituerade sedan integrandens numeriska beräkning vid den kritiska punkten.

Resultat: I tabellen nedan ses felet för respektive metod och steglängd.

Metod\ *h*	*h=1/2*	*h=1/4*	*h=1/8*	*h=1/16*
Mittpunkts.	0.0267	0.0136	0.0064	0.0029
Trapets.	0.1994	0.0863	0.0364	0.0150
Simpsons	0.0487	0.0197	0.0078	0.0031

Ingen av metoderna uppvisar snabb konvergens mot det exakta värdet och mittpunktsformeln genererar bäst värden för de givna steglängderna. En uppskattning av felet beroende av h genom tabellvärden blir för mittpunktsformeln $e_M \approx 0.063h^{1.1}$ det vill säga felet är hyfsat linjärt för ansatta steglängder. Motsvarande uppskattning för trapetsapproximationen och Simpsons formel kan skrivas $e_T \approx 0.46h^{1.2}$ respektive $e_S \approx 0.12h^{1.3}$.

Uppgift 6b: MATLABs inbyggda rutiner för numerisk integration skulle användas. Tillförlitligheten i ansatt tolerans skulle utredas.

Resultat: Funktionerna quad (Simpson) och quadl (Lobatto) användes med olika toleranser. Den skickade toleransen till quad användes inte skarpt, den angivna toleransens storleksordning användes dock korrekt, t ex gav en tolerans på 1e-7 ett fel på 5.3761e-7. Lobatto funktionen quadl klarade dock varje toleranskrav skarpt.

Koden:

Upp6.m

```
syms x
f=sqrt(x)*log(x);
n=2;
h=1/n;
dx=0:h:1;
% R(f)
I=0;
for i=1:n
I=I+(dx(i+1)-dx(i))*subs(f,x,(dx(i+1)+dx(i))/2);
end
R=abs(I+4/9)
% T(f)
I=(dx(2)-dx(1))/2*(subs(f,x,dx(2)));
for i=2:n
I=I+(dx(i+1)-dx(i))/2*(subs(f,x,dx(i+1))+subs(f,x,dx(i)));
end
T=abs(I+4/9)
% S(f)
I=(dx(2)-dx(1))/6*(subs(f,x,dx(2))+4*subs(f,x,(dx(2)+dx(1))/2));
for i=2:n
I=I+(dx(i+1)-
dx(i))/6*(subs(f,x,dx(i+1))+subs(f,x,dx(i))+4*subs(f,x,(dx(i+1)+dx(i))/2));
end
S=abs(I+4/9)
Quad_Simpson=quad(@fx,0,1,1e-7)+4/9
Quad_Lobatto=quadl(@fx,0,1,1e-15)+4/9
```

fx.m

```
function val=fx(x)
ind = find(x>1e-100);
val = zeros(size(x));
val(ind) = sqrt(x(ind)).*log(x(ind));
```