Kontrollstrukturer i MATLAB

1 Inledning

Redan vid första laborationstillfället gjorde ni ett litet program. Ni skrev ett script som beräknade summan $s = 1^2 + 2^2 + 3^2 + 4^2 + 5^2$ med en programkod som kanske såg ut så här

```
s=0;
for i=1:5
    s=s+i^2;
end
s
```

Nu skall vi lära oss lite mer om kontrollstrukturer som `if`-, `for`- och `while`-satser med vilka vi styr flödet genom programkoden.

2 Logiska uttryck och operationer

Vi kommer behöva använda logiska villkor av typen $x > 5$. Detta uttryck är sant om ett tal vi betecknar med x är strängt större än talet 5, annars falskt. Vi skriver detta uttryck i MATLAB som `x>5` helt enkelt. Det logiska värdet sant beskrivs i MATLAB av talet 1 och falskt beskrivs av talet 0.

Relationsoperatorerna $<$, $\leq$, $>$, $\geq$, $=$ och $\neq$ skrivs i MATLAB med `<`, `<=`, `>`, `>=`, `==` respektive `~=`. Observera dubbla likhetstecken i MATLAB för att beteckna (logisk) likhet, enkelt likhetstecken används ju för tilldelning, dvs. att ge en variabel ett värde. Vidare har vi ibland nytta av funktionerna `any` och `all` som arbetar på radvektorer (se gärna hjälptexten i MATLAB).

De logiska operatorerna ”och”, ”eller” samt ”negation” skrivs i MATLAB med `&`, `|` respektive `~`.

3 Villkorssatser

Det allmänna utseendet på en `if`-sats är någon av följande alternativ

```
if uttryck
    satser
end
```

```
if uttryck
    satser
else
    satser
end
```

```
if uttryck
    satser
elseif uttryck
    satser
end
```

```
if uttryck
    satser
elseif uttryck
    satser
else
    satser
end
```

där vi kan ha godtyckligt många `eleseif` i de två sista alternativen. Med `uttryck` avser vi ett logiskt uttryck av den typen vi nämnde ovan.

Exempel 1. Vi har två värden a och b och vill att c skall ges det största av dessa värden. Detta kan göras med följade kod

```
if a<b
    c=b
else
    c=a
end
```

Om $a < b$ är sant, så är b störst och c ges värdet av b, annars är a störst och c ges värdet av a.

4 Repetitionssatser

För att upprepa en grupp av satser flera gånger används `for`-satser eller `while`-satser. Vet vi på förhand hur många gånger upprepningen skall ske, så är normalt en `for`-sats att föredra i annat fall är en `while`-sats lämpligare.

4.1 `for`-satser

Det allmänna utseendet på en `for`-sats är

```
for variabel = uttryck
    satser
end
```

Här är `uttryck` en radvektor av tal eller ett uttryck som bygger upp en sådan radvektor. Successivt kommer `variabel` tilldelas värdena i `uttryck` i tur och ordning och samtidigt kommer alla `satser` ned till `end` att utföras. En gång för varje värde som `variabel` ges.

Allra vanligast är följande enkla variant

```
for variabel = start:steg:slut
    satser
end
```

Vi har redan sett på några summor som vi beräknat med `for`-sats och här kommer några ytterligare exempel.

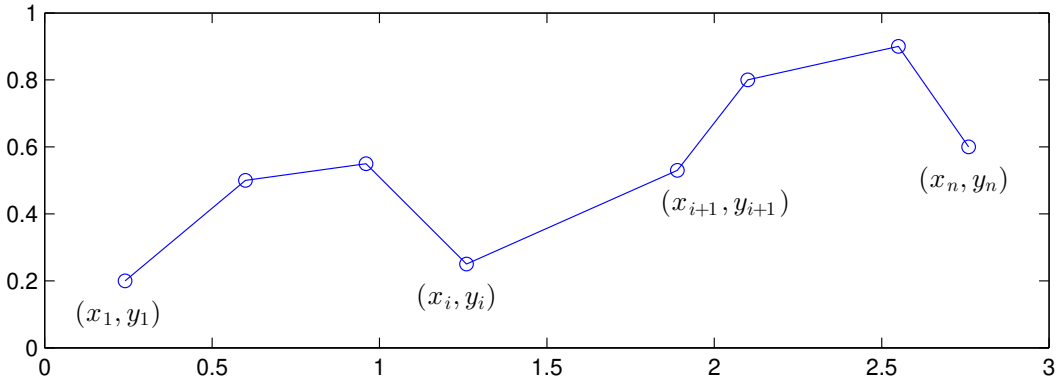
Exempel 2. Vi kan beräkna summan $s = 1 + 3 + 5 + 7 + 9 + 11 + 13$ med en programkod som ser ut så här

```
s=0;
for i=1:2:13
    s=s+i;
end
s
```

Vi kan se det som att vi har en låda `s` som vi samlar värden (termer) i. Först ser vi till att lådan från början är tom med `s=0`.

När `for`-satsen utförs kommer `i` successivt få värdena $1, 3, 5, \cdots, 13$. För varje värde som `i` får kommer `s=s+i` utföras, dvs. vi kommer lägga aktuellt värde på `i` till det vi redan har i lådan `s`.

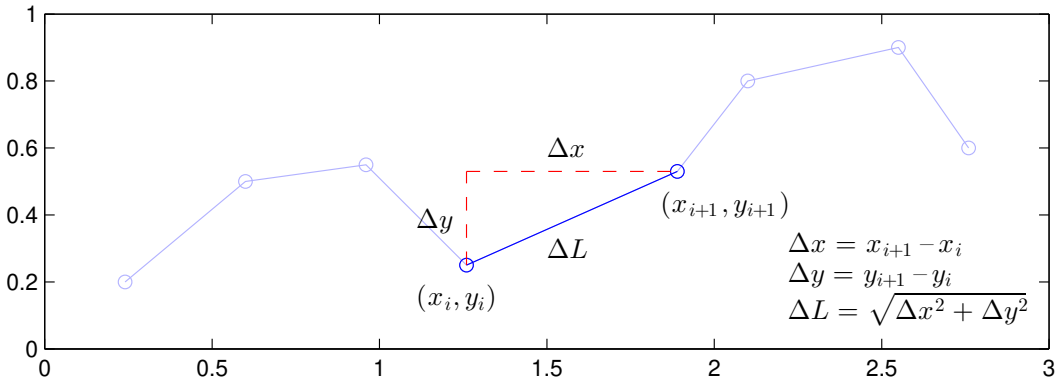
Exempel 3. Vi tänker oss att vi har ett polygontåg $(x_1, y_1), (x_2, y_2) \cdots, (x_n, y_n)$ som vi ritat en figur av



Vill vi beräkna polygontågets längd kan vi göra det med

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} + \cdots$$
$$+\sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2}$$

Denna formel fås genom att använda Pytagoras sats på varje segment i polygontåget.



Antag att koordinaterna samlade i två radvektorer $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ och $\mathbf{y} = (y_1, y_2, \cdots, y_n)$, då beräknar vi längden enligt

```
>> n=length(x);
>> L=0;
>> for i=1:n-1
       L=L+sqrt((x(i+1)-x(i))^2+(y(i+1)-y(i))^2);
   end
>> L
```

Uppgift 1. Skriv en function med namnet `polylen`, för beräkning av längden av ett polygontåg enligt exemplet ovan. Funktionen skall ha följande struktur (repetera laboration 2, avsnitt 4)

```
function L = polylen(x,y)
    satser
```

Pröva din funktion på en triangel och en rektangel, dvs. skapa två radvektorer `x` och `y` med koordinater för t.ex. en triangel och beräkna längden av polygontåget (i detta fall omkretsen) med `Omkrets=polylen(x,y)`.

4.2 `while`-satser

En `while`-sats tillåter en grupp av satser att bli repeterade under kontroll av ett logiskt villkor enligt

```
while uttryck
    satser
end
```

Uttrycket i while-satsen är ett logiskt uttryck. Satserna repeteras så länge det logiska uttrycket är sant.

Exempel 4. Man kan beräkna $\sqrt{c}$ med upprepade additioner och divisioner med iterationsformeln

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{c}{x_k}\right),\ k = 0, 1, 2, \cdots$$

där $x_0 = c$. Iterationen avbryts då $d = |x_{k+1} - x_k| < \text{tol}$, där tol är måttet på önskad noggrannhet i approximationen.

Vi beräknar en approximation av t.ex. $\sqrt{2}$ med noggrannheten $\text{tol} = 10^{-16}$. Först ser vi på några steg i iterationen

$$\begin{array}{ll} x_0 = c & x_0 = 2 \\ x_1 = \frac{1}{2}(x_0 + \frac{c}{x_0}) & x_1 = \frac{1}{2}(2 + \frac{2}{2}) = 1.5 \\ x_2 = \frac{1}{2}(x_1 + \frac{c}{x_1}) & x_2 = \frac{1}{2}(1.5 + \frac{2}{1.5}) = 1.4166\cdots \\ x_3 = \frac{1}{2}(x_2 + \frac{c}{x_2}) & x_3 = \frac{1}{2}(1.4166\cdots + \frac{2}{1.4166\cdots}) = 1.4142\cdots \\ \quad\vdots & \quad\vdots \end{array}$$

Sedan använder vi MATLAB enligt

```
>> c=2;
>> tol=1e-16;
>> x=c;
>> d=1;
>> while d>tol          % så länge d>tol görs följande
       xny=(x+c/x)/2;
       d=abs(xny-x);
       x=xny;
>> end                  % slutet på while-satsen
>> x
x =
    1.4142
```

Vi kan gör en skriptfil av koden ovan, alternativt kan vi göra det som en function, (`min_sqrt.m`)

```
function x=min_sqrt(c)
    tol=1e-16;
    x=c; d=1;
    while d>tol
        xny=(x+c/x)/2;
        d=abs(xny-x);
        x=xny;
    end
```

Vi använder den så här (för att beräkna $\sqrt{5}$)

```
>> x=min_sqrt(5)
x =
    2.2361
```

Uppgift 2. Det gäller att

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$$

Hur många termer måste man ta med i summan för att approximera π med fem korrekta decimaler?

Termerna i summan kan skrivas $\frac{(-1)^i}{2i+1}$ för $i = 0, 1, \cdots$.

(a). Bilda successivt delsummor

$$s_n = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^n}{2n+1}$$

för allt större heltal n och avbryt när s_n tillräckligt nära $\frac{\pi}{4}$. Använd en `while`-sats.

(b). Använd sedan en `for`-sats och beräkna $\frac{\pi}{4}$ med 1000 termer.

5 Utskriftsformat

Utskriften av beräkningsresultatet i MATLAB görs som standard med fem siffror. Vill vi få fler siffror utskrivna kan vi ge kommandot `format long` innan utskriften. Med `format short` får vi tillbaka den korta varianten.

```
>> format short
>> a=pi
a =
   3.1416

>> format long
>> a
a =
   3.141592653589793
```

Med `format short e` respektive `format long e` får vi s.k. scientific notation.

```
>> format short
>> b=exp(-5)
b =
    0.0067

>> format short e
>> b
b =
   6.7379e-03
```

```
>> format long e
>> b
b =
     6.737946999085467e-03
```

Talet vi beräknar med `exp(-5)` är e^{-5}. Exponentialfunktionen heter alltså `exp` i MATLAB. Svaret `6.7379e-03` betyder 6.7379×10^{-3}, dvs. här står `e` för exponent (basen 10). MATLAB räknar alltid med samma antal siffror (ungefär 16 decimala siffror), med `format` påverkar vi bara vad som skrivs ut.

6 In- och utmatning

Avslutningsvis ser vi lite på in- och utmatning. Ibland kan man t.ex. vilja ha inmatning till ett script medan det utförs (körs) eller att beräkningsresultat skall skrivas ut med ett visst format, t.ex. ett visst antal decimaler.

Med `input` kan vi mata in ett värde enligt

```
>> antal=input('Ange antal kast: ');
Ange antal kast: 5
```

När MATLAB kommer till `input`-kommandot så skrivs texten `'Ange antal kast:'` ut och programmet väntar på raden tills vi skriver ett svar, ivårt fall 5, variabeln `antal` ges detta värde och programmet fortsätter sedan med nästa kommando. Semikolonet (`;`) efter `input` gör att vi inte får någon utskrift då `antal` får sitt värde, 5:an vi ser är den vi skrev. (Variabeln får givetvis ha vilket namn som helst, vi valde namnet `antal` i exemplet.)

Vill vi mata in en textsträng med `input` lägger vi till ett `'s'` och vill vi få en ny rand för vårt svar får vi det med `\n` enligt

```
>> svar=input('Hej, hur mår du?\n','s');
Hej, hur mår du?
Bra
```

Om vi inte använder semikolon (`;`) efter t.ex. en tilldelning så skrivs variabelnamnet ut tillsammans med det värde variabeln fått. Vill vi bara skriva ut värdet av variabeln kan vi använda `disp`. Vi testar på variabeln `svar`

```
>> disp(svar)
Bra
```

Med `sprintf` och `fprintf` kan vi skriva formatterad text. Vill vi ha utskriften till en textsträng använder vi `sprintf` och vill vi ha den till en textfil använder vi `fprintf`. Som exempel skriver vi ut π med 7 decimaler i Command Window enligt

```
>> disp(sprintf('Pi =%10.7f',pi))
Pi = 3.1415927
```

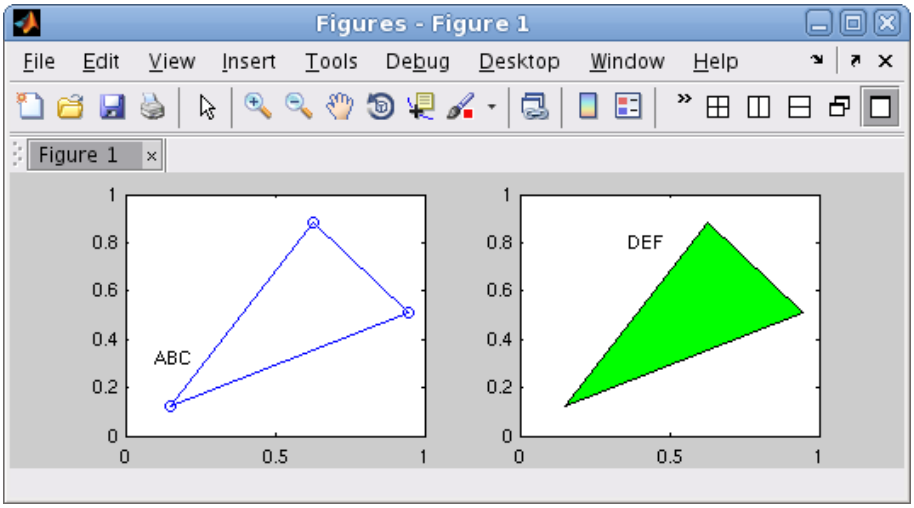
Formatkoderna ser ut som i programspråken C och Java, men det är inget vi behöver lära oss nu. Vid behov söker man i hjälptexterna.

Om man vill läsa in koordinater med musen kan man använda `ginput`. Vi kan placera ut text i ett koordinatsystem med kommandona `text` och `gtext`. Med `text` får vi ge koordinater för var texten skall placeras och med `gtext` använder du musen för att peka på önskad plats. Vi ser på detta med hjälp av ett exempel.

Exempel 5. I förra laborationen ritade vi en triangel genom att ge koordinaterna med siffervärden. Ibland är det smidigare att använda `ginput`, vi pekar på punkter i koordinatsystemet och trycker på en musknapp.

```
>> clf
>> subplot(1,2,1)
>> [x,y]=ginput(3);              % Triangelns 3 hörn, x och y blir radvektorer
>> x=[x, x(1)];                  % Vi sluter polygontåget, så alla sidor ritas
>> y=[y, y(1)];
>> plot(x,y,'-o')
>> axis([0 1 0 1])
>> text(0.1,0.3,'ABC')           % Koordinatplacerad text på grafen

>> subplot(1,2,2)
>> fill(x,y,'g')
>> axis([0 1 0 1])
>> gtext('DEF')                  % Med markören placerad text på grafen
```



Att vi sluter polygontåget innan vi ritar upp beror på att vi vill att triangelns alla sidor skall ritas, men vi vill inte behöva försöka markera det första hörnet en gång till.

Med `[x, x(1)]` bildar vi en radvektor med alla ursprungliga x-koordinater (`x`) först och till dessa lägger vi första x-koordinaten (`x(1)`) en gång till (vi sluter). Hakparanterna (`[ ]`) används för att bygga upp den nya (större) radvektorn. Sedan får `x` detta värde. På motsvarande sätt gör vi sedan i y-led.

Uppgift 3. Mata in punkter med `ginput` så att du får ett polygontåg som du sedan sluter. Du skall inte bestämma antal punkter i förväg, men det skall vara fler punkter än i en triangel. Färglägg sedan området som innesluts med någon färg. Gå till Help och läs i hjälptexten för `ginput` så du kan ta reda på hur man kan läsa in koordinater för flera punkter, utan att i förväg behöva bestämma hur många.