TMV156 Inledande matematik E, 2011

DATORÖVNING 3 — FUNKTIONSANROP OCH OPTIMERING

Instruktioner syfte och innehåll

Skapa en ny filkatalog ("directory") `Lab3` 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.

Laborationen syftar till att skapa ytterligare vana vid grafritning samt att förstå funktionsfilers uppbyggnad och användande av olika antal in- och utargument samt olika typer av argument. Du skall också också bekanta dig med Matlabs färdiga funktioner för att hitta nollställen samt minima och maxima till funktioner. I synnerhet presenteras funktionsfilerna `roots.m`, `fzero.m`, `fminsearch.m`, `fminbnd.m`, `min.m` och `max.m`. Från förut dyker dessutom några elementära matematiska funktioner upp (t.ex `sin`, `exp`) samt `plot`, `legend.m`, `title.m`, `linspace.m`. Mera perifiera saker är `nargin` och `num2str.m`. Vi presenterar också en hemmabygd funktion `sinfaf.m` och Matlabs `if-elseif-else`-konstruktion. Därvidlag dyker också olika jämförelseoperatorer upp.

Vi räknar inte med att du hinner med alla uppgifter under laborationen utan du förväntas göra klart det hela hemma innan nästa laboration.

1. Uppgifter – Att skriva funktionsfiler med olika in- och utargument

1.1. **Matematiska funktionsfiler.** Skapa en *funktionsfil* som tar ett inargument x och returnerar ett värde $y = (x^2 - 1)e^x$. Se till så att Matlab inte skriver ut uträkningen i kommandofönstret om man inte vill det. Använd semikolon. Plotta funktionsgrafen på lämpligt intervall. Vad är skillnaden mellan en (icke-anonym) funktion och en anomym sådan?

1.2. **Två in- och utargument.** Skriv en funktionsfil `sumdiff.m` som tar två tal som inargument och returnerar två utargument: de båda argumentens summa och differensen mellan det första och det andra inargumentet. Testa med ett och två utargument för några olika värden på inargumenten.

1.3. **Sinus med olika frekvens, amplitud och fas.** Det finns två syften med den här uppgiften: Att du ska förstå antalet arguments koppling till hur programmet är skrivet och att få intution för sinusfunktioner och deras frekvens, amplitud och fas.

Ladda ned matlabfunktionen `sinfaf.m` från kurshemsidan eller Fredriks hemsida och spara den i katalogen `Lab3`. Skriv `help sinfaf` i kommandofönstret. Vad gör funktionen? Titta på hela filen med hjälp av `edit sinfaf`!
Första "aktiva" raden lyder

```
function [y,dy]=sinfaf(x,vfreq,amp,phase)
```

- Hur många variabler kan man som mest få ut av funktionen?
- Hur många inargument kan man som mest skicka med?
- Försök förstå vad hela funktionsfilen gör för något!

Testa följande kommandosekvenser i en skript-fil. Om du skriver alla sekvenserna i samma skriptfil kan du separera sekvenserna med %% så att de bildar *celler*. Då kan du köra varje cell för sig genom att välja **cell** och sedan **evaluate current cell** i panelen i editorn. Tänk efter vad som kommer hända innan du exekverar (kör) programmet!

Date: 2010–09–27, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

Fungerar ett in- och utargument?

```
x=linspace(-pi,pi);
freq=5; %Egentligen vinkelfrekvensen som är 2*pi*frekvensen
amp=2;
phase=pi/4;
freq2=10;
ut1=sinfaf(x);
plot(x,ut1)
hold on
```

Två inargument?

```
ut2=sinfaf(x,freq);
plot(x,ut2,'r')
```

Tre inargument?

```
ut3=sinfaf(x,freq,amp);
plot(x,ut3,'g')
```

Fyra inargument? Vad händer i den andra plotten?

```
ut4=sinfaf(x,freq,amp,phase);
plot(x,ut4,'k')
plot(ut4,x,'k')
```

Fem inargument?

```
ut5=sinfaf(x,freq,amp,phase,freq2);
```

Två utargument? Notera hur man kan skapa strängar av dubblarna och bygga ihop strängar t.ex. för att använda till `title`, `legend` etc.

```
figure(2)
[ut6,ut7]=sinfaf(x,freq,amp,phase);
plot(x,ut6,x,ut7)
str1=[num2str(amp) 'sin(' num2str(freq) '(x-' num2str(phase) '))']%!!??
%help num2str
str2=[num2str(freq*amp) 'cos(' num2str(freq) '(x-' num2str(phase) '))']
legend(str1,str2)
```

Tre utargument?

```
[ut8,ut9,ut10]=sinfaf(x,freq2,amp,phase);
```

Jämför med en annan vinkelfrekvens.

```
[ut8,ut9]=sinfaf(x,freq2,amp,phase);
figure(3)
plot(x,ut8,'r:',x,ut9,'k-o')
str3=[num2str(amp) 'sin(' num2str(freq2) '(x-' num2str(phase) '))']
str4=[num2str(freq2*amp) 'cos(' num2str(freq2) '(x-' num2str(phase) '))']
legend(str3,str4)
```

Vi kastar om inargumenten. Det är väldigt viktigt att förstå vad som sker här!

```
figure(4)
[ut10,ut11]=sinfaf(x,amp,phase,freq2);
plot(x,ut10,x,ut11)
```

Ändra gärna lite på fas och amplitud också! Var säker på att du kan förklar hur grafen förändras när de olika värdena minskar/ökar. Det är något särskilt när fasföskjutningen är π respektive $\frac{\pi}{2}$. Vad?

Läs igenom filen och se hur `if-elseif-else`-satser använd för att hantera olika antal argument!

1.4. **Derivata.** Hitintills har vi sett matlabfunktioner som har tagit inargument som är tal eller listor med tal (vektorer). I den här uppgiften ska du skriva en funktion som tar ett funktionshandtag (det vill säga en adress till en anonym- eller ickeanonym funktion) och använder det för att räkna ut derivatan av denna funktion. Du ska alltså skriva en funktionsfil som deriverar en given funktion med hjälp av den symmetriska differenskvoten

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

Laborationshandledaren förklarar varför den ekvationen är bättre än den som man får direkt ur derivatans definition! Kalla funktionen `derivera.m` och låt den ta tre inargument, ett funktionshandtag f, en steglängd h samt punkten där derivatan ska beräknas x, och returnera derivatans värde df. Se till att funktionen är vektoriserad. Plotta resultatet för två olika frekvenser på en sinusfunktion! Överkurs är att sätta ett defaultvärde på h med hjälp av `nargin` och en `if-sats`. Ett bra värde är $h = 10^{-6}$. Titta hur det fungerar i `sinfaf.m`!

DEN HÄR FUNKTIONEN KOMMER DU BEHÖVA NÄSTA LABORATION! Om ni jobbar tillsammans, se till så att alla har den på sin dator ifall någon blir sjuk etc.

2. Optimering och Ekvationslösning – att anropa Matlabs funktioner

2.1. **Hitta nollställen med Matlabs färdiga funktioner.** Om man vill lösa ekvationen $f(x) = 0$ i Matlab så finns (minst) två alternativ till hands. För godtycklig funktion kan man använda `fzero`. Om man vill hitta nollställen till ett polynom kan man dessutom använda `roots`. För att använda `roots` behöver man skapa en vektor med koefficienterna till polynomet med den framför monomet (x^n är ett monom för varje $n \in \mathbb{N}$, av fler monom har det med högst n högst grad) av högst grad först. Om någon koefficient är noll måste man lägga in en nolla i vektorn (varför?). Funktionen `fzero.m` kan man ge ett funktionshandtag och ett begynnelsevärde där den ska starta sitt sökande efter nollstället. Alternativt kan man ge den ett intervall inom vilket nollstället finns. Då får det bara finnas ett nollställe mellan.

Använd hjälp- eller dokumentationsfunktionen för att ta reda på hur dessa båda funktioner används och använd dem på följande funktioner. Jämför med de analytiska lösningarna! Hur stora blir felen?

(1) $p(x) = x^3 - 6x^2 + 11x - 6$, gissa heltalslösningarna med hjälp av de numeriska.
(2) $q(x) = x^3 - 3x^2 + 3x - 1$, Pascals triangel ger dig trippelroten direkt!
(3) $g(x) = (x^2 - 1)\exp(x)$, inga problem att hitta rötterna?! Har exponentialfunktionen någon rot?

Titta hur funktionerna `fzero` och `roots` ser ut genom att skriva `edit fzero` och `edit roots` bläddra igenom programmet. Titta särskilt på funktionsdeklarationen under hjälptexten! Det är ingenting som skiljer dessa från de program du själv skrivit utom komplexiteten. Anrop av dessa funktioner är exakt samma sak som anrop till de du skrivit själv.

2.2. **Minimering.** Ett oerhört vanlig problem i teknik, ekonomi, fysik, ... är att man vill hitta minimum eller maximum till någon funktion. Matlab har flera färdiga funktioner som gör detta. Gemensamt för dem och nästan alla andra optimeringsfunktioner på marknaden är att de bara hittar minima till funktioner. Hur gör man om man vill hitta ett maxima?

De funktioner som vi ska titta på är `fminsearch` och `fminbnd`. Den förra letar efter minima i närheten av en startgissning x_0 och den andra letar efter minima på ett intervall $[a, b]$ som specificeras som en vektor `X0=[a b]` där a och b är reella tal. Använd hjälpfunktionen för att förstå hur de ska användas och vad du får ut för svar av dessa. Testa dem sedan på funktionerna från föregående fråga. Försök hitta alla lokala minima och maxima!

2.3. **Matlabs `min` och `max`.** Funktionen `min` kan användas för att hitta det minsta talet av två (`min(a,b)` där a, b är reella tal) eller det minsta talet i en vektor x (`min(x)`) eller det minsta talet i varje rad i en matris A (`min(A,[],2)`) eller det minsta talet i varje kolumn (`min(A,[],1)`). Den kan även jämföra två lika stora vektorer/matriser x, y och returnera ytterligare en vektor/matris z med det elementvis minsta talet (`z=min(x,y)`). Om vi frågar efter två utargument så får vi om vi inte jämför två tal eller matriser ut platsen som minimat är funnet på:

```
x=randn(1,100);
[M,ind]=min(x);
santfalskt=M==x(ind)
```

I det sista uttrycket evalueras den logiska satsen `M==x(ind)` som har värdet sant. Därför sätts variabeln *santfalskt* till sant (1) och blir av typen logisk variabel.

Det här leder till ett mycket primitivt och farligt sett att hitta minima. Om vi har ett funktionshandtag f så och vet att minimat finns mellan -1 och 2 säg så kan vi utföra följande kommandosekvens:

```
x=linspace(-1,2);
y=f(x);
[minimat,ind]=min(y);
x0=x(ind)
logik=f(x0)==minimat
```

Varför är det så farligt? Det hänger ihop med att det är farligt att lita på datorplottar om man inte vet något om funktionerna i fråga. Testa följande med `max` som fungerar helt analogt:

```
f=@(x)(1/(sqrt(2*pi)*1e-1)*exp(-((x-1)/1e-1).^2/2));%normalfördelningens
                                                    %frekvensfunktion
dx=.3;
y=0:dx:5;
z=f(y);
[m,ind]=max(z);
x=linspace(0,5,1e5);
plot(x,f(x),y(ind),m,'*r')
```

Uppenbarligen är det hittade maximat inte funktionens maximum! Testa

```
hold on
plot(y,z,'*k')
```

2.4. **Mera optimering!** Gör optimeringsuppgifterna från förra laborationen!

2.5. **Andraderivatan.** Implementera en funktion som beräknar andraderivatan av en funktion med hjälp av funktionen `derivera.m`.

2.6. **Newtons metod.** Kolla upp Newtons metod i Adams *Calculus* och skriv en funktion som tar ett steg med denna metod. Det är första steget mot att skriva egna funktioner som hittar nollställen och minima.