

TENTAMEN, DEL 2
SF1544
 NUMERISKA METODER, GRUNDKURS IV
 Torsdag 16 mars 2023 kl 14.00–17.00

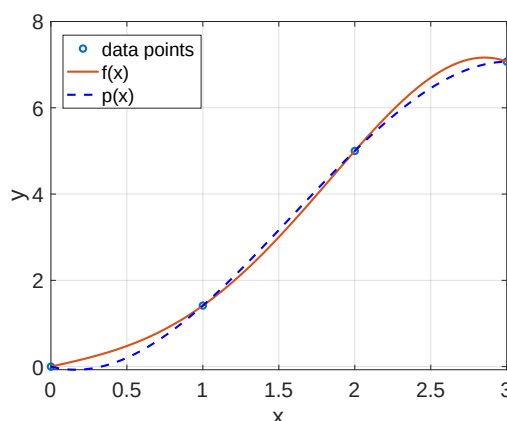
Rättas endast om Del 1 är godkänd. Maximalt antal poäng på denna del är 50.

Betygsgränser: 10 p D, 20 p C, 30 p B, 40 p A.

Svar skall motiveras och uträkningar redovisas detaljerat. Korrekt svar utan motivering eller med felaktig motivering medför poängavdrag.

Inga hjälpmedel är tillåtna (ej heller miniräknare).

- P1.** Tredjegradspolynomet $p(x)$ interpolerar funktionen $f(x) = \sin(\frac{\pi}{4}x)(1 + x^2)$ i punkterna $x = 0, 1, 2, 3$ (se figur).



Figur 1: Grafen till $\sin(\frac{\pi}{4}x)(1 + x^2)$ och $p(x)$.

- (4 p) **a)** Skriv en algoritm som bestämmer polynomet $p(x)$, utan att använda inbyggda fit-funktioner (som t.ex. `polyfit`).
- (7 p) **b)** Skriv ett program som räknar ut skillnaden i båglängd för kurvorna $f(x)$ och $p(x)$ på intervallet $0 \leq x \leq 3$. Skillnaden i båglängd ska beräknas med ett fel mindre än en given feltolerans. Alla integraler måste beräknas med hjälp av en egenskriven trapetsregel. (Du får alltså inte använda inbyggda funktioner som t.ex. `trapz` för att beräkna integraler.)

Tips: Båglängden L för en kurva $y(x)$ i intervallet $a \leq x \leq b$ ges av integralen

$$L = \int_a^b \sqrt{1 + y'(x)^2} dx.$$

- (6 p) c) Beskriv en detaljerad algoritm som beräknar b^* så att båglängden för kurvan $f(x)$ på intervallet $0 \leq x \leq b^*$ blir 1. Algoritmen måste vara effektiv. Programkod är inte nödvändig för full poäng.

Lösningsförslag:

- a) Förslag på Matlab-kod, här med den naiva ansatsen:

```
x = [0 1 2 3]';
y = sin(pi*x/4).*(1+x.^2);
V = [x.^0, x.^1, x.^2, x.^3]; % Vandermonde-matrisen
c = V\y;
p = @(x) c(1) + c(2)*x + c(3)*x.^2 + c(4)*x.^3;
```

- b) Fortsättning på koden från uppgift a):

```
h = 1; felenskatt = 1; iter = 1;
% antar given feltolerans "tol" finns
while felenskatt >= tol
    x = 0:h:3;
    % Beräkna båglängd för f(x)
    fp = (pi/4)*cos(pi*x/4).*(1+x.^2) + sin(pi*x/4).*(2*x);
    sf = sqrt(1 + fp.^2);
    Lf = h*(sf(1)/2 + sf(end)/2 + sum(sf(2:end-1)));
    % Beräkna båglängd för p(x)
    pp = c(2) + 2*c(3)*x + 3*c(4)*x.^2;
    sp = sqrt(1 + pp.^2);
    Lp = h*(sp(1)/2 + sp(end)/2 + sum(sp(2:end-1)));
    % Beräkna skillnaden
    skillnad = abs(Lf - Lp);
    % Uppskatta felet
    if iter > 1
        felenskatt = abs(skillnad - old);
    end
    old = skillnad;
    iter = iter + 1;
    h = h/2; % halvera steglängden
end
```

Lösningar utan while-slingor kan också ge full poäng om det till exempel beskrivs i ord hur felet uppskattas genom att halvera steglängden tills ändringen blir mindre än toleransen.

c) Vi kan till exempel använda sekantmetoden. Inför

$$L(b) = \int_0^b \sqrt{1 + f'(x)^2} dx$$

och beräkna integralen med trapetsregeln som i uppgift b). Låt $Q(b) = L(b) - 1$. Vi vill hitta ett nollställe till $Q(b)$. För att göra detta med sekantmetoden kan vi till exempel låta $b_0 = 1$, $b_1 = 1.1$ och

$$b_{k+1} = b_k - Q(b_k) \frac{b_k - b_{k-1}}{Q(b_k) - Q(b_{k-1})}, \quad k = 1, 2, \dots$$

Dessa iterat b_k konvergerar mot det sökta värdet b^* .

(I detta fall är även Newtons metod ett enkelt alternativ eftersom $Q'(b) = L'(b) = \sqrt{1 + f'(b)^2}$ enligt analysens huvudsats.)

P2. Vi vill lösa ekvationssystemet

$$\begin{aligned} ye^x + \ln(x) &= 2 \\ x^2y + ye^x &= -1. \end{aligned}$$

- (7 p) a) Inför lämpliga beteckningar och formulera Newtons metod för systemet i detalj. Alla ingående storheter ska skrivas ut explicit.
- (6 p) b) Skriv ett program som använder Newtons metod för att bestämma lösningen med ett fel mindre än 10^{-10} i både x och y . Använd $x = 1$, $y = 1$ som startgissning.
- (2 p) c) Formulera en fixpunktsiteration, som inte är Newtons metod, för att lösa ekvationssystemet. Du behöver inte kontrollera konvergensen i detta exempel.

Lösningsförslag:

a) Flytta över allt till ena sidan så fås ekvationen

$$\vec{F}(x, y) = \begin{bmatrix} ye^x + \ln(x) - 2 \\ x^2y + ye^x + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

dvs vi vill hitta ett nollställe till funktionen $\vec{F}(x, y)$. Vi behöver Jacobimatrisen

$$D\vec{F}(x, y) = \begin{bmatrix} ye^x + \frac{1}{x} & e^x \\ 2xy + ye^x & x^2 + e^x \end{bmatrix}$$

och kan sedan skriva upp Newtons metod

$$\vec{u}^{n+1} = \vec{u}^n - [D\vec{F}(\vec{u}_1^n, \vec{u}_2^n)]^{-1} \vec{F}(\vec{u}_1^n, \vec{u}_2^n), \quad n = 0, 1, 2, \dots$$

där $\vec{u}^0$ ges av en startgissning.

b) Förslag på Matlab-kod:

```

u = [1 1]';
s = 1;
while norm(s) >= 10^-10
    x = u(1); y = u(2);
    F = [y*exp(x)+log(x)-2;
          x^2*y+y*exp(x)+1];
    DF = [y*exp(x)+1/x, exp(x);
           2*x*y+y*exp(x), x^2+exp(x)];
    s = -DF\F;
    u = u + s;
end

```

c) För att skriva ekvationssystemet på fixpunktsform kan vi till exempel addera x respektive y till ekvationernas båda led:

$$\begin{aligned}
 ye^x + \ln(x) - 2 + x &= x \\
 x^2y + ye^x + 1 + y &= y.
 \end{aligned}$$

Vi kan nu använda fixpunktsiterationen

$$\begin{aligned}
 x_{n+1} &= y_n e^{x_n} + \ln(x_n) - 2 + x_n, \\
 y_{n+1} &= x_n^2 y_n + y_n e^{x_n} + 1 + y_n,
 \end{aligned}$$

$$n = 0, 1, 2, \dots$$

(15 p)

Alternativ uppgift P2. Formulera och bevisa en sats om stabilitet av en explicit differensmetod för värmeledningsekvationen.

Lösningsförslag: Se Sauer Theorem 8.2 (sida 380) eller anteckningarna från Föreläsning 9, 2023-02-13 (Stabilitetsanalys för framåtdifferens).

Lös antingen P2 eller alternativa uppgift P2, inte båda. Om lösningar till den alternativa uppgiften lämnas in är det de som kommer bedömas.

- P3.** Temperaturen i ett cylindriskt föremål ges av lösningen till den partiella differentialekvationen

$$\begin{aligned}\frac{\partial^2 T}{\partial r^2} + \frac{1}{r} \frac{\partial T}{\partial r} &= \frac{1}{4K} \frac{\partial T}{\partial t}, & \frac{1}{2} < r < 1, \quad 0 < t < 10, \\ T(\tfrac{1}{2}, t) &= t, \quad T(1, t) = 100 + 40t, & 0 \leq t \leq 10, & \quad (\clubsuit) \\ T(r, 0) &= 200(r - 0.5), & \frac{1}{2} \leq r \leq 1,\end{aligned}$$

där $T = T(r, t)$ är temperaturen, r det radiala avståndet från cylinderns centerlinje, t är tiden och K en given materialspecifik parameter. Vi antar att $K = 0.6$.

- (4 p) a) Antag i ett första steg att man förenklar problemet genom att sätta $\frac{\partial T}{\partial t} = 0$ och löser randvärdesproblemet

$$\begin{aligned}\frac{d^2 T}{dr^2} + \frac{1}{r} \frac{dT}{dr} &= 0, & \frac{1}{2} < r < 1, \\ T(\tfrac{1}{2}) &= 0, \quad T(1) = 100.\end{aligned} \quad (\diamond)$$

Ställ upp det linjära ekvationssystem (i matrisform) som behöver lösas just i detta fall då man använder finita differensmetoden med centraldifferens för att approximera $\frac{d^2 T}{dr^2}$ och $\frac{dT}{dr}$.

- (5 p) b) Skriv ett program som löser randvärdesproblemet i uppgift a) med steglängden $h = \Delta r = 0.1$. Programmet ska även plotta grafen till lösningen $T(r)$ i intervallet $\frac{1}{2} \leq r \leq 1$.
- (3 p) c) Vi vill nu använda metoden framåt Euler (framåt differens) för att lösa den ursprungliga partiella differentialekvationen ($\clubsuit$). Skriv upp det numeriska uttrycket för T_i^{j+1} , där $T_i^j \approx T(i\Delta r, j\Delta t)$. Använd som tidigare centraldifferens för att approximera $\frac{\partial^2 T}{\partial r^2}$ och $\frac{\partial T}{\partial r}$. (I detta steg kan du bortse från randvillkor och begynnelsevillkor.)
- (6 p) d) Skriv ett program som använder ditt numeriska uttryck från uppgift c) för att hitta en approximation till $T(r, 10)$ med steglängderna $h = \Delta r = 0.1$ och $k = \Delta t = 0.001$. Glöm inte rand- och begynnelsevillkor.

Lösningsförlag:

- a) Inför diskretiseringen $r_i = \frac{1}{2} + ih$ där $i = 0, 1, \dots, N$ och $h = \frac{1}{2}/N$. Låt $T_i \approx T(r_i)$. Insättning av centraldifferenserna

$$\frac{dT}{dr}(r_i) \approx \frac{T_{i+1} - T_{i-1}}{2h}, \quad \frac{d^2 T}{dr^2}(r_i) \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}$$

i differentialekvationen leder till

$$\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} + \frac{1}{r_i} \frac{T_{i+1} - T_{i-1}}{2h} = 0, \quad i = 1, \dots, N-1.$$

Samla ihop termer så fås

$$\left(\frac{1}{h^2} - \frac{1}{2hr_i}\right) T_{i-1} + \left(-\frac{2}{h^2}\right) T_i + \left(\frac{1}{h^2} + \frac{1}{2hr_i}\right) T_{i+1} = 0, \quad i = 1, \dots, N-1.$$

Vi har även randvärdena $T_0 = T(\frac{1}{2}) = 0$ och $T_N = T(1) = 100$.

Detta leder till det linjära ekvationssystemet

$$\begin{bmatrix} \alpha & \beta_1 & 0 & & & & 0 \\ \gamma_2 & \alpha & \beta_2 & 0 & & & \\ 0 & \gamma_3 & \alpha & \beta_3 & 0 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 0 & \gamma_{N-3} & \alpha & \beta_{N-3} & 0 \\ & & & 0 & \gamma_{N-2} & \alpha & \beta_{N-2} \\ 0 & & & & 0 & \gamma_{N-1} & \alpha \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ -\left(\frac{1}{h^2} + \frac{1}{2hr_{N-1}}\right) \cdot 100 \end{bmatrix}$$

där

$$\alpha = -\frac{2}{h^2}, \quad \beta_i = \frac{1}{h^2} + \frac{1}{2hr_i}, \quad \gamma_i = \frac{1}{h^2} - \frac{1}{2hr_i}$$

b) Förslag på Matlab-kod:

```
h = 0.1; N = 0.5/h; r = 0.5:h:1; rin = r(2:end-1);
% Konstruera matris
A = zeros(N-1, N-1);
for j=1:N-1
    A(j,j) = -2/h^2;
    if j>1
        A(j,j-1) = 1/h^2 - 1/(2*h*rin(j));
    end
    if j<N-1
        A(j,j+1) = 1/h^2 + 1/(2*h*rin(j));
    end
end
% Konstruera högerledet
b = zeros(N-1, 1);
b(end) = -(1/h^2 + 1/(2*h*rin(end)))*100;
% Lös och plotta
T = A\b;
T = [0; T; 100];
plot(r, T)
```

c) Insättning av derivata-approximationerna i (♣) ger

$$\frac{1}{4K} \frac{T_i^{j+1} - T_i^j}{\Delta t} = \frac{T_{i+1}^j - 2T_i^j + T_{i-1}^j}{(\Delta r)^2} + \frac{1}{r_i} \frac{T_{i+1}^j - T_{i-1}^j}{2\Delta r}.$$

Detta räcker som svar, men vi kan även lösa ut T_i^{j+1} :

$$T_i^{j+1} = T_i^j + \frac{4K\Delta t}{\Delta r} \left(\frac{T_{i+1}^j - 2T_i^j + T_{i-1}^j}{\Delta r} + \frac{T_{i+1}^j - T_{i-1}^j}{2r_i} \right).$$

d) Förslag på Matlab-kod:

```
h = 0.1; k = 0.001;
N = 0.5/h; M = 10/k;
r = 0.5:h:1;
T = 200*(r-0.5); % Begynnelsevillkor
K = 0.6;
% Stega med framåt Euler (framåt differens)
for j=1:M
    % Uppdatera enligt formeln ovan
    Tnew = T;
    for i=2:length(T)-2 % inre punkter
        C1 = (T(i+1) - 2*T(i) + T(i-1)) / h;
        C2 = (T(i+1) - T(i-1)) / (2*r(i));
        Tnew(i) = Tnew(i) + 4*K*k/h * (C1+C2);
    end
    % Glöm inte randvillkoren!
    t = k*j;
    Tnew(1) = t;
    Tnew(end) = 100 + 40*t;
    % Uppdatera variabeln
    T = Tnew;
end
% lösningen = T
```

(Det går förstås också bra att definiera en matris som man multiplicerar lösningen med i varje Euler-steg.)