FYSY115 Johdatus Matlabin käyttöön
Kevät 2017

Taneli Kalvas
taneli.kalvas@jyu.fi
FL114

Jyväskylän yliopisto, Fysiikan laitos

http://users.jyu.fi/~tvkalvas/fysy115/

Motivaatio

- Tietokoneen käyttö ongelmanratkaisuun on arkipäivää fyysikolle akateemisessa maailmassa ja yrityksissä
- Matlab on standardityökalu yritysmaailmassa ja laajalti käytössä myös yliopistoissa
- Muita työkaluja
  - Käännettävät kielet: C, C++, C#, Java, Fortran, ...
  - Skriptikielet: Python, Mathematica, ...
- Ohjelmoinnin perusajatus sama kaikissa kielissä. Jollain kielellä on aloitettava — Matlab hyvä ensimmäinen numeriikan työkalu fyysikolle
- Graafien piirtäminen laboratoriotyöselostuksissa
- FYSP120-kurssilla käytetään Matlabia

FYSY115-kurssi

Laajuus 1 op:

Kontaktitunnit	16 h	8×2 h
Harjoituksia kotona	10 h	
Ohjelmiston asentaminen	1 h	
Yhteensä	27 h	

Kontaktitunnit tiistaisin 12:15–14:00, FYS1:

10.1. | 17.1. | 24.1. | 31.1. | 7.2. | 14.2. | 21.2. | 28.2.

Kolme ensimmäistä kertaa vain “luentoja”

Viidellä viimeisellä kerralla käydään myös läpi harjoitustehtävät

FYSY115-kurssi

- Esitiedot: Virkeä mieli ja yleissivistys

- Harjoitukset palautetaan sähköpostilla: **otsikkoon FYSY115/Harjoitus X**
- Saa tehdä parityönä
- Kurssimerkintä tulee suorittamalla 60 % harjoituksista
- Kussakin harjoitussetissä on myös haastavampia tehtäviä
  (saat 60 % helposti vaikka jokunen jäisi tekemättä)

- Kirjallisuus
  - J. W. Eaton, et. al, *GNU Octave*, official documentation
  - MathWorks, *Matlab Documentation*, `http://www.mathworks.com/`
  - MathWorks, *Matlab Programming Fundamentals* (1078 sivua),
    `http://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf`
  - Lukuisia lyhyempiä Matlab-oppaita internetistä

Kurssin sisältö

Tullaan tutuiksi Matlab/Octave -ohjelmiston kanssa:

- Käyttö taskulaskimena
- Perussyntaksi
- Tyypit: reaaliluvut, kompleksiluvut, vektorit, matriisit
- Operaattorit: $+$, $-$, $*$, $/$, $\hat{}$, ...
- Muuttujat, anonyymit ja inline-funktiot
- Datan graafinen esittäminen, datan lukeminen tiedostosta
- Funktio- ja skriptitiedostot
- Ohjelmoinnin alkeet, ehtolausekkeet, silmukkarakenteet
- Aliohjelmat
- Virheiden etsiminen ja korjaaminen
- Liikeyhtälöiden iterointi Eulerin menetelmällä

Osaamistavoitteet

Osaamistavoite	1. Muistaa	2. Ymmärtää	3. Soveltaa	4. Analysoida	5. Arvioida	6. Luoda
Matlabin operaattorit					O	
Muuttujat				O		
Tiedon luku ja kirjoitus			O			
Graafinen esittäminen			O			
Funktio- ja skriptitiedostot			O			
Silmukka- ja ehtolauseet				O		
Aliohjelmat			O			
Debuggaus		O				
Tiedon haku			O			
Algoritmit		O				

Matlab-ohjelmiston hankkiminen

- Matlab on kaupallinen ohjelmisto
- Yliopistolla 102 kpl verkkolisenssejä jyu-verkossa
- Löytyy yliopiston tietokoneilta atk-luokista
- Voi asentaa myös omalle koneelle kotitehtäviä varten:
  `https://www.jyu.fi/itp/palvelut/ohjelmistot/ohjelmistot/matlab`
- Yksittäinen opiskelijalisenssi 35 €
- Saatavilla Windows, Mac OSX ja Linux -alustoille
- Asennus varsin suoraviivainen

Octave-ohjelmisto

- Octave on Matlabin ilmainen klooni
- GNU General Public License, käytettävissä kaikkialla ja aina, 0 €
- Löytyy yliopiston tietokoneilta atk-luokista
- Saatavilla Windows, Mac OSX ja Linux -alustoille
- Linux-koneilla asennus helposti paketinhallintatyökaluilla
- Windowsille ja Macille:
  `https://www.gnu.org/software/octave/`
- Uusin versio Octave 4.2.0 (14.11.2016), kurssilla käy vanhempikin versio
- Muitakin Matlab-klooneja on: esim. SciLab

Matlab vs. Octave

- Octavessa toteutettuna kaikki Matlabin perustoiminnot
  - FYSY115 ja FYSP120-kursseilla pärjää molemmilla!
- Eroavaisuuksia kuitenkin on.
- Octave on hidas laskemaan (ei merkitystä näillä kursseilla ja raskaiden ongelmien ratkaisuun käännetyt ohjelmointikielet)
- Matlab on hidas käynnistymään
- Octavessa joustavampi syntaksi

Tarkemmin eroavaisuuksista:

`http://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB`

Matlab-ympäristön esittely

Matlab-ympäristön esittely

Kehote, syöte ja tuloste

```
>> 1+2
ans = 3
```

Vastaavasti myös muut normaalit taskulaskintoiminnot

```
1-2
2*3
3/(2+2)
sin(1)    % Huom, trigonometria aina radiaaneina
sqrt(2)
2^10
```

Matlab ei kuitenkaan ole taskulaskin (ei takaa tuloksen tarkkuuta)

```
0.2+0.3-0.4-0.1
```

Laskenta suoritetaan ns. liukulukuja käyttäen, joilla rajallinen tarkkuus:
$1 + \epsilon = 1$, mikäli $\epsilon < 2.22 \cdot 10^{-16}$

Muuttujat

Kaikkien laskutoimitusten tulos tallennetaan johonkin muuttujaan, joka on `ans` ellei muuta määrätä. Voidaan siis laskea

```
2*2
ans*2
```

Tulos voidaan tallentaa muuttujaan *sijoitusoperaattorilla* =

```
x = 5
```

Määrittelyn jälkeen muuttujaa voidaan käyttää laskutoimituksissa

```
2*x
```

Muuttujan nimi alkaa kirjaimella ja siinä voi olla kirjaimien lisäksi numeroita ja alaviivoja. Pien- ja suuraakkosilla on ero.

```
hauki1 = 10
Kala = hauki
xy_52 = 2+5^4
```

Huom: `2x` ei ole `2*x` vaan syntaksivirhe

Muuttujat

Sisäänrakennettuja muuttujia ovat mm. `pi`, `eps`, `i` ja `j`. Luonnotieteen vakiot pitää määritellä itse. Muuttujan `i` avulla voidaan määritellä kompleksilukuja

```
1+5i
```

Määritellyt muuttujat voidaan tulostaa komennolla `who` ja lisää tietoa niistä saadaan komennolla `whos`

```
who
whos
```

Yhden tai kaikki muuttujat voidaan poistaa muistista komennoilla

```
clear x
clear all
```

Tulostustarkkuus (ei vaikuta laskentaan)

```
format short     % Oletus
format long
```

Tyypit

Matlabissa muuttujia on useaa tyyppiä. Lähtökohtaisesti kaikki numerot ovat 64-bittisiä liukulukuja (double) ellei muuta määritellä

```
x = 2.4
```

Muita peruskäytössä tarvittavia tyyppejä ovat merkit (char)

```
y = 't'
```

merkkijonot

```
z = 'Hauki on kala'
```

ja totuusarvot, jotka seuraavat vertailuoperaatioista, esimerkiksi

```
q = 2 < 4
```

Toisinaan tyyppejä täytyy muuttaa toiseksi. Esimerkiksi voimme selvittää mistä kokonaisluvuista merkkijono koostuu

```
r = int8('kala')
```

Komentojen ja funktioiden dualismi

Funktion argumentit (jos niitä on) syötetään sulkeisiin funktion nimen perään pilkuilla erotettuna.

Komento on erityinen funktio, jonka argumentit erotetaan välilyönneillä. Komento voi ottaa argumenteikseen vain merkkijonoja. Matlabin komentoja voi kutsua myös käyttäen funktioiden syntaksia.

Funktioita:

```
sin(pi)
clear( 'all' )
who()
disp( pi )
cd( '..' )
dir()
quit()
```

Komentoja:

```
clear all
who
disp pi
cd ..
dir
quit
```

Syötteestä

Matlab tulostaa laskutoimituksen vastauksen ruudulle. Suurilla datamäärillä tai pitkissä laskutoimituksissa tämä ei ole toivottavaa. Tulos saadaan piilotettua puolipisteellä

```
2+2;
```

Syötteessä `%` on kommenttimerkki, jonka jälkeen samalla rivillä olevan syötteen Matlab jättää huomiotta, esim.

```
2^2 - 5   % hyödyllinen kommentti
```

Rivin jatkaminen

```
1 + 2 + 3^1 - 7/3 ...
  2*(5 - 1)
```

Loogisia operaattoreita ja funktioita

Totuusarvot

```
1 = tosi
0 = epätosi
```

Vertailuoperaattorit

```
1 == 2
1 < 2
1 <= 1
1 > 2
1 >= 2
2 ~= 0.2
```

Loogiset operaattorit

```
1 && 0
1 || 1
xor(1,1)
~1
```

Hyödyllisiä matemaattisia funktioita

```
sin(x)
cos(x)
tan(x)
asin(x)
acos(x)
atan(x)
atan2(y,x)
log(x)
log10(x)
exp(x)
abs(x)
round(x)
ceil(x)
floor(x)
mod(x)
```

Vektorit ja matriisit

Skaalaarimuuttujien lisäksi Matlabissa voidaan käsitellä vektoreita ja matriiseja.

Vektorithan ovat kaikille tuttuja N-ulotteisen (vektori)avaruuden alkioita, missä N on vektorin alkioiden lukumäärä. Esimerkiksi

$$\vec{x} = \begin{pmatrix} 2 \\ -5 \\ 4 \end{pmatrix},$$

missä $\vec{x}$ on 3-ulotteinen pystyvektori ja

$$\vec{y} = \begin{pmatrix} 1 & -3 & 2 & 6 \end{pmatrix},$$

missä $\vec{y}$ on 4-ulotteinen vaakavektori.

Vektorit ja matriisit

Mikäli kuvaus M-ulotteisesta vektoriavaruudesta N-ulotteiseen vektoriavaruuteen on lineaarinen eli se voidaan lausua summana

$$y_i = \sum_{j=1}^{N} a_{i,j} x_j ,$$

niin se voidaan esittää $N \times M$-matriisin A avulla, esimerkiksi

$$\vec{y} = A\vec{x} \quad \text{eli} \quad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} .$$

Yllä on käytetty matriisin ja vektorin kertolaskua kuvaukseen. Vastaavasti vektori $\vec{y}$ kuvataan K-ulotteiseen avaruuteen $K \times N$-matriisin B avulla laskemalla $\vec{z} = B\vec{y}$. Vektori $\vec{z}$ voidaan laskea myös

$$\vec{z} = B\vec{y} = BA\vec{x} = C\vec{x},$$

missä $K \times M$-matriisi C saadaan matriisien B ja A tulona. Lisää vektorien ja matriisien laskutoimituksista lineaarialgebran kursseilla.

Vektorit ja matriisit

Matlabissa vektorit ovat matriisien erikoismuoto. Pystyvektori on matriisi, jossa on vain yksi sarake ja vaakavektori on matriisi, jossa on vain yksi rivi.

Matriisi kirjoitetaan hakasulkujen sisään ja rivinvaihto merkitään puolipisteellä. Matlab itse tulostaa matriisin “oikeassa” muodossa

```
>> A = [1 2 3; 4 5 6]
A =
   1   2   3
   4   5   6
```

Vastaavasti voidaan määritellä vaakavektori

```
A = [1.1 0.1 -2.2]
```

ja pystyvektori

```
A = [6; 5; 4; 3; 2]
```

Vektorit ja matriisit

Yksittäiseen alkioon/elementtiin voidaan viitata muodossa `A(i,j)`, esimerkiksi

```
A(2,1) = 4
```

jolla asetetaan matriisin `A`, toisen rivin, ensimmäisen sarakkeen alkion arvoksi 4. Huom: indeksointi Matlabissa alkaa luvusta 1. Asetettaessa alkiota voidaan kasvattaa automaattisesti matriisin/vektorin kokoa:

```
b = []
b(1) = 2
b(2) = -5
```

Matlabilta voidaan kysyä vektorin/matriisin kokoa `size`-funktiolla:

```
>> a = [1 2 3; 4 5 6];
>> size(a)
ans =
  2  3
```

Vektorit ja matriisit

Vektorien ja matriisien laskutoimitukset ovat voimassa vain kun tietyt ehdot toteutuvat niiden dimensioille: Operaatioilla + ja − dimensiot oltava samat:

```
a = [1 2 3; 4 5 6]
b = [3 2 1; -1 -2 -3]
c = [1 2 3 4; 5 6 7 8; 9 10 11 12]
a+b
a-b
a+c    % Ei onnistu
```

Matriisien kertolaskulle ensimmäisen matriisin sarakemäärän on oltava sama kuin toisen matriisin rivien lukumäärä. Esimerkiksi laskutoimitus

```
a*c
```

voidaan laskea koska a:n koko on 2×3 ja c:n koko on 3×4. Laskutoimitus

```
c*a
```

taas ei ole määritely.

Vektorit ja matriisit

Vektoreita ja matriiseja käytetään Matlabissa usein skaalaarien taulukkoina. Esimerkiksi jos meillä on kymmenen erimassaista palloa, voimme tallentaa massat vektoriin

```
m = [1.1 2.2 0.9 2.1 2.2 1.2 1.5 2.0 1.2 3.5]
```

Jos pallojen nopeudet ovat vastaavasti taulukossa v, saadaan liikemäärä laskettua skalaaritulona

```
v = [42 55 27 43 29 76 43 36 78 34]
p = m.*v
```

Huom! Operaattori `*` on matriisi/vektoritulo. Skalaaritulo kullekin alkioparille saadaan laskettua operaatorilla `.*`

Vastaavasti on määritely myös skalaarioperaattorit jakolaskulle `./` ja potenssille `.^`

Yhteen- ja vähennyslasku lasketaan joka tapauksessa alkioittain joten erillistä skalaariyhteenlaskua tai skalaarivähennyslaskua ei ole tarpeen määritellä.

Vektorit ja matriisit

Vektoreille ja matriiseille on määritely myös muita operaatioita:

```
a = [1 2 3; 4 5 6]
b = a'     % Transpoosi
2*a        % Skalaaritulo
a/10       % Skalaariosamäärä
a+5        % Vakion lisääminen kaikkiin alkioihin
a-2        % Vakion vähentäminen kaikista alkioista
```

Vektorin ja matriisin "jakolasku", ratkaisee yhtälön $AX = B$

```
A = [6 12 4; 7 -2 3; 2 8 -9]
B = [70;5;64];
A\B
```

Myös oikealta voidaan jakaa, ratkaisee yhtälön $XA = B$

```
A = [6 12 4; 7 -2 3; 2 8 -9]
B = [25;-14;53];
B/A
```

Lisää matriisiyhtälöistä (eli yhtälöryhmistä) lineaarialgebran kursseilla.

Vektorit ja matriisit

Kaksoispisteoperaattori

```
1:10        % Muodostetaan vektori lukuja yhdestä kymmeneen
1:2:10      % Vektori kahden välein yhdestä kymmeneen
0:0.1:1     % Vektori lukuja nollasta yhteen 0.1 välein
5:-1:2      % Vektori lukuja viidestä alaspäin kahteen asti,
            % yhden välein
```

Kaksoispistettä voidaan käyttää myös indeksointiin:

```
A(1,4:6)    % Palauttaa sarakkeet 4-6 rivillä 1
A(1,4:2:8)  % Palauttaa sarakkeet 4,6,8 rivillä 1
A(2,:)      % Palauttaa kaikki sarakkeet rivillä 2
A(:,5)      % Palauttaa kaikki rivit sarakkeessa 5
```

Lisäksi on olemassa avainsana `end`, jolla voidaan viitata viimeiseen riviin/sarakkeeseen:

```
A(:,end)    % Palauttaa kaikki rivit viimeiseltä sarakkeelta
```

Operaattorit

Operaattoreiden laskentajärjestys Matlabissa:
Korkeammalla taulukossa → lasketaan ensin, muuten vasemmalta oikealle.

`( )`	Sulkeet
`' .' ^ .^`	Kompleksikonjugaattitranspoosi, transpoosi, potenssit
`+ - ~`	Etumerkit, looginen negaatio
`* .* / ./ \ .\`	Kertolaskut, jakolaskut
`+ -`	Summa, erotus
`:`	Kaksoispisteoperaattori
`== ~= <= < >= >`	Vertailut
`&`	alkioittainen ja
`|`	alkioittainen tai
`&&`	looginen ja
`||`	looginen tai

Funktiot

Useimpia sisäänrakennettuja funktioita voidaan käyttää monella tavalla. Esim:

```
A = [1 -3 2]
B = [5 8 -7]
```

Suurimman alkion arvo vektorista saadaan kutsumalla

```
max(A)
```

Toinen kutsumuoto

```
C = max(A,B)
```

palauttaa vektorin C, jossa $C_i = A_i$, jos $A_i > B_i$, muutoin $C_i = B_i$.
Funktio voi palauttaa myös useamman arvon, jotka täytyy sijoittaa muuttujiin käyttäen seuraavaa syntaksia:

```
[X,I] = max(B)
```

Tämän `max`-funktion tapauksessa tämä kutsun muoto palauttaa suurimman alkion arvon X ja sen sijainti-indeksin I.

Lisätietoa `help`-komennolla, esim. `help max`.

Funktiot

Muita hyödyllisiä sisäänrakennettuja funktioita on mm.

`linspace(x1,x2)`	Tuottaa 100-pituisen vektorin lukuja tasavälein väliltä $[x1, x2]$
`linspace(x1,x2,N)`	Tuottaa N-pituisen vektorin lukuja tasavälein väliltä $[x1, x2]$
`logspace(x1,x2,N)`	Tuottaa N-pituisen vektorin lukuja logaritmisesti väliltä $[10^{x1}, 10^{x2}]$
`zeros(N,M)`	$N \times M$ nollamatriisi
`zeros(N)`	$N \times N$ nollamatriisi
`ones(N,M)`	$N \times M$ matriisi ykkösiä
`eye(N)`	$N \times N$ identiteettimatriisi
`rand(N,M)`	$N \times M$ satunnaismatriisi, luvut tasajakaumasta väliltä $[0, 1]$
`length(x)`	Palauttaa x-vektorin pituuden
`[n,m] = size(x)`	Palauttaa x-matriisin koon
`n = size(x,k)`	Palauttaa x-matriisin rivien määrän jos $k = 1$ ja sarakkeiden määrän jos $k = 2$
`sum(x)`	Palauttaa vektorin x alkioiden summan

Funktiot

Yksinkertaisia yhden rivin funktioita voi määritellä kahdella tavalla.

Anonyymi funktio määritellään käyttäen syntaksia `h = @(arg) func`, missä `h` on funktion kahva (nimi), `arg` on argumenttilista pilkulla erotettuna ja `func` on funktion määrittely. Esimerkkejä:

```
f2c = @(x) (x-32)*5/9
dist = @(x,y) sqrt(x.^2+y.^2)
```

Inline-funktio määritellään merkkijonosta, esimerkiksi

```
f2c = inline( '(x-32)*5/9' )
dist = inline( 'sqrt(x.^2+y.^2)' );
dist = inline( 'sin(2*pi*f + t)', 't', 'f' );
```

Jos argumentteja ei ole listattu ne tunnistetaan automaattisesti ja järjestetään aakkosjärjestykseen.

Inline-tyypin funktio tulee poistumaan tulevaisuudessa. Käytä anonyymeja funktioita!

Funktiot

Anonyymin funktion määrittelyssä voidaan käyttää muuttujia, esim.

```
K = 1.2
f = @(t) K*t.^2
```

Funktio `f` ei ole kuitenkaan riippuvainen muuttujasta `K`. Vaikka `K`:n arvo muuttuisi funktion määrittelyn jälkeen on funktio edelleen `1.2*t^2`. Anonyymin funktion määritelmä on **eksplisiittinen**.

Vastaavasti anonyymi funktio voi olla riippuvainen toisista funktioista, mutta alemman tason funktion määritelmää ei voi muuttaa jälkikäteen. Esim:

```
f = @(x) x.^2
g = @(x) 4*f(x)
f = @(x) x
```

Funktion `g` määritelmä on edelleen `4*x^2`, vaikka `f` määriteltiin uudelleen.

Datan graafinen esittäminen

Plot

Yksinkertaisin ja kenties myös kaikkein käytetyin esitysmuoto, kaksiulotteinen esitys pistejoukolle voidaan tuottaa komennolla `plot(x,y)`. Tälle funktiolle on annettava esitettävät datapisteet vektorimuodossa, esimerkiksi

```
x = linspace( -pi, pi );
plot( x, sin(x) );
```

`plot`-komennolle voi antaa kolmantena argumenttina esitystyylin merkkijonona

```
x = linspace( -pi, pi, 20 );
plot( x, sin(x), 'ro' );
```

Useita graafeja voi tulostaa samaan kuvaan kahdella tavalla

```
plot( x, sin(x), 'r', x, cos(x), 'b' );
```

tai

```
plot( x, sin(x), 'r' );
hold on
plot( x, cos(x), 'r' );
hold off
```

Plot-komennon perustyylit

Matlabin dokumentaatiosta:

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

Plot-komennon hienosäätäminen

Perussäätöjen lisäksi lähes kaikkia graafin esitykseen vaikuttavia ominaisuuksia voi säätää *PropertyName, PropertyValue* -pareja hyödyntämällä. Esim:

```
x = -pi:pi/10:pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```

Lisätietoa Matlabin dokumentaatiosta. Toistaiseksi pärjätään perusominaisuuksilla.

Plot

Kuvaan voidaan lisätä selitteitä ja sen skaalaa voidaan muuttaa, esim:

```
t = linspace(0,10*pi,1000);
x = sin(t);
y = cos(1.4*t);
plot(x,y,'-r')
xlabel( 'x (mm)' )
ylabel( 'y (mm)' )
title( 'Heilurin rata (sin \theta, cos 1.4\theta)' )
axis( [-1 1 -1 1], 'square' )
```

Tekstiä voi muotoilla TEX-syntaksin mukaisesti.

Kuva voidaan myös tallentaa komennolla `print`. Esimerkkejä:

```
print -dpng kuva.png
print -dpdf kuva.pdf
print -depsc2 kuva.eps
```

Plot

Oletusarvoisesti kuva tulostetaan ruudulle ympäristöön “Figure 1”. Aktiivinen ympäristö voidaan valita komennolla `figure(N)` tai uusi figure ympäristö voidaan luoda kutsumalla vain `figure`. Kuvaikkunoita voidaan sulkea komennolla `close`.

Esim:

```
x = linspace(0,10);
figure(1)
plot(x,x.^2-8*x)
figure(2)
plot(x,sin(x))
close all
```

Muut graafisen esityksen tyylit

Muita tyylejä graafiseen esitykseen käytetään eri komentojen kautta, mm:

```
plotyy                  errorbar
semilogx                polar
semilogy                quiver
loglog                  pcolor
bar                     pie
barh                    rose
stairs
hist                    mesh
stem                    surf
scatter                 plot3
contour
contourf
```

Näistä lisää dokumentaatiossa, myöhemmin tällä kurssilla ja/tai FYSP120-kurssilla.

Skriptit ja funktiot

Skriptitiedosto

Monimutkaisia, pitkiä tai toistuvia komentojen sarjoja ei kannata kirjoittaa käsin Matlabiin. On järkevämpää käyttää *skriptitiedostoja* (engl. script file), johon komennot on tallennettuna. Tällaisen tiedoston ajaminen on ekvivalenttia sen kanssa, että komennot kirjoitettaisiin käsin kehotteelle.

Matlabin skriptit ovat tavallisia tekstitiedostoja, joita voi käsitellä millä tahansa tekstieditorilla. Matlabissa on oma sisäänrakennettu editorinsa, mutta esim. Octavessa sellaista ei ole. Suositeltavia tekstieditoreita Octaven kanssa käytettäväksi ovat mm. *Vim*, *Emacs* ja *NotePad++*.

Esimerkkitiedosto `esim.m`:

```
a = 9.81;
t = linspace(0,3,100);
v = v0+a*t
x = v0*t+0.5*a*t.^2;
xend = x(end)
vend = v(end)
plot(t,x,t,v)
```

Skriptin ajaminen:

```
>> esim
xend =  44.145
vend =  29.430
>>
```

Skriptiesimerkki

Tehdään ohjelma, joka

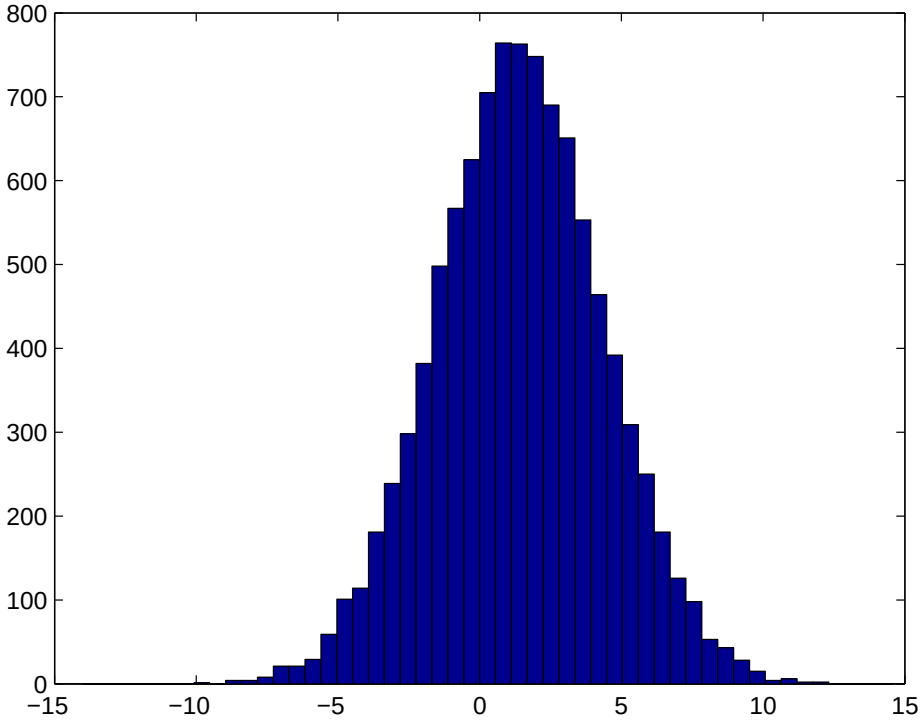
- Tuottaa satunnaislukuja Gaussin jakaumasta keskiarvolla 1.4 ja keskihajonnalla 3
- Tuottaa lukujen jakaumaa kuvaavan histogrammiin
- Plottaa histogrammin

Tutkitaan vapaiden parametrien vaikutusta tulokseen

- Satunnaislukujen määrä
- Histogrammin lokerojen määrä ja laajuus

Skriptiesimerkki gausstest.m

```
x = normrnd( 1.4, 3, 10000, 1 );
N = 50;
edges = linspace( -14, 14, N+1 );
counts = histc( x, edges );
bar( edges, counts, 'histc' );
```



Huomioita skriptiesimerkistä

- Ohjelman muokkaaminen ja muutoksien testaaminen on helppoa
- Muuttujat jäävät yleiseen työmuistiin
- Parametrien muuttaminen vaatii skriptin muokkaamista tai skripti voidaan tehdä riippuvaiseksi ennakkoon määritellyistä muuttujista.
- Skriptit sopivat pienten yksittäisten ongelmien käsittelyyn, mutta laajemmissa kokonaisuuksissa tiedon välitys muuttujien avulla osoittautuu hankalaksi.
- Halutaan ajatella aliohjelmia “mustina laatikkoina”.
- Ratkaisu: funktiotiedostot

Funktiotiedosto

Aiemmin käytettiin yhden rivin mittaisia anonyymejä funktioita. Jos halutaan tehdä jotain monimutkaisempaa on kirjoitettava funktio tiedostoon.

Funktiotiedosto on kuten skriptitiedosto, mutta muutamin poikkeuksin

- Funktiolle voi välittää argumentteja
- Funktio palauttaa arvon/arvoja
- Funktio toimii omassa työtilassaan,
  - Funktio ei näe yleisessä työtilassa eikä muissa funktioissa olevia muuttujia
  - Funktiossa määritellyt muuttujat eivät jää Matlabin työmuistiin

Funktiotiedosto `f2c.m` alkaa rivillä, joka esittelee funktion:

```
function result = f2c( f )
  result = (f-32)*5/9;
end
```

Esimerkkejä funktiotiedoston esittelyrivistä

Yksi argumentti, yksi palautettava muuttuja

```
function [area_square] = square( side )
```

Sulut eivät pakolliset yhdelle palautettavalle

```
function area_square = square( side )
```

Kaksi argumenttia, yksi palautettava

```
function [volume_box] = box( height, width, length )
```

Yksi argumentti, kaksi palautettavaa muuttujaa

```
function [area_circle,circumf] = circle( radius )
```

Ei nimettyä ulostuloa

```
function  = sqplot( side )
```

Hakujärjestys

Kun Matlab kohtaa nimen, esim `kala` etsitään tälle funktiolle/skriptille/muuttujalle toteutusta seuraavassa järjestyksessä:

1. Onko `kala` määritelty muuttuja?
2. Onko `kala` Matlabin sisäinen funktio?
3. Onko tämän hetkisessä hakemistossa tiedosto `kala.m`? Jos on niin ajetaan tiedostossa määritelty funktio.
4. Onko tiedosto `kala.m` jossain hakulistassa olevassa hakemistossa?

Hakulistasta lisää tietoa, ks. `path` ja `addpath`.

Globaalit muuttujat

Toisinaan on tarpeen funktioille käsitellä muuttujia muuten kuin argumenttien kautta. Tämä voidaan toteuttaa määrittelemällä tällaiset muuttujat globaaleiksi. Globaalit muuttujat sijaitsevat omassa työmuistissaan.

Muuttujat a, b ja x määritellään globaaleiksi syntaksilla

```
global a b x
```

Näihin muuttujiin pääsee käsiksi missä tahansa kontekstissa määrittelemällä ne globaaleiksi. Esim funktiotiedosto `hauki.m`:

```
function y = hauki( t )
  global k
  y = sin(k*t);
end
```

Kutsutaan `hauki`-funktiota komentoriviltä:

```
global k;
k = 10;
x = 0:0.01:1;
y = hauki(x);
```

Funktiokahvat

Toisinaan on tarpeen välittää funktio toiselle funktiolle. Siis mahdollistaa esimerkiksi sen että jonkun muun tekemä funktio kutsuu minun tekemääni funktiota. Käyttämällä `@`-merkkiä voidaan muodostaa *kahva* mihin tahansa funktioon.

Esim `f1.m`:

```
function y = f1(x)
  y = x + 2*sin(x);
end
```

ja `calc.m`:

```
function y = calc(h)
  x = 0:0.1:10;
  y = ave(h(x));
end
```

joita kutsutaan komentoriviltä:

```
calc(@f1)
```

Todellinen esimerkki funktiokahvoista

Työkalu funktion nollakohtien etsimiseen: `fzero(@function,x0)`, missä `x0` on alkuarvaus nollakohdan sijainnista. Esimerkiksi:

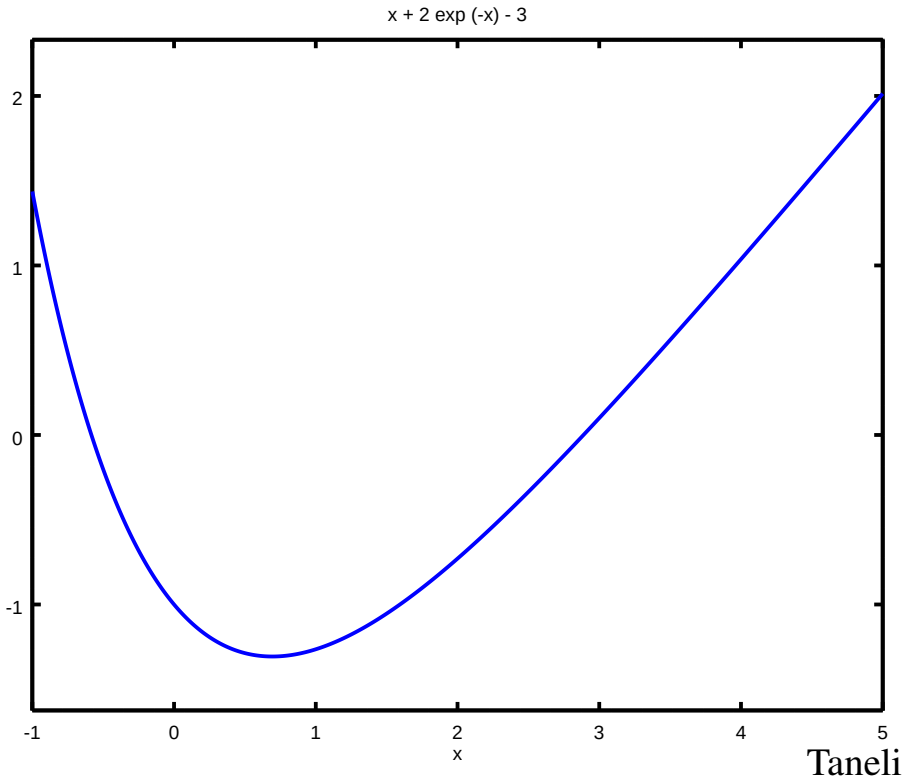
```
>> fzero(@cos,2)
ans = 1.5708
```

Etsitään funktion $f(x) = x + 2e^{-x} - 3$ nollakohdat käyttäen plotteria ja `fzero`-funktiota. Määritellään funktio

```
f = @(x) x + 2*exp(-x) - 3
```

Käytetään "helppoa" plotteria

```
ezplot(f,[-1,5])
```



Todellinen esimerkki funktiokahvoista

Ratkaistaan nollakohdat plotterin avulla valittuja alkuarvauksia käyttäen

```
>> fzero(f,-0.5)
ans = -0.58307
>> fzero(f,3)
ans =  2.8887
```

Vastaavasti on olemassa työkalu funktion minimikohtien etsimiseen:
`fminunc(@function,x0)`, missä `x0` on alkuarvaus minimin sijainnista.
Käytetään yllä määriteltyä funktiota $f(x)$:

```
>> fminunc(f,1)
ans = 0.69315
```

Ohjelmointi

Käyttäjän syöte

Ohjelmalle voidaan antaa syötettä `input`-komennolla:

```
x = input('Anna luku: ');
```

- Input-komennolle annettu syöte tulkataan Matlabin syntaksin mukaisesti
- Voi olla laskutoimituksia, hyödyntää muuttujia, jne.

Jos halutaan syöttää merkkijono:

```
x = input('Anna merkkijono: ', 's');
```

Tulostus

Yksinkertaisin tulostuskomento on `disp`. Sillä voidaan tulostaa yksittäinen skalaarin tai matriisin arvo ilman, että muuttujan nimeä tulostetaan. Esim:

```
>> disp(pi)
  3.1416
>> A=[1 2]
>> disp(A)
  [1 2]
>> disp('pi')
  pi
```

Tulostus

Usein halutaan tietyllä tavalla muotoiltua tulostusta, esim. tekstiä ja numeroita samalle riville. Tällöin käytetään `fprintf`-funktiota. Esim:

```
>> fprintf('Pii on %f, Neperin luku on %f\n', pi, exp(1) )
Pii on 3.141593, Neperin luku on 2.718282
```

`fprintf`-funktion ensimmäinen argumentti on merkkijono (formaatti) joka määrittelee muotoilun. Tämän jälkeen määritellään tulostettavat muuttujat samassa järjestyksessä kuin ne esiintyvät merkkijonossa.

Erikoismerkit:

`\b`	Backspace	`''`	Single quotation mark
`\f`	Form feed	`%%`	Percent character
`\n`	New line	`\\`	Backslash
`\r`	Carriage return	`\xN`	Hexadecimal number N
`\t`	Horizontal tab	`\N`	Octal number N

Tulostus

Formaatti:

`%d,%i`	Kokonaisluku, 10-kanta
`%o`	Kokonaisluku, 8-kanta
`%x`	Kokonaisluku, 16-kanta, luvut 0-9,a-f
`%X`	Kokonaisluku, 16-kanta, luvut 0-9,A-X
`%f`	Liukuluku
`%e`	Liukuluku, eksponentiaalinen notaatio, kuten 1.23456e+00
`%E`	Liukuluku, eksponentiaalinen notaatio, kuten 1.23456E+00
`%g`	Lyhyempi muodoista `%f` tai `%e`
`%G`	Lyhyempi muodoista `%f` tai `%E`
`%bx,%bX,%bo,%bu`	Binaarimuotoinen double-liukulukuesitys
`%tx,%tX,%to,%tu`	Binaarimuotoinen single-liukulukuesitys
`%c`	Merkki
`%s`	Merkkijono

Tulostus

Formaatin parametrit:

```
%I$FW.Pf
```

I$ = järjestys indeksi

Kuinka mones arvo tulostetaan tähän sijaintiin. Esim: `'%3$s %2$s %1$s %2$s'` tulostaa `'C B A B'` kun argumentit ovat `'A', 'B', 'C'`.

F = lippuja (flags)

`'-'`	Sisennys vasemmalle
`'+'`	Tulosta + positiivisille luvuille
`' '`	Täytä välilyönneillä kentän leveyteen
`'0'`	Täytä nollilla kentän leveyteen
`'#'`	Muokattu tulostus
	`%o`, `%x` tai `%X`: tulosta `0`, `0x` tai `0X` etuliite
	`%f`, `%e` tai `%E`: tulosta desimaalipiste aina
	`%g` tai `%G`: älä poista nollia perästä tai desimaalipistettä

Tulostus

Formaatin parametrit:

```
%I$FW.Pf
```

W$ = kentän leveys

Määrittää pienimmän määrän merkkejä jota tulostetaan.

Voi olla `'*'`, jolloin arvo luetaan argumenteista

P = tarkkuus `%f`, `%e` tai `%E`: Numeroiden määrä desimaalipisteen jälkeen.

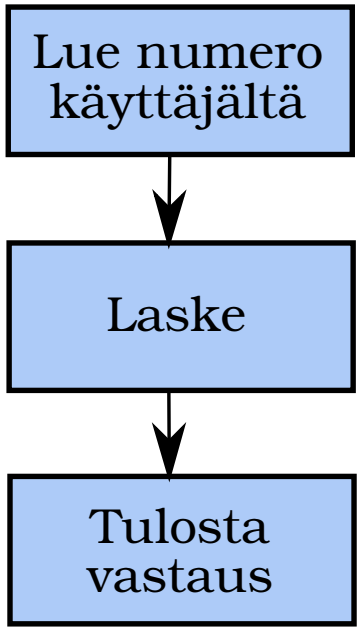
`%g` tai `%G`: Merkitsevien numeroiden määrä.

Voi olla `'*'` kuten yllä.

Suorituksen ohjaus

- Toistaiseksi kaikki skriptit ovat olleet rakenteeltaan suoraviivaisia
- Tietokone suorittaa rivin kerrallaan järjestyksessä ylhäältä alas.
- Ohjelmakoodi voidaan esittää suoraviivaisena vuokaaviona:

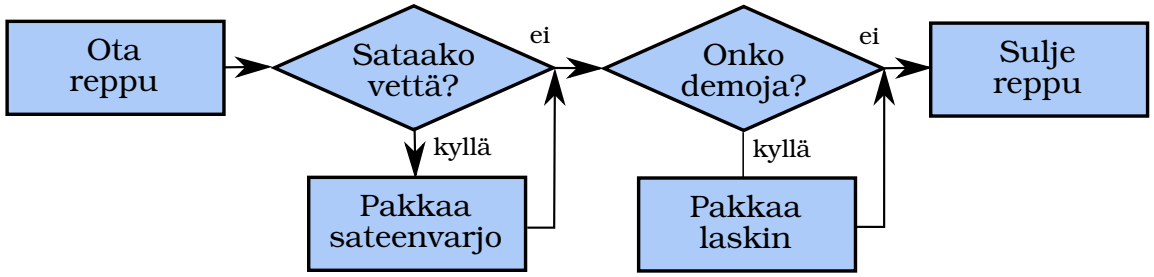
```
x = input('Anna numero: ');

y = 3*x^2-1;

fprintf('3*x^2-1=%f\n',y);
```



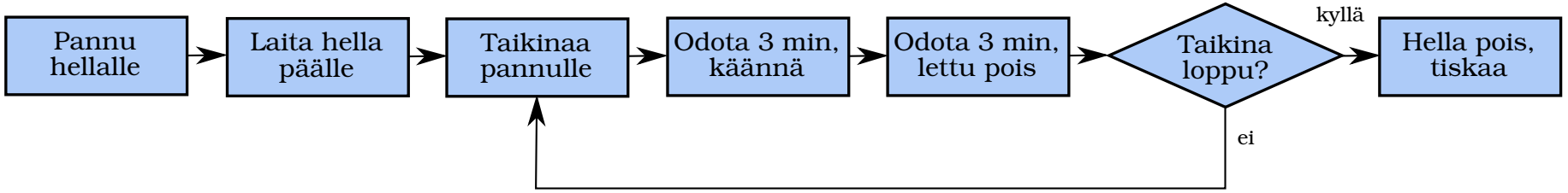
Suorituksen ohjaus

- Mitä jos halutaan mahdollistaa vaihtoehtoisia polkuja tai toistoja?
- Tätä varten on olemassa ehto- ja silmukkarakenteet

Ohjelma repun pakkaamiseen:



Ohjelma lättyjen paistamiseen:



Ehtolauseet

Yksinkertaisin ehtolause on yksittäinen if-lause:

```
if ehto
  lauseet
end
```

Esimerkiksi:

```
x = input('Anna luku: ');
if x == 1
  disp('Annoit ykkösen')
end
```

Ehtolauseet

Ehtolauseita on myös muita muotoja:

```
if ehto
  lauseet1
else ehto
  lauseet2
end
```

Esimerkiksi:

```
x = input('Anna luku: ');
if x == 1
  disp('Annoit ykkösen')
else
  disp('Annoit jotain muuta kuin ykkösen')
end
```

Ehtolauseet

If-lauseita voidaan ketjuttaa käyttämällä `elseif`-lausetta

```
if ehto
  lauseet1
elseif ehto
  lauseet2
else
  lauseet3
end
```

Esimerkiksi:

```
x = input('Anna luku: ');
if x < 0
  disp('Annoit negatiivisen luvun')
elseif x > 0
  disp('Annoit positiivisen luvun')
else
  disp('Annoit nollan')
end
```

Ehtolauseet

Ehtolauseita voidaan tietenkin laittaa myös sisäkkäin, esimerkiksi:

```
x = input('Anna luku: ');
if x > 0
  if mod(x,2) == 0
    disp('Annoit positiivisen, kahdella jaollisen luvun')
  end
end
```

Vastaavasti:

```
x = input('Anna luku: ');
if x > 0 && mod(x,2) == 0
  disp('Annoit positiivisen, kahdella jaollisen luvun')
end
```

Silmukkarakenteet

Yksinkertainen for-silmukka: Toistetaan lauseita indeksin `i` arvoille `i=x1, x1+1, x1+2, ..., x2-1, x2`:

```
for i=x1:x2
  lause
end
```

Esimerkiksi:

```
for i=0:10
  x = 2^i;
  fprintf('2 potenssiin %f on %f\n',i,x);
end
```

Silmukkarakenteet

Monimutkaisempia rakenteita voidaan tuottaa määrittelemällä askeleen pituus `s` käyttäen syntaksia `i=x1:s:x2`

- Askelkoko `s` voi olla myös negatiivinen tai desimaaliluku esim. `k=10:-2:4` tuottaa `k=10,8,6,4`
- Jos `s` on positiivinen, silmukkaa ei ajeta jos `x1>x2`
- Jos `s` on negatiivinen, silmukkaa ei ajeta jos `x1<x2`
- Jos `x1==x2`, silmukka ajetaan vain kerran
- Jos `s` ei ole kokonaisluku, pyöristysvirheet saattavat aiheuttaa sen, että silmukka ajetaan eri määrän kertoja kuin olet ajatellut. Usein on parempi käyttää kokonaislukuja. Esim:

```
for i=1:N
  x = x1+(x2-x1)*(i-1)/(N-1);
end
```

Silmukkarakenteet

Kaikkein alkeellisin/tehokkain silmukkarakenne on while-silmukka. Lauseita toistetaan niin kauan kuin ehto on tosi.

```
while ehto
  lause
end
```

Rakenteella voi tehdä esimerkiksi for-silmukkaa vastaavan rakenteen:

```
i = 1;
while i<=N
  lause
  i = i+1;
end
```

Myös ääretön silmukka on mahdollinen (Ctrl-C pysäyttää suorituksen):

```
while 1
  lause
end
```

Silmukkarakenteet

While-silmukka sopii tilanteisiin, jossa tarvittavien toistojen määrä ei ole tiedossa etukäteen. Esimerkiksi liukulukulaskennan tarkkuuden voi testata seuraavalla ohjelmalla:

```
x = 1;
eps = 1;
while x+eps ~= x
  eps = 0.5*eps;
end
eps
```

Silmukkarakenteet

Silmukkarakenteiden ohjausta varten on kaksi lisäkäskyä: `continue` ja `break`.

- `continue` palaa silmukan indeksin korottamiseen ja testiin ja jatkaa silmukan suorittamista alusta.
- `break` keskeyttää silmukan suorittamisen. Suoritus jatkuu silmukan `end`-lauseesta

Summataan positiiviset luvut taulukosta ja keskeytetään mikäli kohdataan nolla

```
x = [9 4 -2 4 0 5];
sum = 0;
for k = 1:length(x);
  if x(k) < 0
    continue;
  elseif x(k) == 0
    break;
  end
  sum = sum + x(k);
end
sum
```

Esimerkkejä silmukoiden käytöstä funktioissa

- Tehdään funktio, joka muodostaa vektoriin N ensimmäistä Fibonaccin lukua.
- Tehdään funktio, joka lukee vektorin A, palauttaa vektorit B ja C, missä B sisältää A:n ei-negatiiviset alkiot ja C sisältää A:n negatiiviset alkiot.
- Lasketaan likiarvo $\sqrt{2}$:lle Newton-Raphsonin menetelmällä ratkaisemalla yhtälö $f(x) = x^2 - 2 = 0$ iteratiivisesti

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

  Iteraatio lopetetaan kun x:n muutos eli $|x_{i+1} - x_i|$ on riittävän pientä.

Matriisimuotoisten ehtojen käyttö

Ehtolauseiden ja `while`-silmukkarakenteiden ehtolauseissa voi käyttää myös matriisi- tai vektorimuotoisia totuusarvoja. Ehtolauseke on tällöin tosi vain jos se on tosi kaikille alkioille.

```
x = [4,-9,25];
if x >= 0
  y = sqrt(x)
else
  disp('Osa alkioista on negatiivisia')
end
```

Huom! Operaattoreiden `&&` ja `&` sekä `||` ja `|` ero.

Switch-rakenne

`switch`-rakenne tarjoaa vaihtoehdon `if-elseif-else`-rakenteille tietyissä tapauksissa.

```
switch input % skalaari tai merkkijono
  case arvo1
    lauseet1
  case arvo1
    lauseet2
  .
  .
  .
  otherwise
    lauseetN
end
```

Switch-rakenne

Esimerkiksi

```
switch kulma
  case 90
    disp('suorakulma')
  case 180
    disp('oikokulma')
  otherwise
    disp('tuntematon kulma')
end
```

Tyylistä

Kannattaa opetella heti alusta lähtien koodin sisennystyyli. Vertaa:

```
function ret = func(x,y)
  for i=1:10
    if x > 0 && abs(y) > 2
      if sin(x+y) > 0
        ret = cos(x)
      end
    elseif y^4 < sin(x)
      x = x/2;
      z = y+cos(x);
      if z > 0
        ret = x^2+z^4;
      else
        continue;
      end
    end
  end
end
```

```
function ret = func(x,y)
for i=1:10
if x > 0 && abs(y) > 2
if sin(x+y) > 0
ret = cos(x)
end
elseif y^4 < sin(x)
x = x/2;
z = y+cos(x);
if z > 0
ret = x^2+z^4;
else
continue;
end
end
end
end
```

Lisää funktioista

Aliohjelmat

Monimutkaisempia ohjelmia ei ole järkevää kirjoittaa yhteen funktioon. Kannattaa pilkkoa ongelma pienempiin palasiin eli aliohjelmiin, joita kutsutaan pää/primäärifunktiosta. Aliohjelmat voivat sijaita useassa paikassa:

- Omassa funktiotiedostossa
  - Alifunktiota voi kutsua mistä vaan
- Alifunktiot samassa tiedostossa kuin pääfunktio (subfunctions)
  - Alifunktio kutsuttavissa vain tiedoston sisältä
- Sisäkkäiset alifunktiot (nested functions)
  - Alifunktio kutsuttavissa vain ulommasta funktiosta
  - Sisäfunktio näkee ulomman funktion muuttujat
- Omassa funktiotiedostossa `private`-hakemistossa
  - Käytetään suuremmissa ohjelmakirjastoissa

Alifunktiot samassa tiedostossa

Esimerkki `main.m`:

```
function main
  x = 5;
  y = subf(x)
end

function ret = subf(x)
  ret = x*x;
end
```

Funktio `main` on kutsuttavissa ulkopuolelta, `subf` ei.

Sisäkkäiset alifunktiot

Esimerkki `main1.m`:

```
function main1
  x = 5
  nestfun1

  function nestfun1
    x = x + 1
  end

end
```

Esimerkki `main2.m`:

```
function main2
  nestfun2

  function nestfun2
    x = 5
  end

  x = x + 1
end
```

Funktion lopettaminen

Aikaisemmin funktiot loppuivat aina funktion `end`-lauseeseen:

```
function ret = func(x)
.
.
end
```

Funktiosta voi *palautua* kutsuvaan funktioon myös kesken funktion `return` -käskyllä. Hyödyllistä esim. iteraation lopettamiseen:

```
function ret = mand(c)
  z = 0;
  for i=1:1000
    z = z^2 + c;
    if abs(z) > 2
      ret = i;
      return
    end
  end
  .
  .
```

Funktion lopettaminen

Toinen yleinen syy funktion kesken lopettamiseen on virhetilanne. Tällöin on syytä keskeyttää koko ohjelma. Tämä tapahtuu `error`-käskyllä. Esim:

```
function ret = func(x)
  if ~isscalar(x)
    error('Syöte ei ole skalaari')
  end
  .
  .
```

Monimutkaisempia virheilmoituksia voi tuottaa käyttämällä `fprintf`-syntaksia error-funktiossa:

```
function ret = func(x)
  if ~isscalar(x)
    error('Syöte on %s, pitäisi olla skalaari', class(x))
  end
  .
  .
```

Olemassa on myös vastaava komento `warning` varoituksien tuottamiseen.

Funktioiden argumenteista

Matlab ei automaattisesti tarkista funktiolle annettujen argumenttien määrää.

- Virhe tapahtuu kun määrittelemätöntä muuttujaa yritetään käyttää.
- Ylimääräisistä argumenteista ei haittaa.

Funktio voi kysyä sille annettujen argumenttien määrän komennolla `nargin`.
Esim:

```
function c = addme(a,b)

switch nargin
  case 2
      c = a + b;
  case 1
      c = a + a;
  otherwise
      c = 0;
end
```

Funktioiden palautusarvoista

Vastaavasti funktion toiminta voi olla riippuvainen palautusarvojen lukumäärästä:

```
function [b,c,d] = powers(a)

if nargout > 2
  d = a^3
end
if nargout > 1
  c = a^2
end
if nargout > 0
  b = a^1
end

end
```

Ei kuitenkaan haittaa vaikka `c` ja `d` määritellään vaikka kutsuva funktio ei niitä tarvitse. Usein funktion toimintaa voidaan kuitenkin tehostaa.

Virheiden korjaaminen = debuggaaminen

Useimmiten virheet ovat kahta eri tyyppiä:

1. Syntaksivirheet. Esim. sulkeiden tai pilkun puuttuminen, komennon nimen väärin kirjoittaminen. Matlab antaa virheilmoituksen ja osoittaa virheen sijainnin. Useimmiten helppoja korjata.
2. Algoritmivirheet. Ohjelma pystytään suorittamaan mutta tulos on humpuukkia. Vaikeampia korjata.

Korjaaminen:

1. Yksinkertaista ohjelmaasi
2. Vertaa käsin laskettuun
3. Näytä välitulokset (poista puolipisteitä)
4. Käytä Matlab-editorin debuggeriominaisuuksia

Liikeyhtälöiden iterointi Eulerin menetelmällä

Johdetaan Eulerin menetelmä liikeyhtälöille

Kinematiikasta tiedämme

$$v(t) = \frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}$$
$$a(t) = \frac{dv}{dt} = \lim_{\Delta t \to 0} \frac{v(t+\Delta t) - v(t)}{\Delta t}.$$

Korvataan nyt Δt äärellisellä aika-askeleen pituudella

$$v(t) \approx \frac{x(t+\Delta t) - x(t)}{\Delta t}$$
$$a(t) \approx \frac{v(t+\Delta t) - v(t)}{\Delta t}$$

ja käytetään ajalle alaindeksiä n, jolloin saadaan vastaavasti

$$v_n \approx \frac{x_{n+1} - x_n}{\Delta t}$$
$$a_n \approx \frac{v_{n+1} - v_n}{\Delta t}.$$

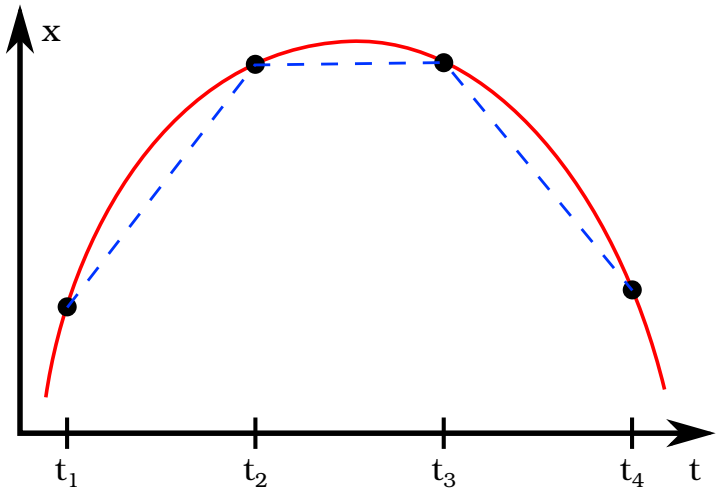
Ratkaistaan v_{n+1} ja x_{n+1}, jolloin saadaan **Eulerin menetelmä liikeyhtälöiden iterointiin:**

$$x_{n+1} = x_n + v_n \Delta t$$
$$v_{n+1} = v_n + a_n \Delta t$$

Euler-Cromer

Olisimme voineet valita yhtälailla erotusosamäärän taaksepäin

$$v(t) = \frac{dx}{dt} = \frac{x(t) - x(t - \Delta t)}{\Delta t}$$
$$a(t) = \frac{dv}{dt} = \frac{v(t) - v(t - \Delta t)}{\Delta t},$$



jolloin saisimme iteraatiomenetelmän:

$$x_{n+1} = x_n + v_{n+1}\Delta t$$
$$v_{n+1} = v_n + a_{n+1}\Delta t$$

Tätä muotoa on kuitenkin hankala käyttää.

Parhaaksi vaihtoehdoksi liikeyhtälöiden iterointiin osoittautuu niin sanottu **Euler-Cromer -menetelmä:**

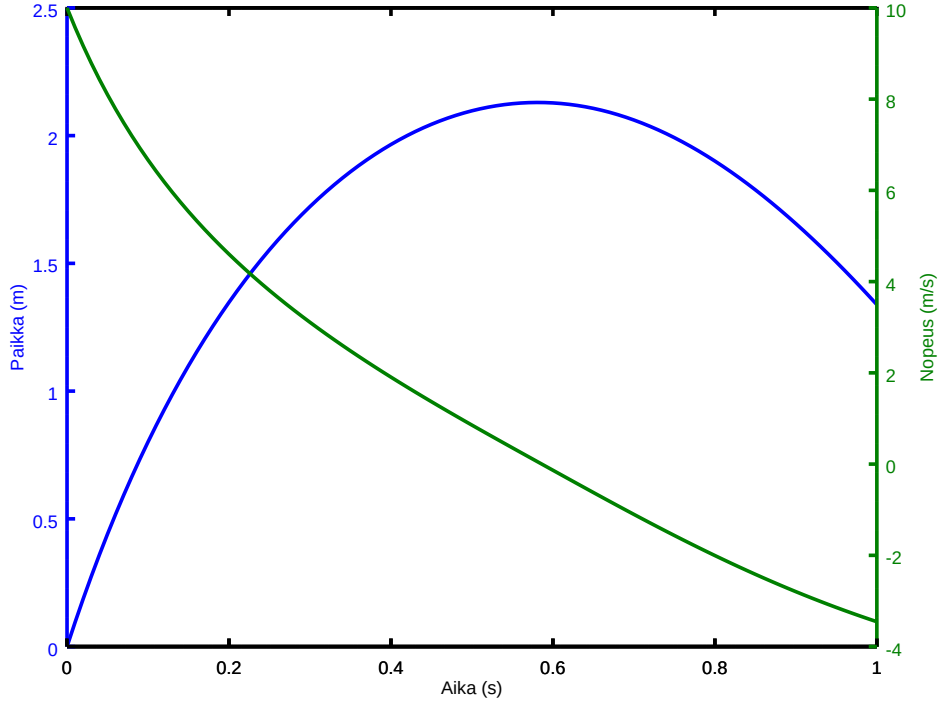
$$v_{n+1} = v_n + a_n\Delta t$$
$$x_{n+1} = x_n + v_{n+1}\Delta t$$

Esimerkki: 1-ulotteinen heittoliike ilmanvastuksella

Heitetään pallo ylös nopeudella 10 m/s. Palloon vaikuttaa gravitaatiovoima ja ilmanvastus:

$$F = -mg - \text{sign}(v)\beta v^2$$

Ratkaistaan iteratiivisesti Euler-Cromer menetelmällä yhden sekunnin ajan:




```
v(n+1) = v(n) + (-g-sign(v(n))*beta*v(n)^2/m)*dt
x(n+1) = x(n) + v(n+1)*dt
```

Esimerkkikoodi: `euler_heittoliike1d.m`

Esimerkki: heiluri

Yksinkertainen heiluri:

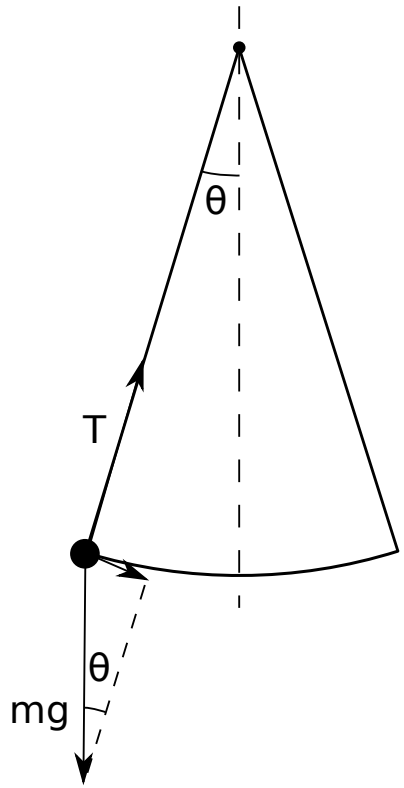
$$mL\frac{d^2\theta}{dt^2} = -mg\sin\theta$$

Ratkaistaan usein analyyttisesti käyttäen pienen kulman approksimaatiota $\sin\theta \approx \theta$, jolloin saadaan lineaarinen DY:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\theta$$

Tällöin heilurin jakson aika

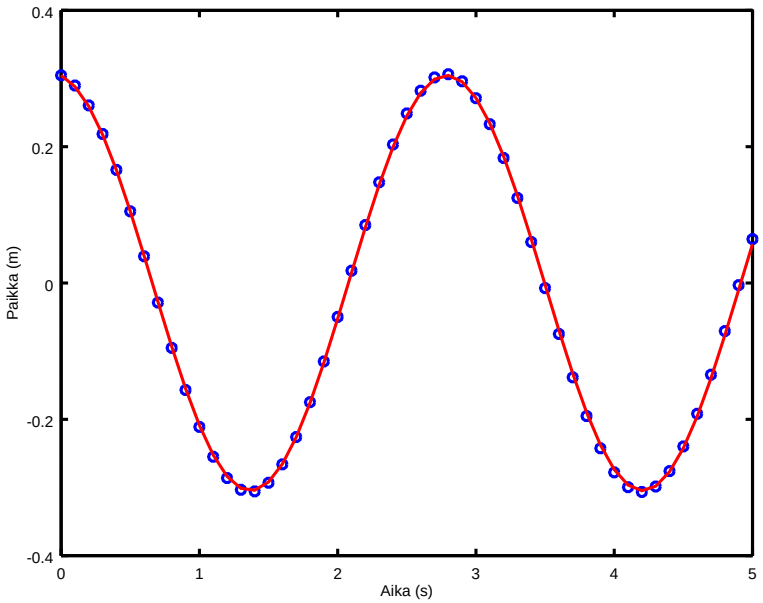
$$T = 2\pi\sqrt{\frac{L}{g}}$$

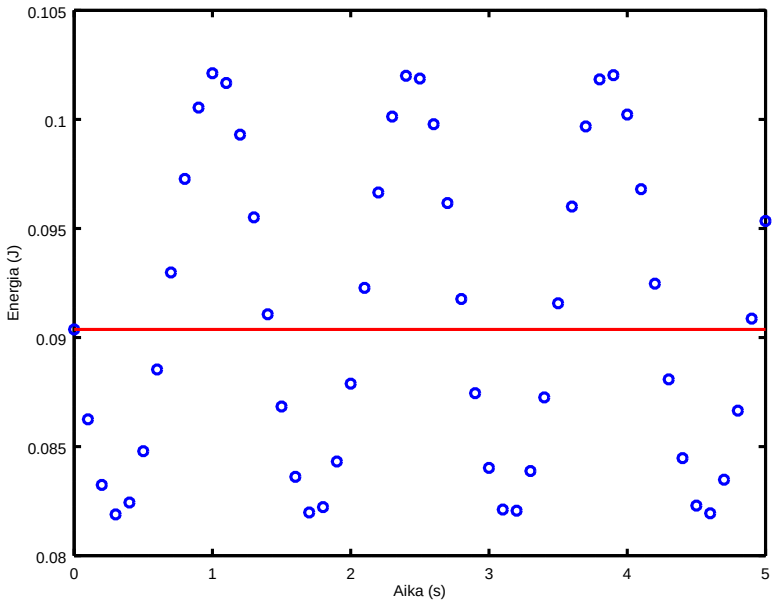


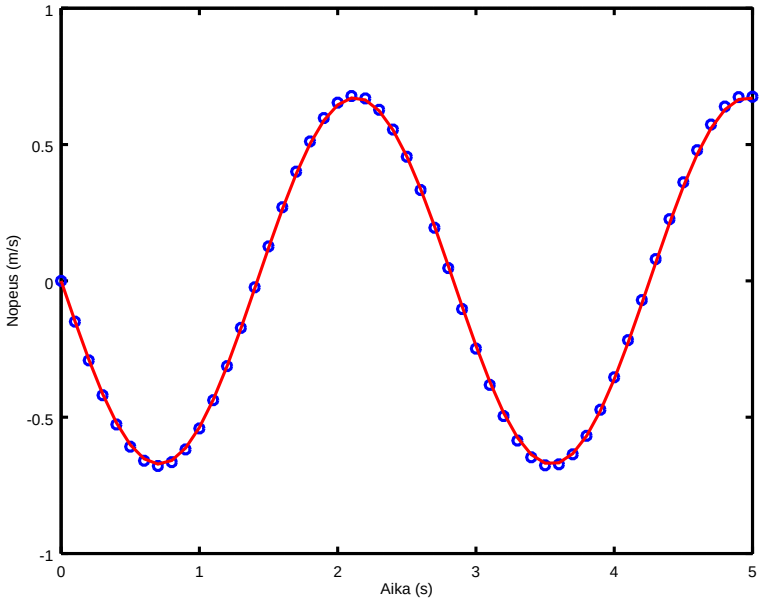
Sekä lineaarinen että epälineaarinen DY voidaan ratkaista numeerisesti.

Esimerkkikoodi: `euler_heiluri.m`.

Heiluri: Euler-Cromer







Heiluri: Euler