Laborationer MMG410

Gustav Lindwall

May 23, 2019

Laboration 1

1.1 IEEE 754

IEEE 754 är den standard för flyttalsberäkning som används i de flesta datorer. Som bekant kan en dator generellt inte representera något annat än heltal exakt. Hur operationerna $+$, $\times$, / och $\sqrt{\cdot}$ skall gå till för flyttal (vanligen av precisionen `double`) är specificerat i IEEE 754. I den här laborationen skall vi belysa några av de vanligaste felen som kan uppstå när man är oförsiktig i sitt hanterande av flyttalsaritmetik.

1.1.1 Hanterandet av singulära uttryck i Matlab

Kör följande kodsnutt i Matlab:

```
a  = 1.0/0.0;
b  = 1.0/a;
sa = sin(a);
sb = sin(b);
la = log(a);
lb = log(b);
c  = 0.0/0.0;
d  = 1.0/c;
sd = sin(d);
```

Vid en första anblick är det enkelt att tro att samtliga av dessa uttryck skall returnera `NaN` (Not a Number), då rent matematiskt utgör de allihopa icke-definierade uttryck. Så är dock inte fallet. Vilka får man en väldefinierade output av? Vad tror ni anledningen till det är?

1.1.2 Minsta möjliga precision vid flyttalsräkning

Kör följande kodsnutt i Matlab:

```
format long e
logspace(-301, -324, 24)'
```

Som output skall ni få en vektor $[10^{-301}, 10^{-302}, \ldots, 10^{-324}]^T$. Är det verkligen vad ni får? Vad är orsaken till att outputen skiljer sig från vad det borde vara?

1.1.3 Fullständing utskiftning

Antag att vi har två flyttal, x och δ, så att $x \gg \delta$. Vi säger att *fullständig utskiftning* sker när $x + \delta \to x$, det vill säga att enligt IEEE 754-standarden ger addition av talen x och δ enbart x som output. Detta är såklart matematiskt omöjligt för alla $\delta \neq 0$, men det kan ske i en dator som en konsekvens av flyttalsaritmetikens begränsningar.

Sätt $x = 1$ och hitta sedan ett ungefärligt värde på δ så att $x + \delta$ resulterar i fullständig utskiftning. Gör sedan samma sak för $x = 10^{20}$ samt $x = 10^{-20}$. Hittar ni ett mönster i när utskiftning sker?

Ledning: Följande kod-stubb kan vara till hjälp:

```
x     = 1;
delta = x;
diff  = abs((x+delta)-x);
while diff > 0
    % Uppdatera delta och berakna diff igen
end
disp(delta)
```

1.1.4 Konsekvenser av utskiftning

I den här uppgiften skall vi studera vad som kan hända om man inte tar utskiftning i beaktning. Beräkna först summan $S_1 = \sum_{n=1}^{10^6} \frac{1}{n}$, förslagsvis med hjälp av en `for`-loop. Beräkna sedan summan $S_2 = \sum_{n=10^6}^{1} \frac{1}{n}$, det vill säga samma summa fast genomlöpt baklänges.

För att tydligare belysa utskiftningen skall summorna beräknas med `single`-precision till skillnad från den vanliga `double`-precisionen. Initialisera därför summorna som `S1 = single(0)` och `S2 = single(0)`. De två sätten att beräkna summan på kommer resultera i olika resultat. Varför det? Och vilken av summorna är närmst det verkliga värdet?

1.1.5 Konsekvenser av utskiftning, fortsättning

Kör följande kodsnutt i Matlab:

```
x  = linspace(0.995,1.005,1001);
y1 = (x-1).^6;
y2 = 1-6*x+15*x.^2-20*x.^3+15*x.^4-6*x.^5+x.^6;
plot(x,y1)
hold on
plot(x,y2)
```

Notera att uttrycken $(x-1)^6$ och $1 - 6x + 15x^2 - 20x^3 + 15x^4 - 6x^5 + x^6$ är identiska rent matematiskt. Vad är det då som ger de synliga skillnaderna i plotten? Vilket sätt är det bättre att uttrycka funktionen på?

1.2 Matrisfaktoriseringar

I beräkningsmatematik stöter man ofta på mycket stora matrisproblem. Vid hantering av stora matriser stöter man på två huvudsakliga problem; det ena är att datorn får flyttalsaritmetiska bekymmer likt de i den föregående uppgiften. Det andra är att antalet beräkningar växer sig så stora att det blir orimligt för datorn att kunna lösa ens problem inom en rimlig tid. Som tur är så finns det en uppsjö av tekniker man kan använda för att förenkla beräkningsarbetet. Vi skall illustrera ett par av dessa i den här uppgiften.

1.2.1 Lösning av linjära ekvationssystem i Matlab

Kör följande kodsnutt i Matlab:

```
n = 3000;
A = randn(n);
b = randn(n, 1);
tic; x = inv(A) * b; toc
tic; y = A \ b; toc
```

Koden genererar ett slumpmässigt, stort ekvationssystem som vi sedan löser dels med hjälp av `inv(A)*b`, och dels med Matlab-rutinen \. Vilken av dem går snabbast? Varför gör den det? Det kan hjälpa att kolla dokumentationen[1] för \.

1.2.2 Glesa matrisproblem

Vid numerisk lösning av differentialekvationer är det mycket vanligt att man får ut stora matriser där en överväldigande majoritet av elementen är lika med noll. Dessa matriser kallas för *glesa matriser*. Som ni strax kommer att se är det alldeles speciellt ineffektivt att arbeta med inverser av glesa matriser. Man brukar därför när det är möjligt utnyttja någon känd struktur hos matrisen till sin fördel.

Kör följande kodsnutt i Matlab:

```
n = 6000;
e = ones(n, 1);
A = spdiags([e 2*e e], -1:1, n, n);
b = randn(n, 1);
tic; x = inv(A) * b; toc
tic; y = A \ b; toc
```

Kommandot `spy(A)` markerar nollskilda element i matrisen A med en blå prick. A är konstruerad så att elementen på diagonalen, samt de två diagonalerna direkt över och under, är de enda nollskilda. En sådan matris kallas för *tridiagonal* och är mycket vanligt förekommande.

Jämför återigen tidsskillnaden i att använda `inv(A)` kontra \-rutinen för att lösa linjära ekvationssystem. Den tridiagonala matrisen A är även positivt definit. Vilken matrisfaktorisering kommer \ att använda sig av? Avslutningsvis kan ni köra `spy(inv(A))`. Är inversen av A gles bara för att A är gles?

[1] Här får ni en länk: `https://se.mathworks.com/help/matlab/ref/mldivide.html`.

Laboration 2

2.1 Minstakvadratproblem

2.1.1 Greve Rumfords experiment

I den här uppgiften skall vi anpassa mätdata till en enkel fysikalisk modell. Greve Rumford utförde 1798 följande experiment. Ett kanonrör uppvärmdes till 130° F genom att en trubbig borr vreds runt av hästar i 30 minuter. Sedan fick röret svalna medan man då och då mätte temperaturen T i röret. Man fick följande mätserie:

t (min)	4	5	7	12	14	16	20	24	28	31	34	37.5	41
T (F)	126	125	123	120	119	118	116	115	114	113	112	111	110

Den omgivande temperaturen T_{omg} uppmättes till 60° F. Med denna informationen kan vi beräkna temperatursutvecklingen med hjälp av Newtons avsvalningslag,

$$\frac{\mathrm{d}T(t)}{\mathrm{d}t} = -\beta(T(t) - T_{omg})$$

där β är värmekapaciteten. Vårt mål att är att bestämma β för kanonen utifrån mätdatan. Vi skall använda tre metoder.

1. Differentialekvationen ovan har lösningen

   $$T(t) = T_{omg} + (T_0 - T_{omg})e^{-\beta t}, \quad t \geq 0.$$

   Skriv om den här ekvationen som ett *linjärt* problem i den okända parametern β. Det vill säga, gör en omskrivning med hjälp av logaritmlagarna så att vi får ett problem på formen

   $$A(t)\beta = b(T),$$

   där $A(t)$ är en funktion av enbart t och $b(T)$ är en funktion av enbart T. Använd sedan \-rutinen i Matlab för att uppskatta β.

2. Använd Matlab-rutinen `lsqnonlin` för att lösa det ickelinjära problemet

   $$\min_{\beta} \|T - \big(T_{omg} + (T_0 - T_{omg})e^{-\beta t}\big)\|_2^2.$$

   `lsqnonlin` kräver att man ger en första gissning β_0 på vad β kan vara. Vad är ett rimligt val av β_0 här?

3. Ovanstående approximationer stämmer inte så bra (om ni plottar upp mätvärden och approximationer). Kanske har några värden blivit fel avskrivna (något kan ha tappats bort under de 200 år sedan experimentet utfördes). Antag nu att den omgivande temperaturen T_{omg} var felaktigt avskriven. Använd `lsqnonlin` för att lösa det ickelinjära problemet

   $$\min_{\beta, T_{omg}} \|T - \big(T_{omg} + (T_0 - T_{omg})e^{-\beta t}\big)\|_2^2.$$

Plotta slutligen samtliga versioner av $T(t)$ samt mätdatan i samma diagram. Vilken modell kan man betrakta som mest realistisk i det långa loppet?

2.1.2 Anpassning av en potensfunktion till mätdata

Vi skall nu anpassa en potenslag (engelska *power law*) $y = ax^b$ efter mätdata. Mätdatan är given som

```
x = [1.000e-03
     1.584e-02
     2.511e-01
     3.981e+00
     6.309e+01
     1.000e+03];
y = [3.155e-04
     1.787e-02
     3.475e-01
     9.596e+00
     5.388e+02
     7.147e+03];
```

Vi skall alltså bestämma parametrarna a och b. Vi gör detta återigen med hjälp av tre olika metoder.

1. Bestäm a och b genom att logaritmera och sedan lösa ett *linjärt* minstakvadratproblem.

2. Använd `lsqnonlin` för att lösa det ickelinjära problemet

$$\min_{a,b} \|ax^b - y\|_2^2.$$

3. Använd `lsqnonlin` för att lösa det ickelinjära problemet

$$\min_{a,b} \|\frac{ax^b}{y} - 1\|_2^2.$$

Beräkna också storleken på residualen i samtliga fall. Det vill säga, om (`a_approx`, `b_approx`) är de tal som ni har funnit med hjälp av minstakvadrat-metoden eller `lsqnonlin`, beräkna hur mycket eran uppskattning skiljer sig ifrån mätdatan genom att skriva

```
approx = a_approx*x.^b_approx;
res    = norm(y-approx)
```

Vilken uppskattning är bäst i det här avseendet?

Plotta slutligen mätdatan och dina tre uppskattade potensfunktioner i samma `loglog`-diagram. Kan man fortfarande säga att den uppskattningen med lägst residual var den som bäst anpassades till datan?

2.2 System av icke-linjära ekvationer

Syftet med denna laboration är att ni ska lära er att ställa upp Newtons metod för ett system av ekvationer. Antag att vi har tre stycken sändare med känd position. Exempelvis kan de sitta på toppen av varsin radiomast. Var och en av sändarna kommunicerar med en mottagare som befinner sig någonstans i $\mathbb{R}^3$. Vi vet hur långt mottagaren befinner sig ifrån vart och en av de tre sändarna. Vårt mål är att ta fram exakt vilken punkt mottagaren befinner sig på. Problemet kan illustreras med figuren nedan.

En illustration av problemet att finna mottagarens position. I mitten av vart och en av de tre sfärerna sitter en sändare. Vi vet att mottagaren befinner sig på ett avstånden r_1 från sändare ett, r_2 från sändare två och r_3 från sändare tre. De tre sfärerna har radie r_1, r_2 respektive r_3. Det innebär att sändaren befinner sig någonstans där de tre sfärerna skär varandra.

1. Som första uppgift skall ni bestämma hur många lösningar problemet kan tänkas ha.
2. Formulera sedan det ekvationssystem som behöver lösas för att hitta mottagarens position. Antag att den k:te sändaren har mittpunkt i (x_k, y_k, z_k) och befinner sig på avståndet r_k från mottagaren, $k = 1, 2, 3$. Kalla mottagarens koordinater för $\mathbf{p} = (p_1, p_2, p_3)$.
3. Skriv om ekvationssystemet i uppgift 2 på formen $\mathbf{f}(\mathbf{p}) = \mathbf{0}$ och skriv en `while`-sats som utför Netwons metod för problemet,

   $$\mathbf{p}_{n+1} = \mathbf{p}_n - J^{-1}(\mathbf{p}_n)\mathbf{f}(\mathbf{p}_n).$$

   Här skall ni bestämma Jacobianen J för hand genom att derivera $\mathbf{f}(\mathbf{p})$ med avseende på $\mathbf{p}$. Som bekant är Newtons metod en algoritm som hittar nollställen till system av ekvationer. Algoritmen är klar när två successiva gissningar $\mathbf{p}_{n+1}$ och $\mathbf{p}_n$ befinner sig en tolerans ε ifrån varandra. Sätt $\varepsilon = 10^{-5}$ och använd följande data för sändarnas positioner:

```
x = [ 1.23,  0.12, −0.22]';   % x−koordinaterna for sandarna
y = [ 0.01,  0.98,  0.02]';   % y−koordinaterna for sandarna
z = [−0.11, −0.12,  1.76]';   % z−koordinaterna for sandarna
r = [ 1.22,  0.98,  1.52]';   % avstanden mellan sandare och mottagare
```

Testa följande tre punkter som p_0:

```
p1 = zeros(3, 1);
p2 = ones(3, 1),
p3 = [3.407007110432360e−01
     3.907070706849901e−01
     4.758069306448354e−01];
```

Får vi samma lösning för alla tre gissningar? Hur många iterationer krävs det för att hitta en lösning om vi börjar i `p1`, `p2` respektive `p3`? Om det är någon av initialigissningarna som sticker ut, kan ni förklara *varför* det händer?

Laboration 3

3.1 Kvadratur och interpolation

I den här övningen skall vi använda Matlabs `integral`-rutin[2] för att beräkna integraler. Vi skall också öva på att skriva om integralproblem så att de blir numeriskt hanterbara vid svårare fall. Till sist så skall vi se över tre olika metoder för interpolation av mätdata.

3.1.1 Beräkning av integraler med singulär integrand

För hand beräknar vi enkelt att

$$\int_0^1 x^{-7/10}\mathrm{d}x = \frac{10}{3}\left[x^{3/10}\right]_0^1 = \frac{10}{3}.$$

När vi försöker göra detsamma i Matlab får vi däremot ett felmeddelande:

```
>> integral(@(x) x.^(-0.7),0,1)
Warning: Infinite or Not-a-Number value encountered.
> In integralCalc/iterateScalarValued (line 349)
  In integralCalc/vadapt (line 132)
  In integralCalc (line 75)
  In integral (line 88)
ans =
   Inf
```

Detta på grund av att vi naivt delar med noll i den vänstra integrationsgränsen. Vi kan rundgå detta på två sätt; genom partiell integration och genom att genomföra ett variabelbyte. Vi skall nu öva på dessa två metoder.

1. Beräkna integralen
$$\int_0^1 x^{-9/10}\cos x\mathrm{d}x$$
genom att med hjälp av partiell integration föra över den till en form så att vi slipper dela med noll. Använd sedan `integral`.

2. Beräkna samma integral som i (1) genom att göra variabelbytet $x = y^p$ för ett väl valt p. Använd sedan `integral`.

3. (*) Vissa singulariteter kan hanteras av `integral` eftersom de är kända undantagsfall. Ett exempel på detta är

```
>> integral(@(x) log(x),0,1)
ans =
   -1.0000
```

trots att $\log(0) = -\infty$. Med vetskap om detta, beräkna integralen
$$\int_0^1 \frac{\log x}{x^{0.85} + x^{0.9} + x^{0.95}}\mathrm{d}x$$
genom att göra variabelbytet $x = y^p$ för ett väl valt p.

[2]Dokumentation: `https://se.mathworks.com/help/matlab/ref/integral.html`

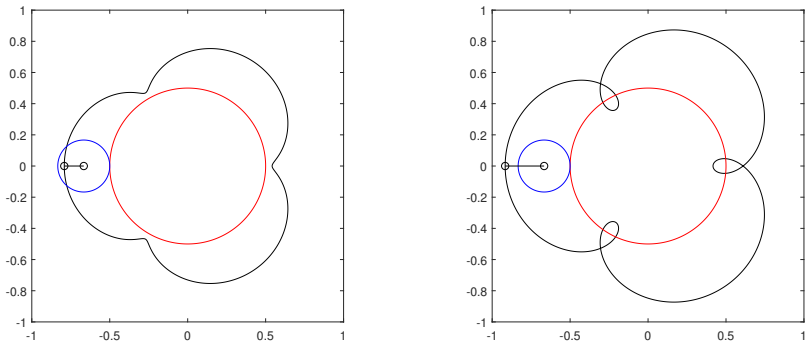
3.1.2 Mer avancerad integration

I den här uppgiften skall vi använda `integral` som ett led i ett större problem.

Antag att vi har tillgång till en maskin som kan skapa bordsskivor som är perfekt formade som en *epicykloid*[3] med tre utbuktningar. En epicykloid med N utbuktningar skapas av att vi låter en liten cirkel med radien $r = R/N$ rulla längs randen på en stor cirkel med radien R. Den parametriseras med avseende på parametern $0 \leq \theta \leq 2\pi$ som

$$x(\theta) = cr\cos(\theta) - qr\cos(c\theta),$$
$$y(\theta) = cr\sin(\theta) - qr\sin(c\theta),$$

där $c = (R+r)/3$ och vi har att $N = 3$. q är en parameter som bestämmer hur accentuerade utbuktningarna skall vara, se figuren nedan.



Två exempel på bord som vår bordmaskin kan skapa. Här har vi att $q = 0.75$ respektive $q = 1.5$.

Maskinen är inställd på $R = 0.5$ meter, vilket betyder att det enda sättet vi kan ändra på bordets form är att ändra på q. Med allt detta sagt har vi kommit fram till problemet ni skall lösa.

Antag att vi har en tjusig bordslist på 4.25 meter som vi absolut vill sätta på ett epicykloidformat bord. För vilket q har vi att omkretsen på bordet är exakt 4.25 meter?

Ledning: Omkretsen kan betraktas som en funktion av q, det vill säga $O(q)$. Omkretsen ges i vanlig ordning av integralen $O(q) = \int_0^{2\pi} \sqrt{(x'(\theta))^2 + (y'(\theta))^2}\mathrm{d}\theta$. Ni kan sedan använda `fsolve` för att lösa problemet $O(q) - 4.25 = 0$.

[3] Se `https://en.wikipedia.org/wiki/Epicycloid` för mer information och en trevlig animation.

3.1.3 Interpolation av mätdata

I den här uppgiften skall vi anpassa tre typer av interpolationskurvor till mätdata. Datan är given som

```
t = linspace(-2, 3, 8)';
y1 = exp(-t.^2);
y2 = [-1 -1 -1 -1 1 1 1 1]';
```

där `t` är tiderna som mätvärdena `y1`, `y2` uppmättes. Ni skall nu använda Matlab-rutinen `interp1`[4] för att anpassa dels en **linjär interpolation**, en **splineinterpolation** och en **shape-preserving interpolation**. I den första dataserien har ni en underliggande funktion $\exp(-t^2)$ att jämföra med när ni studerar för- och nackdelar med de olika metoderna. Plotta alla era tre interpolationer tillsammans med mätdatan i varsin figur för `y1` och `y2`. Avgör vilken interpolationsteknik som matchar datan bäst i båda fallen och motivera era svar.

[4]Dokumentation: `https://se.mathworks.com/help/matlab/ref/interp1.html`

3.2 Modellering och simulering med ordinära differentialekvationer

I den här uppgiften skall vi simulera en laddad partikel som rör sig i ett elektromagnetiskt fält. Vi antar att vi har N stycken punktladdningar placerade i $\mathbb{R}^2$ på koordinaterna $\mathbf{p}_i$, $i = 1, \ldots, N$. Vi har också en fri partikel som rör sig i $\mathbb{R}^2$ under påverkan av elektromagnetiska krafter från punktladdningarna. Dess position vid tiden t ges av $\mathbf{r}(t) = (x(t), y(t))$, en vektorvärd funktion som uppfyller systemet av differentialekvationer

$$\ddot{\mathbf{r}}(t) = c \sum_{i=1}^{N} \frac{\mathbf{r} - \mathbf{p}}{\|\mathbf{r} - \mathbf{p}\|^3}. \tag{3.1}$$

Här är c är en konstant som tar materialegenskaper (och annat som krävs för att ekvationen skall få rätt enheter) i beaktning.

1. Skriv om (3.1) som ett första ordningens system genom att införa hjälpfunktionen $\mathbf{q}(t) = \dot{\mathbf{r}}(t)$. Vi kan sedan skapa en kolonnvektor `y=[r;q]`. $\mathbf{y}(t)$ uppfyller ett första ordningens system av differentialekvationer $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$.

2. Skapa en Matlab-funktion `dydt = f(t,y)` som implementerar systemet ni tog fram i föregående uppgift. Som stöd får ni följande kodstubb:

```
function dydt = f(t,y)
    global p % Anvand er av globala variabler for punktladdningarnas
    global c % positioner p och konstanten c
    %
    % Skapa hogerledet i systemet. Se till att det fungerar oavsett
    % antalet punktladdningar! Exempelvis genom att lagra punktladd-
    % ningarnas positioner som kolonner i en matris, och sedan lopa
    % igenom denna matris med en for-loop.
    %
    dydt = % skall fyllas i av er
end
```

3. Använd `ode45`[5] för att lösa systemet $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ med hjälp av funktionen ni formulerade i den föregående deluppgiften. Vi antar att det existerar två punktladdningar, en i $p_1 = (-1, -0.9)$ och en i $p_2 = (1, -0.9)$, och att partikeln börjar i vila i positionen $\mathbf{r}(0) = (0.5, -0.9)$. Följande kod kan vara till hjälp:

```
global p
global c
p     = [-1,1;-0.9,-0.9];
c     = 10^(-5);
y0    = [0.5;-0.9;0;0];
t_max = 10^3;
[t,y] = ode45(@f,[0,t_max],y0);
```

[5]Dokumentation: `https://se.mathworks.com/help/matlab/ref/ode45.html`.

4. I den föregående uppgiften kommer partikeln att oscillera längs linjen $x = 0.5$ i xy-planet. Om ni alltså får ut att partikelns position är konstant i x-koordinaten är ni på rätt spår. För att göra saker mer intressant så skapar vi en *potentialgrop* som partikeln kan fastna i genom att placera ut N stycken punktkällor länges randen på enhetscirkeln på följande sätt:

```
theta = linspace(0,2*pi,N);
p     = [cos(theta);sin(theta)];
```

Pröva att variera startposition och starthastighet genom att variera `y0`. Efter att du har tagit fram lösningen kan du animera lösningen med hjälp av följande kodsnutt:

```
for i=1:size(y,1)
    clf
    plot(p(1,:),p(2,:),'r*')
    hold on
    plot(y(i,1),y(i,2),'bo')
    axis([-2.5 2.5 -2.5 2.5])
    axis equal
    drawnow;
    pause(1/60) % 60FPS
end
```

Lek gärna runt med olika kombinationer av begynnelsevillkor, samt andra konfigurationer på punktladdningarna.