

Programmering i MATLAB

Jönsson: 6.1-6.4, 6.6-6.11, 7.1-7.7, 8.1-8.5

1 Inledning

Redan första tillfället gjorde ni ett litet program. Ni skrev en **script** eller skriptfil som beräknade summan

$$\sum_{i=1}^5 i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2$$

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 programmering. Vi börjar med att se på skript- och funktionsfiler som vi använder för att ge struktur åt program, därefter ser vi hur vi kan styra ut- och inmatning av data för att avslutningsvis se på kontrollstrukturer som **if**-, **for**- och **while**-satser med vilka vi styr flödet genom koden i ett program.

2 Egendefinierade funktioner

Vi har sett lite funktioner. Nu måste vi bli lite grundligare. Läs igenom avsnittet och läs sedan i Jönsson avsnitt 7.1-7.7.

2.1 Skript

En **script** eller skriptfil är en textfil som innehåller det man skulle kunna skriva direkt i **Command Window**, och som utförs i MATLAB genom att man ger textfilens namn som kommando. För att MATLAB skall hitta filen, förutsätter det att katalogen där filen ligger är aktuell katalog eller man satt en sökväg med **path**, se hjälptexten. Utanför MATLAB får namnet på en **script** tillägget **.m** för att skilja den från andra filer.

Programsatserna i en skriptfil opererar *globalt* på variablerna i arbetsarean (**Workspace**).

Alla utskrifter från programmet skrivs som standard i **Command Window**, liksom alla felmeddelanden. Man kan också styra utskrifter av beräkningsresultat till en fil, se Jönsson avsnitt 3.7 vid behov.

Editorn i MATLAB markerar koden med olika färger för att visa vad som är kommentarer, nyckelord, textsträngar, etc., och har flera funktioner för att underlätta vårt arbete. Ett exempel:

Automatisk indentering. Ifall man markerar en bit programkod, och väljer **Text** och sedan **Smart Indent** så indenteras koden om (dvs. vänstermarginalen justeras). Det kan vara användbart när man klippt-och-klistrat lite kod.

2.2 Funktioner

Det finns flera olika sätt att göra egna funktioner i MATLAB. Om funktionen innehåller flera uttryck eller satser måste man göra en **function** eller funktionsfil, dvs. skapa en textfil med funktionsbeskrivningen. Består funktionen av ett enda uttryck så kan vi göra ett s.k. funktionshandtag (**function handle**) eller s.k. anonyma funktioner (**anonymous functions**).

En **function** är en textfil med samma namn som funktionen och som inleds med en funktionsdeklaration.

För större program kan man vilja använda andra sätt att skriva funktioner, exempelvis underfunktioner (**subfunction**). Vi kan t.ex. ha en funktion som behöver hjälpfunktioner som inte är av intresse utanför huvudfunktionen, då lägger vi dem som underfunktioner.

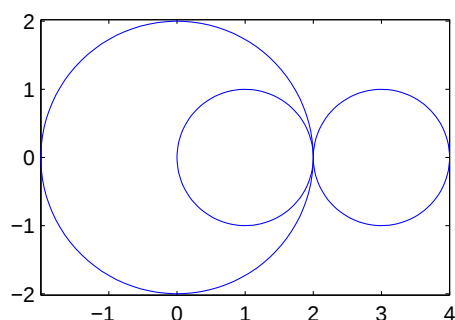
En funktionsfil påminner mycket om en skriptfil. Det som skiljer är att i första raden innehåller textfilen ordet *function* och att argument kan skickas med. Alla variabler som definieras inne i en funktionsfil är *lokala*, dvs. de opererar inte globalt på arbetsarean (**Workspace**).

Exempel 1. Vi vill rita cirklar med olika medelpunkt (a, b) och radie r . Vi gör en funktion med namnet **cirkel.m** med a, b och r som indata och två vektorer x och y som utdata enligt

```
function [x,y]=cirkel(a,b,r)
t=linspace(0,2*pi);
x=a+r*cos(t);
y=b+r*sin(t);
```

I de två vektorerna har vi koordinater för punkter jämnt fördelade på cirkeln. Vi ritar några cirklar med

```
>> [x,y]=cirkel(0,0,2);
>> plot(x,y)
>> axis equal
>> hold on
>> [x,y]=cirkel(1,0,1);
>> plot(x,y)
>> [x,y]=cirkel(3,0,1);
>> plot(x,y)
>> hold off
```



Uppgift 1(a). Gör funktionen som en funktionsfil, med namnet `cirkel.m` förslagsvis. Prova funktionen, rita någon cirkel, vilken typ av vektorer ger den som resultat (rad eller kolonn) och hur långa är de. Ändra er funktion så att ni får 50 punkter på cirkeln, eller varför inte 5 punkter. Rita de "nya" cirkklarna.

(b). Om vi lägger till kommentarer direkt under `function` enligt

```
function [x,y]=cirkel(a,b,r)
% Hjälpstext, det man skriver direkt under function kommer att skrivas
% ut som hjälpstext om man använder help i kommandofönstret
t=linspace(0,2*pi);
x=a+r*cos(t);
y=b+r*sin(t);
```

och skriver `help cirkel` i Command Window så kommer hjälpstexten skrivas ut. Vilken hjälpstext tycker ni är bra till vår funktion? Skriv in er hjälpstext i funktionsfilen och prova med `help`.

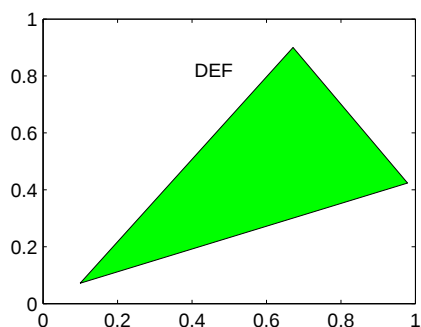
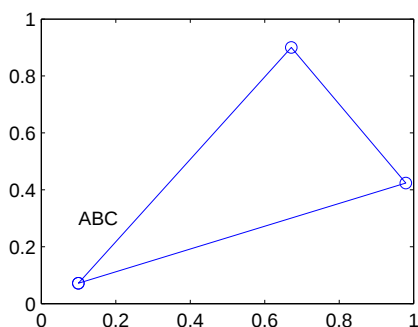
3 In- och utmatning

I samband med programmering behöver vi olika former av in- och utmatning, läs lite snabbt igenom Jönsson avsnitt 3.3, 3.6-3.7. Vi lyfter bara fram grafisk in- och utmatning genom följande exempel.

Exempel 2. Tidigare ritade vi en triangel genom att ge koordinaterna med siffrvärden. 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)]; y=[y; y(1)];  % Vi sluter polygontåget, så alla sidor ritas
>> plot(x,y,'o-')
>> text(0.1,0.3,'ABC')       % Koordinatplacerad text på grafen

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



Med `[x,y,b]=ginput(1)` läser man in koordinater för en punkt och får samtidigt reda på vilken musknapp det trycktes på. Vänster musknapp ger `b` värdet 1, mellersta ger `b` värdet 2, osv. Detta kommer vi ha nytta av lite senare.

4 Kontrollstrukturer

Läs först igenom avsnittet, läs sedan Jönsson avsnitt 6.3-6.4, 6.6-6.10.

4.1 Villkorssatser

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

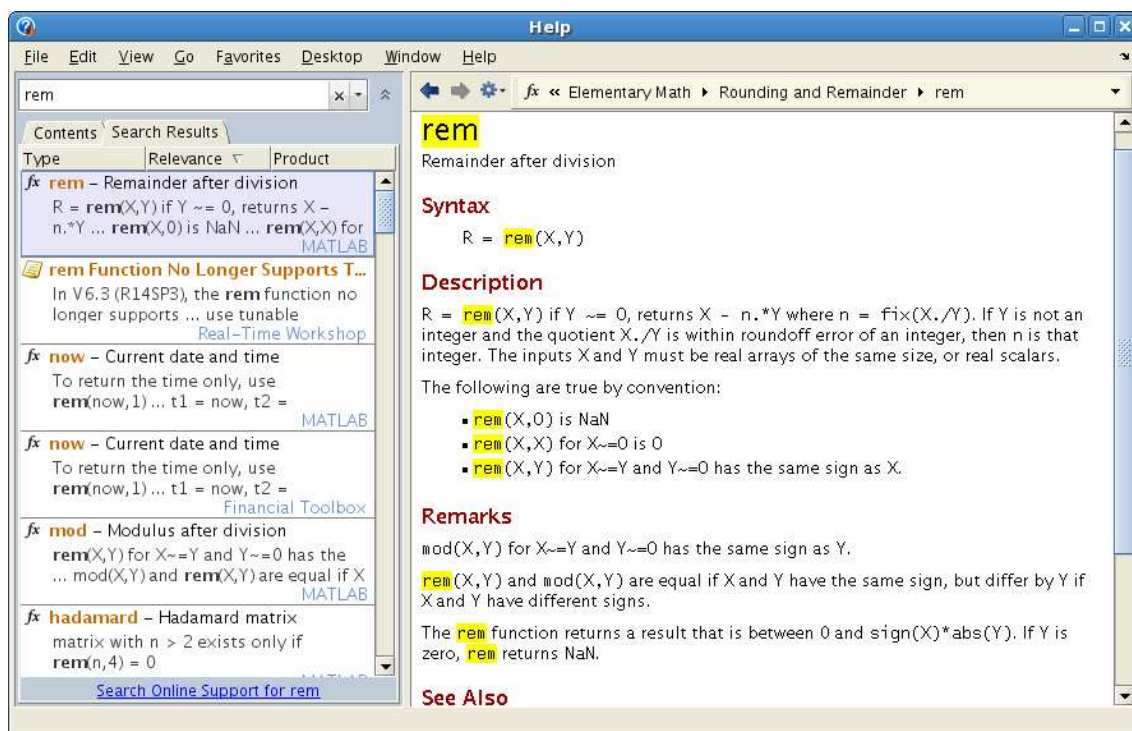
if uttryck	if uttryck	if uttryck
sats	sats	sats
end	else	elseif uttryck
	sats	sats
	end	end

Exempel 3. 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`.

I Help Navigator under Elementary Math i gruppen Rounding and Remainder finner vi funktionen `rem`, resten vid heltalsdivision. T.ex. får `rem(n,2)` värdet 0 om `n` är ett jämnt heltal och värdet 1 om `n` är ett udda heltal.

Så här ser hjälptexten för `rem` ut



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 slumpvalsvektor (`rand` ger slumpstal 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. Prova `help median` för att se en "riktig" hjälptext.

Vi skulle kunna förbättra vår cirkel-funktion lite med en `if`-sats så att vi kan ge antal punkter på cirkeln som indata, om vi vill.

```
function [x,y]=cirkel(a,b,r,n)
if nargin~=4    % nargin ger antal indata som finns med då funktionen används
    n=100;      % om antalet inte fyra, dvs n inte finns med som indata så
end            % får n värdet 100
t=linspace(0,2*pi,n);
x=a+r*cos(t);
y=b+r*sin(t);
```

Med `[x,y]=cirkel(a,b,r)` får vi 100 punkter och det får vi även med `[x,y]=cirkel(a,b,r)`, medan t.ex. `[x,y]=cirkel(a,b,r,50)` ger 50 punkter.

Uppgift 2. Prova att förändra din cirkel-funktion!

4.2 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 ärs en `while`-sats lämpligare.

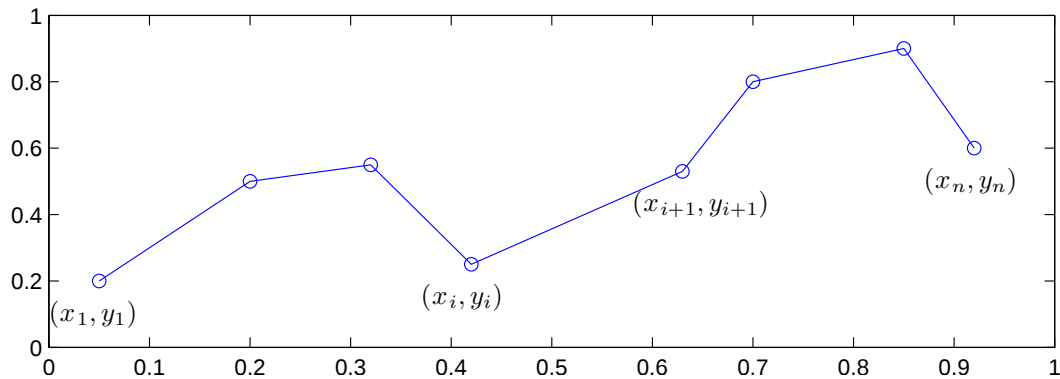
4.2.1 for-satser

Det allmänna utseendet på en **for**-sats är

```
for variabel = uttryck
    sats
end
```

Vi har redan sett på några summor som vi beräknar med **for**-sats och här kommer ytterligare ett exempel.

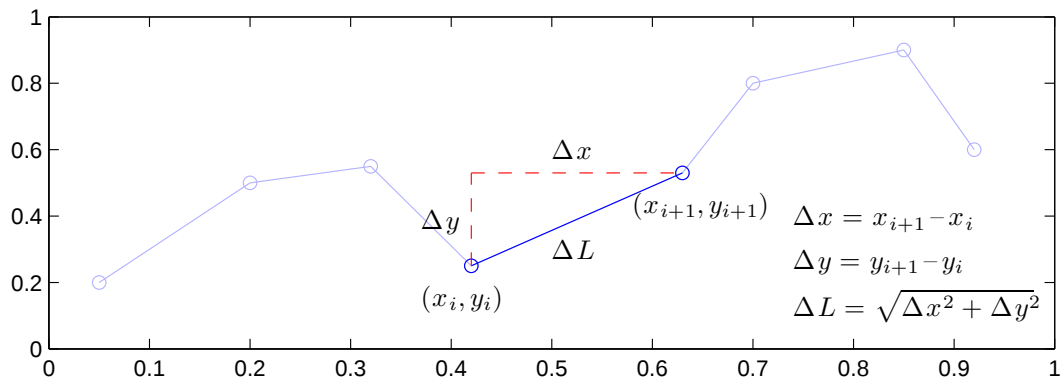
Exempel 4. Vi tänker oss att vi har ett polygontåg $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ som vi ritat en figur av



Vill vi beräkna polygontågets längd kan vi göra det med

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

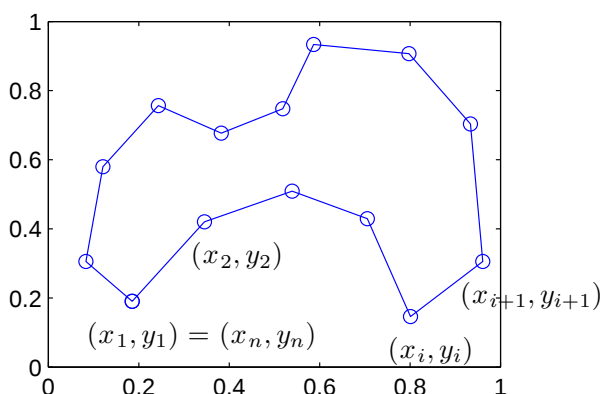
Denna formel fås genom att använda Pytagoras sats på varje segment i polygontåget.



Antag att koordinaterna samlade i två vektorer $\mathbf{x} = (x_1, x_2, \dots, x_n)$ och $\mathbf{y} = (y_1, y_2, \dots, y_n)$, 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
```

Vi tänker oss nu att vi har ett annat polygontåget som är slutet, dvs. att $x_n = x_1$ och $y_n = y_1$, som vi ritat en figur av



Vill vi beräkna arean av det område som omsluts kan vi använda formeln

$$A = \left| \frac{1}{2} \sum_{i=1}^{n-1} (x_{i+1} + x_i)(y_{i+1} - y_i) \right|$$

Denna formel är lite svårare, men i läsperiod 3 kommer ni läsa den matematik som behövs för att ta fram den. Så här beräknar vi arean i alla fall

```
>> n=length(x);
>> A=0;
>> for i=1:n-1
    A=A+(x(i+1)+x(i))*(y(i+1)-y(i));
end
>> A=abs(A)/2;
>> A
```

Uppgift 3. Skriv två funktioner för beräkning av längden respektive arean enligt exempel 4. Prova funktionerna på en triangel och en rektangel.

4.2.2 while-satser

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

```
while uttryck
    sats
end
```

Uttrycket i **while**-satsen är en matris och repetition sker så länge alla elementen i matrisen är skilda från noll.

Exempel 5. Man kan beräkna $\sqrt{c}$ med upprepade additioner och divisioner med iterationsformeln

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

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

Vi beräknar t.ex. approximation av $\sqrt{2}$ med noggrannheten $\text{tol} = 10^{-16}$ 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 4. Det gäller att $\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{\pi}{4}$. Hur många termer måste man ta med i summan för att approximera π med fem korrekta decimaler.

Ibland har man nytta av följande konstruktion

```
while 1
    sats
    if uttryck
        break
    end
end
```

Här kommer upprepning ske ända tills uttrycket i `if`-satsen blir sant (får värdet 1), då avbryts `while`-satsen och programkoden efter denna utförs.

Exempel 6. Vi vill rita ett polygontåg genom att markera punkter med `ginput` och succesivt rita upp linjer som förbinder på varandra följande punkter.

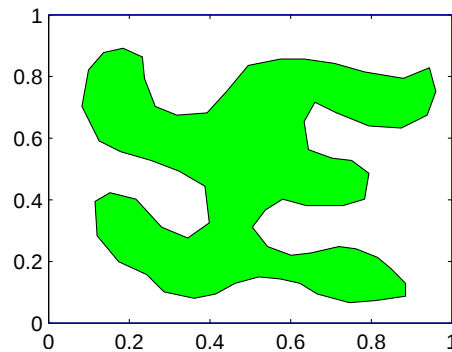
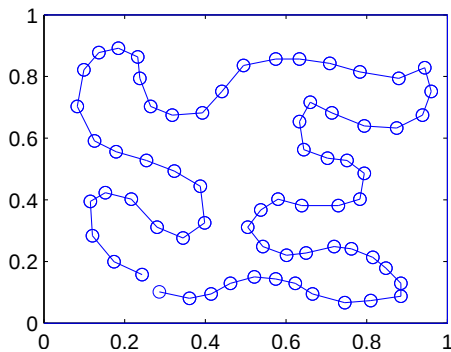
Vi håller på att markera nya punkter och rita linjer så länge vi trycker på vänster musknapp, annars avbryter vi.

```

>> axis([0 1 0 1]), hold on
>> [x,y]=ginput(1);
>> plot(x,y,'o')
>> xpol=x;
>> ypol=y;
>> while 1
    [x,y,knapp]=ginput(1);
    if knapp~=1
        break
    end
    xpol=[xpol x];
    ypol=[ypol y];
    plot(xpol(end-1:end),ypol(end-1:end),'o-')
end
>> hold off

```

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



Uppgift 5. Skriv en skriptfil som gör det möjligt att markera hörnpunkter i ett polygonområde och som beräknar arean av polygonområdet samt längden av dess rand, med hjälp av funktionerna från uppgift 3. Områdets inre skall fyllas i med någon färg. Se exempel 2 och 6.

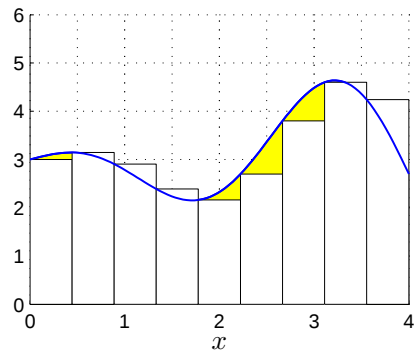
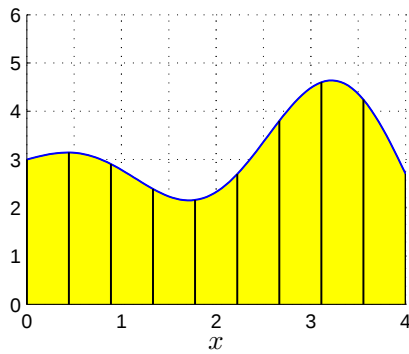
5 Vektorisering och effektivitet

Läs Jönsson avsnitt 8.1-8.5 utom exempel 8.3 och 8.4.

Tidtagning är ett bra sätt att studera effektivitet. Det man kan tjäna i särklass mest på när det gäller effektivisering är fördimensionering av matriser och vektorer samt vektorisering, som vi skall se några exempel på nu.

Exempel 7. (Ersätter exempel 8.3 i Jönsson.) En integral kan approximeras med **vänster rektangelregel** enligt

$$\int_a^b f(x) dx = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} f(x) dx \approx \sum_{i=1}^n h f(x_{i-1}), \quad \text{där } h = \frac{b-a}{n}$$



Vi kan beräkna t.ex. $\int_0^1 x \sin(x) dx$ med vänster rektangelregel med $n = 100$ säg, på följande sätt

```
>> n=100;
>> a=0; b=1; f=@(x)x.*sin(x);
>> h=(b-a)/n
>> q=0;
>> for i=0:n-1
    x=a+i*h;
    q=q+h*f(x);
end
```

Att använda en `for`-sats fungerar, men vi genererar hellre en vektor av alla funktionsvärdena $f(x_i)$ och sedan summerar dessa enligt

```
>> n=100;
>> a=0; b=1; f=@(x)x.*sin(x);
>> x=linspace(a,b,n+1);
>> h=(b-a)/n;
>> q=sum(h*f(x(1:n)))
```

Detta sätt att organisera en beräkning kallas att vektorisera den, dvs. man genererar först en eller flera vektorer och utför sedan den önskade beräkningen på dem.

De komponentvisa operationerna `.*` `./` `.^` är exempel på vektoriserade operationer. Vi använde funktionen `sum` som snabbt summerar en vektor.

Exempel 8. (Ersätter exempel 8.4 i Jönsson.) I exempel 4 såg vi på ett polygontåg, som gavs av $(x_1, y_1), \dots, (x_n, y_n)$. Polygontågets längd och polygonområdet ges av

$$L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad \text{och} \quad A = \left| \frac{1}{2} \sum_{i=1}^{n-1} (x_{i+1} + x_i)(y_{i+1} - y_i) \right|$$

Om koordinaterna samlade i två vektorer $\mathbf{x} = (x_1, x_2, \dots, x_n)$ och $\mathbf{y} = (y_1, y_2, \dots, y_n)$, så beräknade vi längden med

```
>> 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
```

och arean med

```
>> n=length(x);  
>> A=0;  
>> for i=1:n-1  
    A=A+(x(i+1)+x(i))*(y(i+1)-y(i));  
end  
>> A=abs(A)/2;
```

Betydligt effektivare blir det om vi använder vektorbaserade operationer enligt

```
L=sum(sqrt((x(2:end)-x(1:end-1)).^2+(y(2:end)-y(1:end-1)).^2));
```

```
A=abs(sum((x(2:end)+x(1:end-1)).*(y(2:end)-y(1:end-1)))/2);
```

Exempel 9. Beskriv följande funktion i MATLAB och rita dess graf.

$$f(x) = \begin{cases} 0, & x < 1 \\ 1, & 1 \leq x \leq 2 \\ 0, & x > 2 \end{cases}$$

Alternativ med `for`-sats

```
function f=stepfun(x)  
f=zeros(size(x));  
for k=1:length(f)  
    if 1<=x(k) & x(k)<=2  
        f(k)=1;  
    end  
end
```

Vi ritas bild med

```
x=linspace(-1,3,200);  
plot(x,stepfun(x))  
axis([-1 3 -1 2])
```

Smartare alternativ med `find`

```
function f=stepfun(x)  
f=zeros(size(x));  
f(find(1<=x & x<=2))=1;
```

