TMV156/TMV155E Inledande matematik E, 2010

DATORÖVNING 4 —

Instruktioner syfte och innehåll

Skapa en ny filkatalog ("directory") `Lab4` för denna övning. Gör alltid uppgifterna i script-filer eller funktionsfiler om inget annat står. Lista gärna alla kommandon du använder under den här laborationen. Du kommer att ha nytta av samtliga i framtiden.

Efter utförd laboration bör du ha förstått hur Newtons metod fungerar och hur den används för att hitta såväl nollställen som extrempunkter. Dessutom bör du förstå hur `for`-loopar fungerar och inse nyttan av dessa i programmeringssammanhang. Du bör också fortsatt tänka på betydelse av in- och utargument i funktionsanrop samt betydelsen av det faktum att variabler som deklareras inuti funktionsfiler är lokala.

Under den andra föreläsningen visade Fredrik några olika `for`-loopar som kan vara värda att sutdera nu! De ligger på Fredriks hemsida.

1. Uppgifter

1.1. **Derivator.** Se till att du har fungerande funktioner `derivera.m` från den förra laborationen och `andraderivata.m` enligt den senaste inlämningsuppgiften.

1.2. **Newtons metod.** Newtons (Newton-Raphsons) metod finns beskriven i Adams *Calculus*. Man skapar en följd av tal $x_0, x_1, x_2, \dots x_{i-1}, x_i, x_{i+1}, \dots$ genom att succecivt beräkna

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{1}$$

där x_0 är en första gissning. Under vissa förutsättningar på funktionen f och dess derivata så kommer man för varje steg komma närmare lösningen till problemet $f(x) = 0$ ju fler steg man tar med Newtons metod. Om man vill hitta minimum eller maximum till en funktion kan man istället utföra Netons metod på funktionens derivata, dvs

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}. \tag{2}$$

Då hittar man istället lösningar till problemet $f'(x) = 0$ som ju är ett nödvändigt vilkor för ett minima eller maxima.

1.2.1. *Implementera Newtons metod.* Skriv en funktion `newton.m` tar två inargument `f` och `x0` där det förra är ett funktionshandtag och det senare en startgissning, en punkt nära det x som löser $f'(x) = 0$, och som tar ett steg med denna metod. Vilken av (1) och (2) ska du alltså implementera? Funktionen ska returnera den förhoppningsvis bättre approximationen `x1`. Detta är första steget mot att skriva egna funktioner som hittar minima till funktioner. Newtons metod kommer att återkomma i den här laborationen, i nästa inlämningsuppgift och i nästa kurs. Se till att alla har en fil sparade på sitt datorkonto ifall ni är flera som jobbar ihop. Newtons metod är en väldigt viktig metod då den typ av problem som man kan lösa med den uppkommer ofta och ibland på oväntade ställen i datorberäkningar. Testa lösaren på $g(x) = (x.^2 - 1)e^{-x^2}$ med lite olka startgissningar och notera hur metoden konvergerar mot de olika extrempunkterna beroende på var man börjar. Vad händer när man startar på $x = -0.5$ och varför?

Date: 2009–10–06, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

1.3. **for-loopen.** For-loopen är en av de viktigaste programmeringskonstruktionerna och används så fort vi vill upprepa någon typ av operationer ett på förhande känt antal gånger. Uppgifterna nedan syftar till att förstå hur den fungerar och till att visa några standardanvändningar. Tänk också noggrant igenom hur indexering av vektor och matriser fungerar om du känner dig osäker på det och giv akt på hur vi bygger ihop nya matriser av gamla nedan. Det här blickar tillbaka på saker från framför allt den första laborationen.

1.3.1. *En första for-loop.* Vad gör följande for-loop?

```
N=10;
for x=1:1:N
   x
end
```

Det här programmet utför samma sak. Se till att du förstår likheten och skillnaden!

```
N=10;
s=1:1:N;
for x=s
   x
end
```

Vilken vektor som helst går att använda, inte bara heltal:

```
N=10;
s=.75:0.5:N;
for x=s
   x
end
```

```
N=10;
s=rand(1,N); %Skapar N st slumptal i en radvektor
for x=s
   x
end
```

1.3.2. *En andra for-loop.* Vad gör följande for-loop?

```
N=10;
s=0;
for n=1:1:100
   s=s+n;
end
```

Vad är värdet på variabeln s när programmet körts klart? Det kan man enkelt räkna ut!

1.3.3. *Summa.* Man kan visa att summan

$$\Sigma_{k=0}^{N} x^k = \frac{1-x^{N+1}}{1-x} \tag{3}$$

och att om $|x| < 1$ så är

$$\Sigma_{k=0}^{\infty} x^k := \lim_{N\to\infty} \Sigma_{k=0}^{N} x^k = \frac{1}{1-x}.$$

men om $|x| \geq 1$ så är den divergent, gränsvärdet existerar inte. Skriv en funktionsfil `summa.m` med invariablerna `x` och `N` som beräknar summan i (3) *genom att addera en term i taget*. Ledning:

```
s=s+x^k;
```

Sriv `format long` vid kommandoprompten så att du får alla decimaler utskrivna och jämför resulatet då du summerar $N = 75$ respektive $N = 100$ termer i summan för något $x < 1$ och något $x > 1$. Jämför med vad du får i högerledet i (3).

1.4. **Newton igen.**

1.4.1. *Automatisering av stegningen.* Skriv om din fil `newton.m` så att den tar ett till inargument `N` som är antalet steg man vill ta med Newtons metod och gör så att programmet gör det i en `for`-loop. Testa på någon funktion!

1.5. **Att bygga upp matriser och vektorer.** Följande program skapar en vektor med N slumptal. Funktionen 'tic' startar en tidtagning och 'toc' stoppar den och returnerar tiden. Fundera noga över hur skapandet av vektorn x fungerar i de fyra fallen! Varför tar de två första så oerhört mycket längre tid? [1]

```
N=1e5; %Ett stort nummer
tic %Startar tidtagare
x=[]; %Skapar en tom matris
for k=1:N
    x=[x rand];   %Ett sätt att bygga upp matriser/vektorer
end
tid(1)=toc; %Mäter av tiden

tic %Startar tiden från noll igen
x=[];
for k=1:N
     x(k)=rand;  %Ett sätt som i praktiken fungerar likadant som det ovan.
end
tid(2)=toc;

tic
x=zeros(1,N);
for k=1:N
    x(k)=rand;   %Det snabba och i allmänhet rätta sättet!!!
end
tid(3)=toc;

tic
x=rand(1,N);    %I just det här fallet kan man göra så här=jättesnabbt!
tid(4)=toc;
```

Den första loopen använder sig av ett specialfall av matrisbyggande. Om A och B är två matriser med lika många rader (men eventuellt olika många kolumner) så kan vi deklarera en ny matris

`C=[A B];`

Vad kan man, analogt, göra om A och B har lika många kolumner men olika många rader?

1.5.1. *Spara alla punkter Newtons metod besöker.* Skriv ett nytt program `newton2.m` som sparar alla punkter som Newtons metod besöker på väg från startgissningen till den den N:te och sista punkten. Testa på $g(x)$ och $sin(x)$.

1.5.2. *Delsumma.* Skriv en ny funktion `summa2.m` som räknar ut varje delsumma av den aktuella summan och returnerar en vektor där det första elementet är 1, det andra $1+x$, det tredje $1+x+x^2$, det tredje $1+x+x^2+x^3$ och så vidare. Ledning:

`s(i+1)=s(i)+x^(i+1).`

Du måste göra ett litet trick i början för att loopen ska komma igång rätt.

[1] Svar: I de första så skapas en ny vektor i varje varv dit den gamla vektorn kopieras och nästa element läggs till. Detta för att vektorn inte har plats för fler element så som den är. I det tredje fallet skapas en vektor med rätt antal platser från början och värdet av varje plats ändras från noll till det önskade i varje varv i loopen. På så sätt behöver inget kopieras.

1.5.3. *Vektorisera summoberäkningen.* Skriv om funktionen `summa.m` så att den kan ta en vektor med flera värden på x och returnera en $M \times N$-matris där M är antalet element i x och N är antalet termer i summan. Ledning: `M=length(x); S=zeros(M,N)` och
`S(:,i+1)=S(:,i)+x.^(i+1).`
Det här sättet att bygga upp matriser kommer vi använda oss av en del i nästa kurs!

1.6. **En mer kompetent Newtonlösare med `while`-konstruktion.** Man vill i allmänhet inte stanna Newtons metod efter ett fixt antal steg utan när felet är tillräckligt litet. Ett sätt att göra det är att fortätta tills två besökta punkter efter varandra är tillräckligt nära varandra. (Det betyder inte nödvändigtvis att man är nära den rätta lösningen, men under vissa förutsättningar kan man visa att det är fallet.) För att på ett smidigt sätt åstadkomma detta kan man använda sig av `while`-loopar som kan ses som en kombination av `if`-satser och `for`-loopar. Vi har inte diskuterat `while`-loopen men den kommer som en viktig del av nästa kurs. För den som vill få en första glimt och göra en mer användbar Newton-lösare så kan man skriva så här:

```
function  [???]=newton(???, x0,max_error) %Istället för N har vi max_error

x1=2*max_error+x0; %Gör så att det logiska påståendet i while-loopen är
                   %sant första gången.
???? %Eventuell annan kod som kan behövas.
while norm(x1-x0)>max_error  %Om x1 är längre från x0 än det maximalt tillåtna
                             %avståndet max_error så tar vi ett till steg med
                             %Newtons metod.
   x1=x0; %Vi sparar det gamla värdet på x0 i x1
   x0=??? % Och tar ett steg med Newtons metod varvid vi ändrar x0
end
```

Konstruktionen med `while` följt av ett logiskt påstående repeterar det som kommer efter detta om och så länge som det logiska påståendet är sant. Funktionen `norm` beräknar absolutbeloppet, inte bara för tal utan också för vektorer. När vi utvecklar det här för att fungera i flera dimensioner nästa kurs så kommer det behövas. Funktionen `abs` är vektoriserad och beräknar alltså bara beloppet av varje enskilt element i vektorn.

1.6.1. *Implementera!* Färdigställ programmet ovan!

1.7. **Fixpunktsprogram.** Newtons metod är ett specialfall av fixpunktsiteration där man löser ett problem

$$x = g(x) \tag{4}$$

genom att stega sig fram $x_1 = g(x_0), x_2 = g(x_1) \dots x_{k+1} = g(x_k), \dots$. Om $g : I \to I, I = [a, b]$ är en kontraktion, dvs om $|g(x) - g(y)| \leq K|x - y|$ för något $K < 1$ så konvergerar sekvensen tal $(\{x_0, x_1, X_2, \dots\})$ mot lösningen $\hat{x}$ till 4 och felet

$$|x_k - x_0| \leq \frac{|x_k - x_{k-1}|}{1 - K}. \tag{5}$$

Notera att varje ekvation kan skrivas på formen (4). Om man vill lösa $f(x) = 0$ så kan man t.ex. välja $g(x) = x + f(x)$. Om detta är en kontraktion är dock en helt annan sak.

1.7.1. *Fixpunktsprogram.* Spara om programmet från Uppgift 1.6.1 som `fixpunkt.m` och skriv om det så att det använder en generell fixpunktsiteration istället.