TSRT04: Introduktionskurs i Matlab Datorlektion 2

Version: 29 augusti 2017

Den här datorlektionen handlar mestadels om programmering i MATLAB. De mest grundläggande programmeringsbegreppen introduceras, och vi ger ett förslag på hur man kan angripa den typ av programmeringsproblem som man vanligtvis vill lösa i MATLAB. För dig som har programmerat tidigare är antagligen det mesta bekant, men det är ändå viktigt att lära sig syntax.

Observera: Syftet med denna datorlektion är att du ska lära dig grunderna i MATLAB genom att prova dig fram. Det finns mängder av frågor i detta dokument och vi rekommenderar att du skriver ner dina svar och reflektioner. Då blir det lätt för dig att gå tillbaka om du glömt något. Däremot behöver du inte presentera resultatet för någon. Assistenterna är här för att svara på frågor och ge återkoppling på din kod. Passa på – det finns inga dumma frågor!

1 Förberedelseuppgifter

1. Titta igenom alla momenten i lektion 1 och se till att du minns och har förstått alltihop.
2. Läs igenom avsnitt 2 och gör Uppgift 1.

2 Dokumentation

Redan under datorlektion 1 nämnde vi vikten av att kommentera sin kod. Hur mycket man bör dokumentera och kommentera sin kod beror delvis på användningsområdet:

1. Man vill kunna använda funktionen en annan dag och förstå vad den gör.
2. En kompis vill kunna använda funktionen, och vill då kunna förstå vad den gör och vara säker på hur den fungerar. Din kompis vill också att resultatet presenteras på ett tydligt sätt.
3. Du vill övertyga chefen om att funktionen är så användbar att det lönar sig att köpa in MATLAB till ert företag, eller att din idé förtjänar att patenteras. Koden bör då inte bara vara väldokumenterad, utan också presentera resultatet på ett snyggt, tydligt och övertygande sätt.

Byt ut ”kompis” mot ”examinatorn” eller ”labassistenten” och du får en uppfattning om den ambitionsnivå som du bör ha på laborationerna i den här kursen.

Några saker att tänka på för att åstadkomma tydlig och prydlig kod:

- Skriv din kod i en logisk ordning: ”Först gör vi detta, sedan gör vi detta”. Använd rader som börjar med `%%` för att dela upp koden i delberäkningar, och kommentarer för att tala om vad som händer i de olika delarna.
- Använd beskrivande namn på dina variabler. Följande kod gör samma uträkning som `laneberakning` i avsnitt 5 på datorlektion 1, men är svårare att förstå:

```
function y = test(x1,x2)
% Ett test
a = 0.01;
b = 0.1;
y = x1 + x1*a - x2*b;
```

- Skriv hjälptexter och förklaringar till de in- och utargument som du använder. Hjälptexten skrivs som kommentarer på raden/raderna efter funktionsdeklarationen (`function y = ...`). Detta är dessa rader som visas om du skriver `help filnamn` för din m-fil. (Testa det på exemplet `laneberakning`.)

- Glöm inte att ändra kommentarerna om du ändrar i koden. Felaktiga kommentarer är sämre än inga alls, och gör bara koden svårare att förstå.

Uppgift 1. Komplettera funktionen `kurvplot` från lektion 1 med kommentarer och hjälptexter (gör först klart funktionen om du inte redan har gjort det). Skriv `help kurvplot` för att kontrollera att det ser snyggt och rätt ut.

3 Kontrollstrukturer

Villkor – if-satser

Ofta vill man göra olika saker beroende på värdet på någon variabel. I låneexemplet från datorlektion 1 kanske man vill kontrollera om skulden är avbetald. I MATLAB (liksom i de flesta andra programmeringsspråk) finns därför kontrollstrukturen `if`. Följande exempel visar hur den kan användas för att kontrollera om skulden är avbetald och i så fall sätta den till noll:

```
function ny_skuld = laneberakning(skuld,lon,ranta,...
                                  betalningsgrad)
% ny_skuld = laneberakning(skuld,lon,ranta,betalningsgrad)
% Beräknar skulden nästa månad.
% Mata in skuld, lön, ränta och betalningsgrad

ny_skuld = skuld + skuld*ranta - betalningsgrad*lon;

% Testa om skulden är avbetald:
if ny_skuld <= 0
    ny_skuld = 0;
    disp('Skulden är avbetald!')
else
    disp('Skulden är inte avbetald ännu.')
end
```

Tre nyckelord: `if`, `else` och `end`. Efter `if` står vårt villkor: `ny_skuld<=0` (`<=` betyder ”mindre än eller lika med”). I det här exemplet är villkoret en olikhet, men man kan också kontrollera om två variabler är lika genom ett dubbelt likhetstecken (`==`) eller om de är olika genom `~=`. Om villkoret är sant utförs raden (eller raderna) mellan `if` och `else`, annars utförs kommandona som skrivit mellan `else` och `end`. Allmänt kan man skriva syntaxen som

```
if villkor
  % programrader om villkoret är sant
else
  % programrader om villkoret är falskt
end
```

MATLAB tolkar nollskilda värden som sanna och 0 som falskt. Villkoret kan alltså vara ett uttryck som beräknas till något nollskilt eller 0. Använd `help relop` för att ta reda på vilka jämförelseoperationer som finns i MATLAB.

Passa på att prova några olika villkor:

- Är $\sqrt{3}$ större än 1.5? .....
- Är $\sin^2(t)+\cos^2(t)$ lika med 1 (för godtyckligt t)? Vad säger detta resultat oss om MATLABs noggrannhet? .....
- Testa (med ett uttryck) om e^2 är större än 6 men mindre än 7. ....

Felhantering

Ibland använder man if-satser för att hitta inmatningsfel. Eftersom vi i exemplet ovan lät räntan och betalningsgraden vara inargument kanske vi vill kontrollera att räntan är positiv. I annat fall vill vi inte göra någon beräkning utan skriva ut ett felmeddelande och avsluta funktionen. Till detta används funktionen `error`. Lägg till följande test innan `ny_skuld` beräknas:

```
% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end
```

Resten av funktionen kan vara precis som förut. Prova vad som händer om du matar in en negativ ränta.

Funktionen `warning` kan användas för att varna för mindre allvarliga fel, utan att körningen av funktionen avbryts.

Skriv en if-sats som kontrollerar om den månatliga ökningen av skulden, `skuld*ranta`, är större än månadsbetalningen `betalningsgrad*lon`. Om så är fallet så skriv ut en varningstext om att man betalar för lite.

.....

.....

.....

Uppgift 2. Utöka din funktion `kurvplot` så att den kontrollerar att intervallet anges med det minsta värdet `xmin` före det största `xmax`. Om intervallet angetts på något annat sätt kan du antingen skriva ut ett felmeddelande och avsluta funktionen, eller byta plats på värdena och fortsätta efter att ett varningsmeddelande skrivits ut. Välj själv vilket av alternativen som du vill implementera.

Upprepningar

Som tidigare nämnts vill man ofta köra samma kod (eller åtminstone ungefär samma) ett flertal gånger. I så fall finns det bättre sätt än att skriva samma kodrader flera gånger i skriptet/funktionen.

for-loopar

Antag att Emma inte bara vill veta skulden nästa månad utan om, exempelvis, 12 månader. Istället för att köra funktionen 12 gånger med olika inargument kan vi använda en så kallad loop. Vi inför också en ny inparameter: antalet månader.

```
function ny_skuld = laneberakning(skuld,lon,ranta,...
    betalningsgrad,manader)
% ny_skuld = laneberakning(skuld,lon,ranta,...
%      betalningsgrad,manader)
% Beräknar skulden ett antal månader framåt.
% Mata in skuld, lön, ränta och betalningsgrad per
% månad samt antal månader.

% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end

% Beräkna den nya skulden:
for t = 1:manader
    skuld = skuld + skuld*ranta - betalningsgrad*lon;
end
ny_skuld = skuld;

% Testa om skulden är avbetald:
if ny_skuld <= 0
    ny_skuld = 0;
    disp('Skulden är avbetald!')
else
    disp('Skulden är inte avbetald ännu.')
end
```

Nyckelorden `for` och `end` avgränsar upprepningsstrukturen. Variablen `t` stegas igenom det som står efter = och för varje värde körs programraderna mellan `for` och `end`. Kolon-notationen, `:`, känner vi igen från datorlektion 1. Variabeln `t` kommer alltså vara 1 första gången programraden (eller raderna) körs, 2 andra gången osv. Till slut kommer värdet upp till `manader` (t.ex. 12). Varje gång beräknas ett nytt värde på variabeln `skuld`. Prova att ta bort semikolonet efter beräkningen för att se hur delresultaten skrivs ut.

I ovanstående exempel används `t` bara för att hålla räkningen. Man kan också använda den inne i loopen. I exemplet kan man vilja få ut skulden månad för månad istället för bara den sista månaden. Låt därför `skuld` bli en vektor, där `skuld(t)` anger skulden i slutet på månad `t`. Initiera vektorn före for-loopen med

```
skuld = [skuld; zeros(manader,1)];
```

Detta skapar en vektor där vi kan spara skulden månad för månad. Kommandot inuti for-loopen ändrar vi till

```
skuld(t+1) = skuld(t) + skuld(t)*ranta - betalningsgrad*lon;
```

I övrigt kan funktionen vara likadan.

Testa den här varianten av funktionen och rita en kurva över hur skulden ändras, månad för månad. Skriv ut varje iteration så att du ser hur vektorn `skuld` växer. Detta är också användbart vid felsökning av koden!

Uppgift 3. Utöka och modifiera funktionen `kurvplot`, så att den ritar kurvorna $f_2(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin(x\sqrt{1-t^2} + \arccos t)$ för olika värden på t i intervallet $0 \leq t < 1$. Rita de nya kurvorna i samma figur. Du kan låta t variera mellan 0.1 och 0.9 med steglängden 0.2. Du skall alltså använda en `for`-loop för hantera olika `t` inne i funktionen.

while-loopar

Man vet inte alltid på förhand hur många gånger man vill upprepa en beräkning. Emma vill kanske veta efter hur många månader som lånet är avbetalt. Då kan man använda en `while`-loop. `while` kontrollerar ett villkor (på samma sätt som `if`) och utför vissa programrader om det är uppfyllt. Därefter kontrolleras villkoret igen, och programraderna utförs igen om villkoret är uppfyllt. När villkoret inte är uppfyllt längre avslutas upprepningen. Med hjälp av `while` kan vi modifiera exemplet för att räkna ut vilken månad som lånet är avbetalt:

```
function slutmanad = laneberakning(skuld,lon,ranta,...
    betalningsgrad)
% slutmanad = laneberakning(skuld,lon,ranta,...
%     betalningsgrad)
% Beräknar vilken månad som skulden är avbetald.
% Mata in skuld, lön, ränta och betalningsgrad per månad.

% Testa om den inmatade räntan är positiv:
if ranta<0
    error('Räntan måste vara positiv!')
end

manad_nr = 1;
while skuld(manad_nr)>0
    % Beräkna den nya skulden:
    skuld(manad_nr+1) = skuld(manad_nr) + ...
        skuld(manad_nr)*ranta - betalningsgrad*lon;
    manad_nr = manad_nr+1;
end
slutmanad = manad_nr;
```

När man använder en `while`-loop är det viktigt att se till att villkoret för upprepning inte alltid är uppfyllt. Annars kan funktionen fastna i en oändlig loop[1]. Detta bör man förstås kontrollera innan man går in i loopen. I fallet ovan uppstår en oändlig loop om skulden inte minskar för varje månad. Detta kan vi kontrollera på det sätt som vi beskrev tidigare: kontrollera att månadsbetalningen `betalningsgrad*lon` är större än den månatliga ökningen av skulden `skuld*ranta`. Skriv koden så att while-loopen bara startar ifall villkoret är uppfyllt (dvs. det finns ingen risk för en oändlig loop), annars ska ett felmeddelande visas.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Förutom villkoret ovan kan man också sätta en maxgräns på antalet iterationer, dvs. modifiera villkoret i `while`-satsen så att `manad_nr` inte får vara större än t.ex. 200. Man kan också tänka sig att införa denna maxgräns som ytterligare ett inargument till funktionen.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Summera eller ge ett exempel på när `for`-loopar ska användas och när det är bättre att använda `while`-loopar:

. . . . . .

. . . . . .

[1] Skulle man hamna i en oändlig loop kan man avbryta beräkningarna genom att trycka Ctrl-c.

. . . . . .

Uppgift 4. Maxvärdet hos funktionen

$$f(x) = 1 - \frac{e^{-tx}}{\sqrt{1-t^2}} \sin\left(x\sqrt{1-t^2} + \arccos(t)\right)$$

beror på vilket värde som $t \geq 0$ har. Anta att vi bara är intresserade av positiva x-värden. Vi vill göra t så litet som möjligt, men ändå ha ett maxvärde som är mindre än t.ex. 1,2.

Vi kan prova olika t-värden i en `while`-loop. Modifiera funktionen så att den tar fram det minsta t-värdet som ger ett visst maxvärde. Tänk på att undvika oändliga loopar.

Ledning: Observera att maxvärdet av $f(x)$ blir mindre ju större t är. Om du börjar med $t = 0$ och stegvis ökar t så kommer maxvärdet förr eller senare att bli mindre än 1,2.

Uppgift 5. I lånberäkningsfunktionen ovan räknar vi förutom slutmånaden ut skulden månad för månad. Eftersom funktionsvariabler är lokala så kan vi inte komma åt de från Workspace när funktionen avslutat sin körning. Vill vi ha åtkomst till skuldutvecklingen måste vi ange två utargument:

```
function [slutmanad,skuldutveckling] = laneberakning(...
    skuld,lon,ranta,betalningsgrad)
```

Vi måste då se till att variabeln `skuldutveckling` tilldelas ett värde någonstans i funktionen. Funktionen kan sedan anropas med sina två utargument:

```
>> [m, s] = laneberakning(100000,25000,0.01,0.1)
```

Modifiera funktionen så att den producerar två utargument.

4 Simulering av ett enkelt ingenjörsproblem

Nedanstående exempel är hämtat från en kurs i linjär algebra. Det är alltså inte primärt avsett att lösas i MATLAB, men varför inte tillämpa våra MATLAB-kunskaper på det?

> En lokal biluthyrningsfirma har två kontor, ett i Linköping och ett i Norrköping. Det har visat sig att under en månad lämnas 70% av bilarna som hyrts i Linköping tillbaka dit, medan 30% av dem lämnas tillbaka i Norrköping. För Norrköpingskontoret gäller att 40% av deras uthyrda bilar återlämnas där, medan 60% lämnas i Linköping. Företaget har 15 bilar. Varje bil hyrs ut en gång per månad. Hur många ska placeras i Linköping respektive Norrköping? Vad gäller om man köper in fler bilar eller om procenttalen ändras?

Problemet har en analytisk lösning, men vi ska lösa det genom simulering. På detta sätt kan vi se hur fördelningen av bilar ändras över tid och når sin analytiska lösning.

Uppgift 6. (Tänk på att dokumentera och kommentera din kod efter hand)

1. Börja med t.ex. 1 bil i Linköping och 14 i Norrköping vid månad 1. Vad händer under månad 2? 3? 4? Skriv ett skript som simulerar ett antal månader. Kolla att resultatet stämmer. Plotta kurvor över utvecklingen.

2. Prova skriptet för olika många bilar och olika många månader.

3. Skriv om skriptet till en funktion. Inargument: en vektor med antal bilar i varje stad (vid början på första månaden) och antal månader som ska simuleras. Utargument: En vektor/matris med resultatet och en prydlig figur med beskrivande text.

4. Generalisera funktionen så man även kan mata in procentsatser för återlämnandet av bilar. Kontrollera att procentsatserna summerar upp till 100, annars ska det skrivas ut ett felmeddelande.

5. Lägg till en tredje stad: Nyköping. Finns det något sätt att ändra koden så att det är lätt att lägga till fler städer? Kan antalet städer vara en inparameter?

6. Ska man vara riktigt noga är det bara heltalslösningar som är intressanta (man lär ju inte lämna igen en halv bil i Linköping och andra halvan i Norrköping). Modifiera koden så att antalet bilar i varje stad är ett heltal i slutet på varje månad, t.ex. med hjälp av funktionen `round`. Det finns en risk att det totala antalet bilar ökar eller minskar vid avrundningen (särskilt när ni har tre städer), så ni bör använda kontrollfunktioner för att identifiera när det inträffar och ordna problemet.

Uppdelningen i punkter illustrerar ett vanligt angreppssätt på den typ av problem som man ofta löser i MATLAB. Man börjar med att lösa ett mindre delproblem, där man lätt kan kontrollera svaret. Visualisera svaret med lämpliga figurer, som dokumenteras med lämplig text. Om hela problemet inte är löst fortsätter man med att lösa ett annat delproblem, som sedan kombineras med det man redan gjort. Man fortsätter på detta sätt tills hela problem är löst och resultatet har visualiserats.

Detta upplägg liknar en av de vanligaste programmeringsmetoderna: stegvis förfining. Stegvis förfining handlar om att dela upp sitt problem i delproblem, som kan lösas i tur och ordning. Förhoppningsvis är vart och ett av delproblemen lättare att lösas. Antingen är de så små att man vet direkt hur de ska lösas, eller också delar man upp dem i sin tur.

Slutligen ska du öppna dina skript och funktioner i MATLABs editor och titta efter den färgade fyrkanten i övre högra hörnet. Vilken färg har den?

- Grönt: Inga varningar eller fel hittades.
- Gul: Varningar hittades.
- Rött: Fel och varningar hittades

Rött ljus betyder vanligtvis att det finns syntax-fel i koden, så att den inte går att köra. Grönt ljus innebär att du gjort ett bra jobb! I många fall kommer du först se ett gult ljus, vilket betyder att MATLAB har rekommendationer för hur du kan skriva om koden för att göra den mer effektiv och få bättre noggrannhet i beräkningarna. Varje rekommendation är markerad i skrollistan på höger sida i editorn. Gå igenom alla rekommendationer och försöka göra de föreslagna ändringarna. Förhoppningsvis får du till slut ett grönt ljus!

5 For-loopar eller matris/vektor-operationer

Eftersom MATLAB ursprungligen skapades för matrisberäkningar så är detta något som MATLAB utför snabbt och effektivt. For-loopar har traditionellt sett varit långsamma i MATLAB, även om det förbättrats på senare år. Det finns många fall då matris/vektor-operationer kan ersätta for-loopar.

Skapa två stora matriser A och B:

```
>> A = rand(1000,1000);
>> B = rand(1000,1000);
```

Jämför tiden det tar att utföra matrisadditionen $A + B$ som en matrisoperation

```
>> C = A + B;
```

med tiden det tar med hjälp av for-loopar:

```
>> for row = 1:1000
>>     for col = 1:1000
>>         D(row,col) = A(row,col) + B(row,col);
>>     end
>> end
```

Vilken av dessa implementationer är snabbast?

.....

Du kan beräkna tiden som det tar att köra din kod genom att placera `tic` innan koden och avsluta med `toc`. Detta mäter tiden som det tar att komma från `tic` till `toc`. Vilka tidsvärden får du för de två kodsnuttarna ovan? (Glöm inte att tömma workspace mellan operationerna).

.....

.....

Ifall du måste gå igenom elementen i en matris med hjälp av for-loopar så är det bra att först allokera minne. Det innebär att placera raden

```
>> D = zeros(1000,1000);
```

före for-loopen så att MATLAB vet hur stort D ska bli och inte behöver ändra dimensionerna i varje iteration av for-loopen.

Hur påverkar detta körtiden för koden?

.....

6 Hemuppgifter: Vecka 1-2

Ett antal hemuppgifter var givna i slutet av materialet till datorlektion 1. Se till att slutföra dessa uppgifter!