Kontrollstrukturer och funktioner i MATLAB

1 Inledning

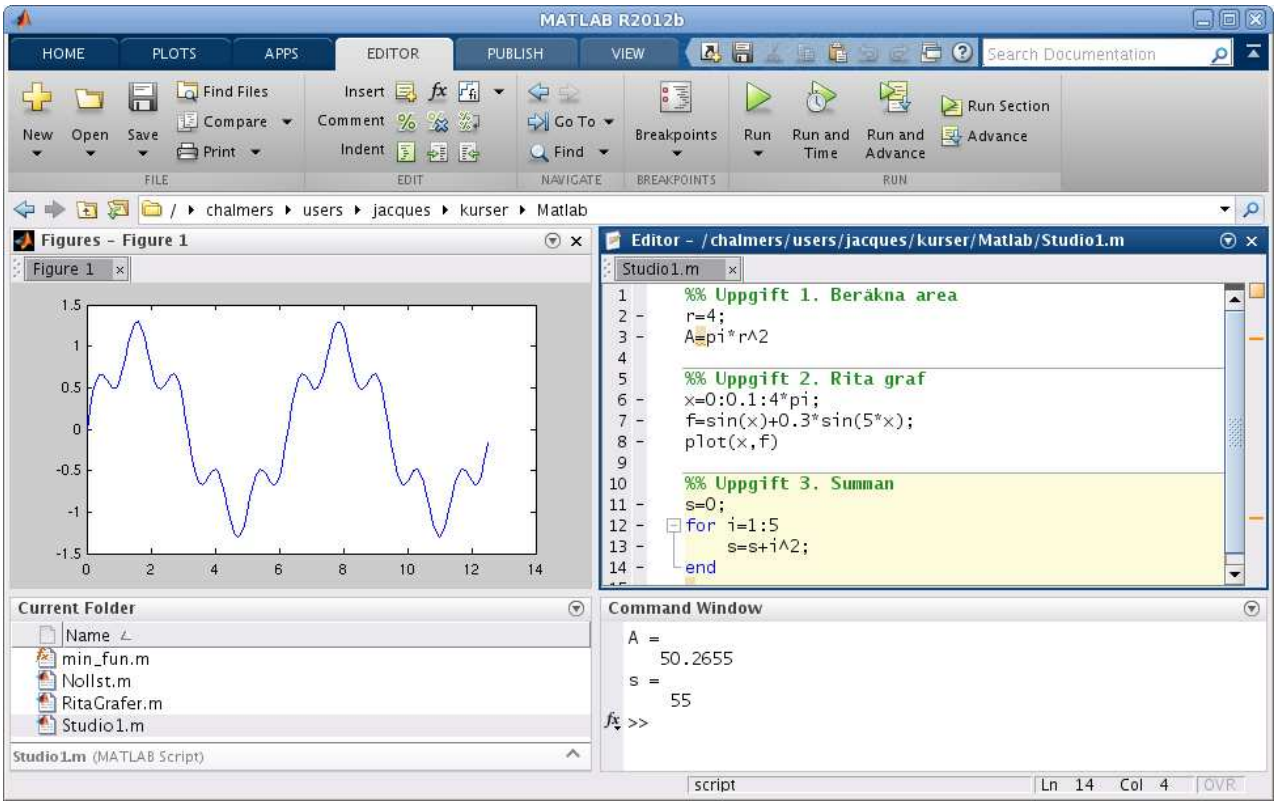
Vid första laborationen gjorde ni ett litet program. Ni skrev ett script som beräknade summan $s = 1^2 + 2^2 + 3^2 + 4^2 + 5^2$ med en programkod som kanske såg ut så här

```
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. Vi skall också se lite mer på funktioner.

Men allra först: För att arbeta på ett överskådligt och effektivt sätt, dokumentera arbetet och redovisa för handledare snabbt och smidigt skall ni använda er av script.

Så här kunde det sett ut efter det att vi löst de tre första uppgifterna i första laborationen.



Editorn i MATLAB har använts i Cell Mode (cell-läge). Skriver man en kommentar som börjar med två procent-tecken, så avgränsar det en cell. Poängen är att man kan låta MATLAB utföra kommandona från en cell, istället för hela filen.

På så sätt kan man dela upp ett stort script (för en hel laboration) i flera delar (varje deluppgift). En cell kan utföras utan att textfilen är sparad, så gör Save då och då. Har du glömt bort script och cell-läge repetera i så fall laboration 1, avsnitt 4.

Vi har också gjort en egen desktop layout så att Figures (figurfönster), Editor samt Command Window syns samtidigt. Läs i texten för laboration 1, avsnitt 7 hur man gör.

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 vektorer (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 `elseif` 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.

Uppgift 1. Skriv en villkorssats som ger c det minsta av värdena på a och b.

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 vektor av tal eller ett uttryck som bygger upp en sådan vektor. 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
```

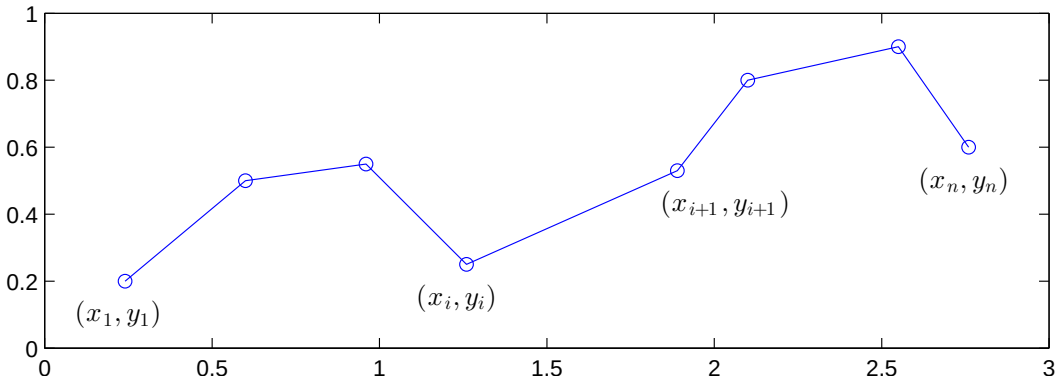
Vi har redan sett på några summor som vi beräknat med `for`-sats och här kommer några ytterligare exempel.

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
```

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 `s`. Vi får inte glömma att sätta `s=0` innan `for`-satsen.

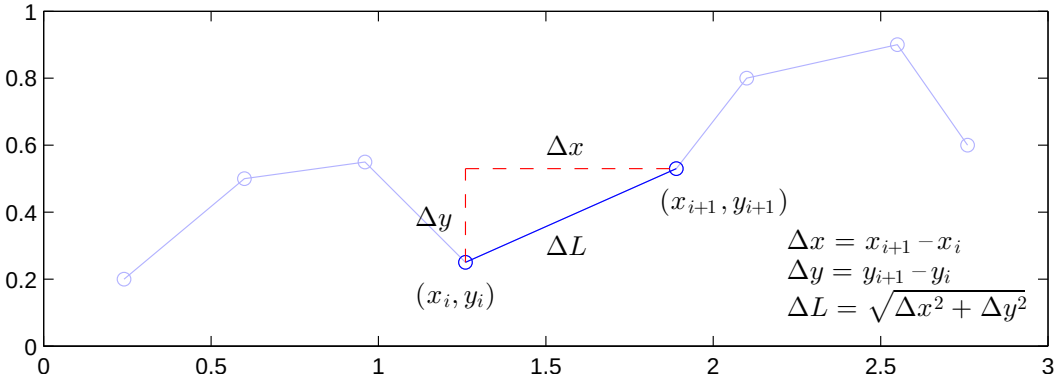
Exempel 3. Vi tänker oss att vi har ett polygontåg $(x_1, y_1), (x_2, y_2) \cdots, (x_n, y_n)$ som vi ritat en figur av



Vill vi beräkna polygontågets längd kan vi göra det med

$$L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

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



Antag att koordinaterna samlade i två vektorer $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ 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
```

Nu skall vi se på några exempel där vi på förhand inte vet hur många gånger repetitionen skall utföras.

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 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
```

Repetitionen avbryts när vi har tillräcklig noggrannhet. Vill vi veta hur många repetitioner vi gjorde kan vi t.ex. lägga till satsen `k=0` före `while`-satsen, en uppräkning `k=k+1` inne i `while`-satsen och en utskrift av `k` efter `while`-satsen.

Vill vi att fler decimaler skrivs ut, får vi det med `format` enligt

```
>> format long
>> x
x =
   1.414213562373095
```

Mer om utskriftsformat i ett senare avsnitt.

Uppgift 2. Det gäller att

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$$

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

(a). Bilda successivt delsummor

$$s_n = \sum_{i=0}^{n} \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^n}{2n+1}$$

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 Funktioner

En funktion (function) i MATLAB är en textfil med följande struktur

```
function ut = funktionsnamn(parametrar)
    satser
```

Här är `funktionsnamn` det namn vi ger funktionen och `funktionsnamn.m` är det namn vi ger textfilen där programkoden lagras. Med `parametrar` avser vi indata till funktionen, ofta en variabel ibland flera. Funktionen måste innehålla en sats där `ut`, som står för utdata eller funktionsvärdet, tilldelas ett värde.

Exempel 5. Gör en funktion som beräknar medianen av värdena i en vektor. Medianen är det mittersta värdet i storleksordning om vektorn har udda antal element, och medelvärdet av de två mittersta (i storleksordning) om antal element är jämnt.

Längden av en vektor (antal element) ges av `length` och elementen i en vektor sorteras i storleksordning av `sort`. Dessa inbyggda funktioner kommer vi se mer på nästa laboration.

Funktionen `rem` ger resten vid heltalsdivision, t.ex. får `rem(n,2)` värdet 0 om `n` är att jämnt heltal och värdet 1 om `n` är ett udda heltal. En av många inbyggda elementära funktioner. Läs gärna i hjälptexten om `rem`. Se laboration 1, avsnitt 8 hur man söker i hjälptexterna.

Nu skriver vi vår funktion

```
function m=min_median(v)
%   m = min_median(v) beräknar medianen av elementen i vektorn v
%
    s=sort(v);                    % s sorterad version av v
    n=length(v);                  % n antal element i v
    if rem(n,2)==0                % n jämnt
        m=(s(n/2)+s(n/2+1))/2;
    else                          % n udda
        m=s((n+1)/2);
    end
```

som vi lagrar under namnet `min_median.m` och ser hjälptexten med

```
>> help min_median
  m = min_median(v) beräknar medianen av elementen i vektorn v
```

och tar medianen av en slumptalsvektor (`rand` ger slumptal mellan 0 och 1)

```
>> v=rand(1,6)
v =
    0.4103    0.8936    0.0579    0.3529    0.8132    0.0099

>> m=min_median(v)
m =
    0.3816
```

Nu har MATLAB en inbyggd funktion `median` för att bilda medianen som vi givetvis använder istället. Titta på hjälptexten för `median` för att se en ”riktig” hjälptext.

En anonym funktion (anonymous function) med ett funktionshandtag (function handle) enligt

```
handtagsnamn = @(parametrar) sats
```

Här är delen `@(parametrar) sats` den anonyma funktionen och `handtagsnamn` är det namn vi väljer på funktionshandtaget som kopplas till funktionen. Med `parametrar` avser vi indata till funktionen, ofta en variabel men ibland flera. (Denna konstruktion är det bara tillåtet med en enda beräkningssats. En mer komplicerad funktion kräver att vi definierar en function.)

Exempel 6. Vi ritar grafen till $f(x) = \cos(x) - \mathrm{e}^{-x^2/2}\sin(7x)$ över intervallet $-2\pi \leq x \leq 2\pi$.

```
>> f=@(x)cos(x)-exp(-0.5*x.^2).*sin(7*x);
>> x=linspace(-2*pi,2*pi);
>> plot(x,f(x))
```

Skriv gärna in i Command Window så du får se grafen.

Uppgift 3. Skriv en function med namnet `polylen`, för beräkning av längden av ett polygontåg enligt exempel 3. Pröva din funktion på en triangel och en rektangel, dvs. skapa två vektorer `x` och `y` med koordinater för t.ex. en triangel och beräkna längden av polygontåget (i detta fall omkretsen) med `Omkrets=polylen(x,y)`.

6 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.

7 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
```

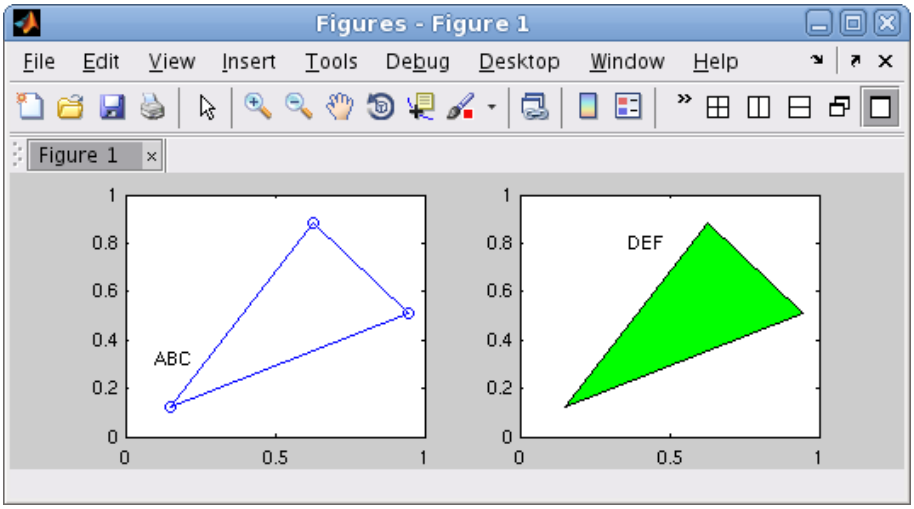
Formatkoderna ser ut som i programspråken C och Java, men det är inget vi behöver lära oss nu. Vid behov söker man i hjälptexterna.

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 7. Vi ritade en triangel genom att ge koordinaterna med siffervärden i laboration 1. Ibland är det smidigare att använda `ginput`, vi pekar på punkter i koordinatsystemet och trycker på en musknapp.

```
>> subplot(1,2,1)
>> [x,y]=ginput(3);              % Triangelns 3 hörn, x och y blir kolonner
>> 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
```



Med `[x; x(1)]` bildar vi en vektor 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) vektorn. Sedan får `x` detta värde. På motsvarande sätt gör vi sedan i y-led.

Uppgift 4. 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.

1 Målsättning

Avsikten med laborationen är att lära sig kontrollstrukturer i MATLAB som `if`-, `for`- och `while`-satser med vilka vi styr flödet genom programkoden. Vidare är avsikten att lära sig skriva enklare program i form av script eller function.

2 Kommentarer och förklaringar

Med kontrollstrukturer styr vi flödet genom programkoden. Med `if`-satser gör vi val och med `for`- och `while`-satser gör vi upprepningar.

Med script och function systematiser vi och strukturerar arbetet. När vi löser beräkningsproblem gör vi oftast en problembeskrivning med function.

I denna laboration finns det kanske inte så mycket att säga om avsnitten, det gäller att repetera många gånger.

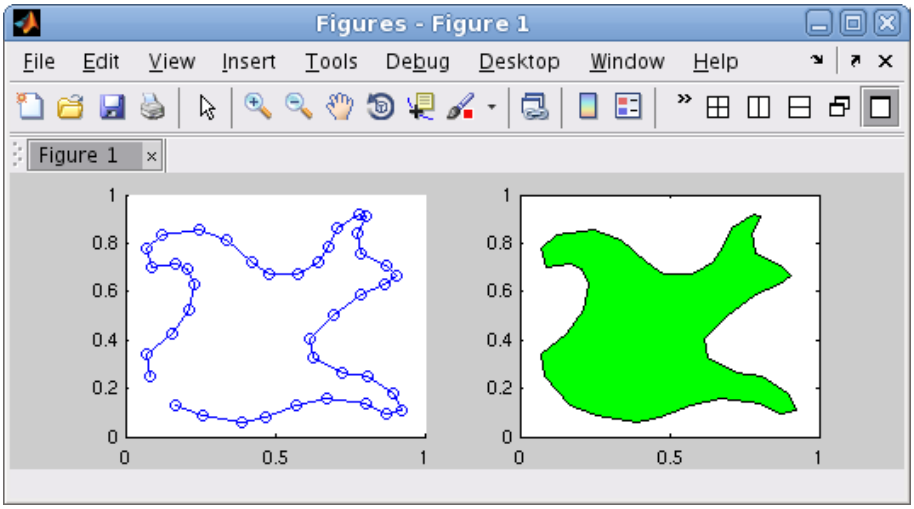
3 Fördjupning

Vi vill rita ett polygontåg genom att markera punkter med `ginput` och succesivt rita upp linjer som förbinder på varandra följande punkter. Vi håller på att markera nya punkter och rita linjer så länge vi trycker på vänster musknapp, annars avbryter vi. Sedan fyller vi området med grön färg. Vi gör lämpligen ett script.

```
clf
subplot(1,2,1)
axis([0 1 0 1]), hold on      % Skapar ritområde
[x,y]=ginput(1);              % Första punkten läses in ...
plot(x,y,'o')                 % ... och ritas ut
while 1                       % För evigt,1 står ju för sant
    [xi,yi,knapp]=ginput(1);  % Ny punkt, knapp är den musknapp som trycktes på
    if knapp~=1               % Om knapp inte vänster musknapp ...
        break                 % ... så avbryts whilesatsen
    end
    x=[x; xi]; y=[y; yi];     % Nya punktens koordinater till tidigare
    plot(x(end-1:end),y(end-1:end),'o-')
end                           % Rita från näst sista punkten till den sista
hold off
x=[x; x(1)]; y=[y; y(1)];     % Vi sluter polygontåget, så alla sidor ritas

subplot(1,2,2)
fill(x,y,'g')
axis([0 1 0 1])
```

Här ser vi resultatet



Läs i hjälptexten för `ginput` så att du förstår hur `[xi,yi,knapp]=ginput(1)` fungerar.

Lägg märke till att vi sparar alla koordinater så att vi kan göra annat med polygontåget, t.ex. beräkna dess längd.

4 Lärandemål

Efter denna laboration skall du i MATLAB

- kunna använda kontrollstrukturer vid enklare beräkningsuppgifter
- kunna skriva ett enklare program i form av ett script
- kunna skriva en function som samlar programkod för en specifik beräkningsuppgift
- ha kännedom om olika möjligheter till in- och utmatning, speciellt grafisk in- och utmatning