Ohjeita Matlabin käyttöön

Kokoelma @CSC- ja SuperMenu-lehdissä
ilmestyneitä artikkeleita

Juha Haataja (toim.)

E-mail: Juha.Haataja@csc.fi

CSC – Tieteellinen laskenta Oy

Versio 1.01 (9.6.2000)

Saatteeksi

Tämä kirjanen on tarkoitettu johdatukseksi Matlab-ohjelmiston käyttöön. Opas ei sisällä kattavaa kuvausta mistään aihepiiristä, vaan sen tarkoitus on auttaa uutta käyttäjää alkuun ja ohjata hänet yksityiskohtaisemman tiedon lähteille. Tähän kirjaseen kootut artikkelit käsittelevät Matlabin käytön perusteita sekä mm. grafiikkaa, tilastollista analyysiä ja numeerisia menetelmiä.

Opas löytyy PDF-muodossa www-osoitteesta `http://www.csc.fi/oppaat/matlab/`.

Sisältö

Lisätietoja

Matlabin käytöstä saa lisätietoja CSC:n koneiden komennolla `help matlab` tai www-osoitteesta `http://www.csc.fi/cschelp/sovellukset/math/matlab/`

Tekijänoikeudet

Johdatus Matlab 5.x:n käyttöön

Juha Haataja
Juha.Haataja@csc.fi

Matlab (tulee sanoista *matrix laboratory*) on erityisesti numeerisen lineaarialgebran sovelluksiin kehitetty työskentely-ympäristö. Matlabilla on mahdollista mm. ratkaista lineaarisia yhtälöryhmiä tai ominaisarvotehtäviä, laskea hajotelmia matriiseille sekä ratkoa vaikkapa differentiaaliyhtälöitä. Tämän lisäksi Matlabissa on monipuoliset grafiikkaominaisuudet, joten sitä voi käyttää myös visualisointiin. Matlabiin voi myös ohjelmoida uusia toimintoja, joten ohjelmistoa voi kehittää omia käyttötarpeita vastaavaksi.

Matlabin hyviä puolia ovat käytön interaktiivisuus ja nopeus sekä mitä moninaisimpiin tarkoituksiin saatavat työkalupakit (toolbox). Ongelmana on tehottomuus suurissa tehtävissä sekä vaatimattomat mahdollisuudet symboliseen laskentaan — näihin tarkoituksiin kannattaa käyttää muita välineitä.

Matlabin käytön perusteet

Matlab käynnistyy komennolla `matlab` ja ohjelmasta poistutaan komennolla `quit`. Unix-komentoriviltä voi myös käynnistää Matlabin avustusjärjestelmän komennolla `matlabdoc`. Manuaalisivuihin pääsee käsiksi komennolla `matlabdoc -man`.

Matlabin sisällä pääsee avustusjärjestelmään komennolla `help`. Funktiosta *fn* saa tietoja Matlabin komennolla `help` *fn*. Lisätietoja Matlabin käytöstä saa myös komennoilla `helpwin` ja `helpdesk`. Esimerkkejä Matlabin käytöstä saa komennolla `demo`.

Matlabin perustietorakenne on *n*-ulotteinen taulukko, jonka erikoistapauksia ovat matriisit, vektorit ja skalaarit. Lisäksi Matlabissa voi luoda myös rakenteisia tyyppejä, joiden alkiot eivät ole välttämättä samaa tyyppiä.

Matlab käyttää merkkijonoa ”`>> `” kehotteena, kun se odottaa käyttäjän komentoa:

```
>> x = 3
x =
     3
>> a = [1; 2]
a =
     1
     2
>> b = [1 2 3; 6 4 5; 9 8 7]
b =
     1     2     3
     6     4     5
     9     8     7
```

Symboli `=` tarkoittaa sijoitusoperaatiota. Edellä sijoitimme muuttujan `x` arvoksi skalaarin eli luvun 3. Vektoreita ja matriiseja muodostetaan käyttämällä hakasulkuja `[···]`. Siten muuttujan `a` arvoksi tuli kolmialkioinen pystyvektori. Taulukon alkiot voi erottaa toisistaan välilyönneillä tai pilkulla. Käytimme edellä puolipistettä `;` erottamaan matriisin `b` rivit toisistaan. Myös rivinvaihtoa voi käyttää rivien syöttämisessä:

```
>> b = [1 2 3
6 4 5
9 8 7]
b =
     1     2     3
     6     4     5
     9     8     7
```

Muuttujien nimissä saa olla 19 kirjainta tai numeroa; nimen pitää alkaa kirjaimella. Matlab tekee eron isojen ja pienten kirjainten välillä. Komento `whos` tulostaa muuttujien nimet ja niiden tyypin ja koon:

```
>> whos
  Name    Size      Bytes  Class
  a       2x1          16  double array
  b       3x3          72  double array
  x       1x1           8  double array
Grand total is 12 elements using 96 bytes
```

Samalla komentorivillä voi antaa useita komentoja puolipisteellä tai pilkulla erotettuina. Puolipiste estää tulostuksen. Vertaa seuraavia komentoja:

```
>> c = 1; d = 2;
>> c = 1, d = 2
c =
     1
d =
     2
```

Matlabissa voi käsitellä myös merkkijonoja:

```
>> s1 = 'terve'; s2 = 'tuloa';
>> s3 = [s1 s2]
s3 =
tervetuloa
```

Taulukkojen käsittely

Voit luoda useampiulotteisia taulukkoja sijoituslauseen avulla tai käyttämällä funktiota `cat`:

```
>> A = cat(3, [1 2 3; 4 5 6], ...
[-1.5 -1 -0.5; -3 -2 -1]);
>> A(:,:,3) = [1 0 0; 0 1 0]
A(:,:,1) =
     1     2     3
     4     5     6
A(:,:,2) =
   -1.5000   -1.0000   -0.5000
   -3.0000   -2.0000   -1.0000
A(:,:,3) =
     1     0     0
     0     1     0
```

Tässä liitimme aluksi yhteen kaksi matriisia kolmiulotteiseksi taulukoksi dimension 3 suhteen komennolla `cat(3, [···], [···])`. Käytimme kolmea pistettä `...` jatkamaan komentoa seuraavalle riville ja puolipistettä estämään tulostuksen. Tämän jälkeen lisäsimme taulukkoon `A` kolmannen kaksiulotteisen taulukon. Kaksoispistettä voi käyttää poimimaan taulukosta haluttuja riviä ja sarakkeita:

```
>> A(1:2, 2:3, 1)
ans =
     2     3
     5     6
```

Komento `clear` poistaa muuttujia muistista:

```
>> clear c d x s1 s2 s3
>> whos
  Name      Size      Bytes  Class
  A         2x3x3       144  double array
  a         2x1          16  double array
  b         3x3          72  double array
Grand total is 29 elements using 232 bytes
```

Taulukosta voi poistaa tietyn osan käyttämällä tyhjää taulukkoa `[]` seuraavasti:

```
>> A(:,:,[2 3]) = []
A =
     1     2     3
     4     5     6
```

Suuret taulukot kannattaa luoda ennen niiden käyttöä esimerkiksi `zeros`-funktiolla, joka luo halutun kokoisen nollia sisältävän taulukon:

```
>> C = zeros(500,500);
```

Jos riittävän isoa taulukkoa ei ole olemassa, Matlab luo uuden riittävän ison taulukon ja kopioi vanhan taulukon sisällön uuteen. Uudet alkiot alustetaan nolliksi. Seuraavassa funktio `size` tulostaa taulukon koon.

```
>> C(501,501) = 1.0;
>> size(C)
ans =
   501   501
```

Siis sijoituslause `C(501,501) = 1.0;` sai aikaan 501×501 -kokoisen taulukon muodostamisen. Tämä on tietenkin tehotonta. Siis muista varata suurille taulukoille riittävästi tilaa etukäteen! Matlabissa voi käsitellä matriiseja myös harvassa talletusmuodossa, lisätietoja saa komennolla `help sparse`.

Matemaattiset funktiot

Matlab tuntee suuren joukon matemaattisia perusfunktioita:

```
>> c = [sin(pi/4) sqrt(2) exp(1) log(10)]
c =
    0.7071    1.4142    2.7183    2.3026
>> d = [exp(i*pi/4) sqrt(-1)]
d =
   0.7071 + 0.7071i         0 + 1.0000i
```

Tässä käytimme valmiiksi määriteltyjä vakioita `pi` ja `i` (imaginaariyksikkö, myös nimellä `j`). Näitä nimiä voi tosin käyttää myös omien muuttujien niminä, jolloin niiden alkuperäiset merkitykset katoavat. Matlabin matemaattisista perusfunktioista saa lisätietoa komennolla `help elfun` ja esimerkiksi `sin`-funktiosta komennolla `help sin`:

```
>> help sin
SIN    Sine.
  SIN(X) is the sine of the elements of X.
```

Vaikka Matlabin avustusteksteissä käytetään funktioiden ja komentojen nimissä isoja kirjaimia, ne täytyy antaa komentorivillä käyttäen pieniä kirjaimia!

Kuvaajien piirtäminen

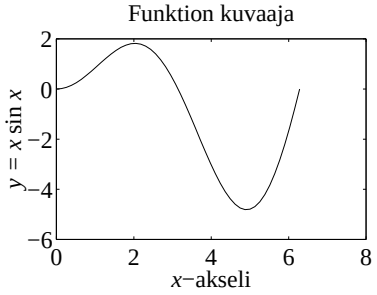
Matlabin vahvoja puolia on monipuolinen grafiikka. Kaikki tapahtuu kuitenkin numeerisesti; symbolista funktion esitystä voi käyttää vain rajallisesti. Siis funktion arvot täytyy ensin laskea numeeriseen vektoriin, jonka jälkeen kuvaaja on piirrettävissä. Apuna on hyvä käyttää `linspace`-funktiota, joka diskretoi lukuvälin haluttuun määrään pisteitä:

```
>> x = linspace(0, 2*pi, 50);
>> y = x.*sin(x);
>> plot(x, y)
```

Kuvaan voi lisätä otsikot ja akselien kuvaukset. Seuraavassa otamme käyttöön kallistetun kirjasintyypin käyttäen TEXiä muistuttavaa `\sl`-notaatiota:

```
>> xlabel('{\slx}-akseli')
>> ylabel('{\sly = x} sin {\slx}')
>> title('Funktion kuvaaja')
```

Tulos näyttää seuraavalta:



Kuvaajan saa tulostettua PostScript-muodossa tiedostoon `kuva.eps` komennolla

```
>> print -deps kuva.eps
```

Lisätietoja kuvaajien piirtämisesta ja tulostamisesta saa komennoilla `help plot` ja `help print`.

Lineaarialgebran operaatiot

Heittomerkin `'` avulla saamme matriisin tai vektorin kompleksikonjugoidun transpoosin. Operaatio `.'` pelkästään transponoi. Reaalimatriiseille operaatiot ovat tietenkin identtisiä.

```
>> i = sqrt(-1);
>> m = [1 i; 1-i 2]
m =
   1.0000                  0 + 1.0000i
   1.0000 - 1.0000i   2.0000
>> m'
ans =
   1.0000             1.0000 + 1.0000i
        0 - 1.0000i   2.0000
>> ans.'
ans =
   1.0000                  0 - 1.0000i
   1.0000 + 1.0000i   2.0000
```

Edellä käytimme `ans`-muuttujaa, johon Matlab sijoittaa komennon tuloksen, jos sitä ei talleteta sijoituslauseella.

Matlabin taulukkoja voi laskea yhteen ja vähentää alkioittain toisistaan käyttämällä operaatioita `+` ja `-`. Kertolaskuoperaatio `*` tarkoittaa matriisi- tai vektorikertolaskua, jolloin dimensioiden täytyy täsmätä.

```
>> clear
>> a = [1 2 3; -2 0 -1; 1 1 1];
>> b = [1 0 0; 0 0 1; 0 1 0];
>> c = a + b
c =
     2     2     3
    -2     0     0
     1     2     1
>> v = [1 4 9]';
>> u = a*v
u =
    36
   -11
    14
```

Operaatioiden `/` ja `\` avulla voi mm. ratkaista lineaarisia yhtälöryhmiä. Saamme ratkaistua vektorin `x` yhtälöryhmästä $\mathtt{a}\mathtt{x} = \mathtt{v}$ seuraavasti:

```
>> x = a\v
x =
    4.3333
   17.3333
  -12.6667
```

Jos yhtälöryhmässä on enemmän yhtälöitä kuin tuntemattomia, tuloksena on pienimmän neliösumman ratkaisu. Huomaa, että operaatio `a\b` tarkoittaa matemaattista lauseketta $\mathtt{a}^{-1}\mathtt{b}$ ja operaatio `a/b` lauseketta $\mathtt{a}\mathtt{b}^{-1}$.

Taulukko-operaatiot

Iso osa Matlabin operaatioista käsittelee kokonaisia taulukoita kerrallaan. Täten esimerkiksi `sin`-funktiolle voi antaa taulukkoargumentin. Seuraavassa on tästä esimerkki:

```
>> t = linspace(0, 1, 6)
t =
         0    0.2000    0.4000    0.6000
    0.8000    1.0000
>> u = t.^2 + 1
u =
    1.0000    1.0400    1.1600    1.3600
    1.6400    2.0000
>> x = linspace(-1, 1, 6)
x =
   -1.0000   -0.6000   -0.2000    0.2000
    0.6000    1.0000
>> y = exp(x)
y =
    0.3679    0.5488    0.8187    1.2214
    1.8221    2.7183
```

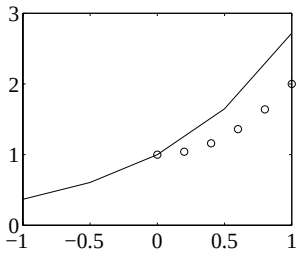
Edellä käytimme alkioittaista operaatiota `.^` korottamaan vektorin `t` jokaisen alkion toiseen potenssiin. Huomaa, että operaatio `^` tarkoittaa matriisieksponenttia, jolloin potenssiin korotettavan täytyy olla joko skalaari tai $n \times n$ -matriisi. Operaattoreilla `.*` ja `*` on vastaava ero: operaattori `.*` kertoo taulukkojen alkiot keskenään ja operaattori `*` tarkoittaa matriisikertolaskua. Samaa pätee operaattoreihin `./` ja `/`, joista ensimmäinen on alkioittainen jakolasku ja jälkimmäinen tarkoittaa käänteismatriisilla kertomista.

Käytimme sijoituslauseessa `u = t.^2 + 1` myös *skalaarin laajennusta*, sillä luku `1` lisätään jokaiseen operaation `t.^2` tuloksena saadun taulukon alkioon.

Saamme tulostettua edellä laskemamme arvot kuvaajaksi seuraavasti:

```
>> plot(t, u, 'o', x, y, '-')
```

Tuloksena on seuraava kuvaaja:



Funktioiden `zeros(n,m)` ja `ones(n,m)` avulla voi luoda halutun kokoisen nollia tai ykkösiä sisältävän taulukon. Funktio `eye(n)` luo puolestaan halutun kokoisen identiteettimatriisin. Funktio `diag(v)` luo lävistäjämatriisin vektorista `v`.

Funktioiden `max`, `min` ja `abs` avulla saa taulukon alkioittaisin maksimin, minimin ja itseisarvon. Funktio `sum` laskee taulukon alkioiden summan ja funktio `prod` tulon. Funktio `size` tulostaa taulukon koon dimensioittain ja funktio `length` alkioiden lukumäärän.

Sekalaisia operaatioita

Komennolla `save` *tiedosto* saat talletettua Matlab-istunnossa määritellyt muuttujat annetun nimiseen tiedostoon. Komennolla `load` *tiedosto* saat muuttujat vastaavasti luettua takaisin. Seuraavassa `rand`-funktio luo halutun kokoisen satunnaistaulukon.

```
>> A = rand(10,10);
>> b = rand(10,1);
>> c = A*b;
>> whos
  Name    Size      Bytes  Class
  A      10x10        800  double array
  b      10x1          80  double array
  c      10x1          80  double array
Grand total is 120 elements using 960 bytes
>> save tila; clear
```

Siis talletimme muuttujat Matlabin omassa talletusmuodossa tiedostoon `tila.mat` ja poistimme kaikki muuttujat muistista. Seuraavassa luemme talletetut tiedot takaisin:

```
>> whos
>> load tila; whos
  Name    Size      Bytes  Class
  A      10x10        800  double array
  b      10x1          80  double array
  c      10x1          80  double array
Grand total is 120 elements using 960 bytes
```

Komennon `format` avulla voi muuttaa Matlabin tulostustapaa. Tässä pikaohjeessa on käytetty asetusta `format compact`, joka käyttää tiiviimpää esitystapaa. Muut mahdollisuudet saat selville komennolla `help format`.

Rakenteiset tyypit

Matlabissa voi määritellä normaalien taulukkojen lisäksi myös *solutaulukkoja* käyttäen apuna kaarisulkuja {···}:

```
>> fun{1} = 6
fun =
    [6]
>> fun{2} = [-2; 3]
fun =
    [6]    [1x2 double]
>> fun{3} = [2 0; 0 120]
fun =
    [6]    [1x2 double]    [2x2 double]
>> whos fun
  Name Size Bytes Class
  fun 1x3 416 cell array
Grand total is 10 elements using 416 bytes
```

Edellä sijoitimme solutaulukon `fun` ensimmäiseksi alkioksi luvun 6, toiseksi alkioksi kahden alkion pystyvektorin ja kolmanneksi alkioksi 2×2 -matriisin. Solutaulukon alkioihin viitataan kaarisulkujen avulla:

```
>> s = - fun{3}\fun{2}
s =
    1.0000
   -0.0250
```

Solutaulukon voi muodostaa suoraan käyttämällä kaarisulkuja alustimena:

```
>> t = {'funktio' fun}
t =
    'funktio'    1x3 cell
```

Tässä solutaulukko `t` koostuu merkkijonosta ja toisesta solutaulukosta. Sisäkkäisten solutaulukkojen alkion voi valita käyttämällä kaarisulkuja tarpeeksi monta kertaa:

```
>> t{2}{2}
ans =
    -2
     3
```

Solutaulukkoja indeksoidaan positiivisilla kokonaisluvuilla, mutta aina tämä ei ole sopiva tietorakenne. Siksi Matlabissa voi myös käyttää *rakennetaulukkoja*, joiden alkioille voi antaa kuvaavat nimet käyttäen pistenotaatiota ja *nimettyjä kenttiä*:

```
>> clear fun s
>> fun.arvo = 6;
>> fun.grad = [-2; 3];
>> fun.hesse = [2 0; 0 120]
fun =
     arvo: 6
     grad: [2x1 double]
    hesse: [2x2 double]
>> whos fun
  Name    Size      Bytes  Class
  fun     1x1         512  struct array
Grand total is 10 elements using 512 bytes
```

```
>> s = - fun.hesse\fun.grad
s =
    1.0000
   -0.0250
```

Rakennetaulukon kenttien nimet saa tulostettua komennolla `fieldnames`:

```
>> fieldnames(fun)
ans =
    'arvo'
    'grad'
    'hesse'
```

Rakennetaulukon alkiot saa kerättyä tavalliseen taulukkoon käyttämällä hakasulkuja:

```
>> arvot = [fun.grad fun.hesse]
arvot =
   -2.0000    2.0000         0
    3.0000         0       120
```

Sama toimii myös solutaulukkojen kanssa:

```
>> v = {[2 3]; [4; 5]}
v =
    [1x2 double]
    [2x1 double]
>> [v{1} v{2}']
ans =
     2     3     4     5
```

Tässä transponoimme pystyvektorin `v{2}`, jotta alkioiden liittäminen vaakavektoriksi onnistuu.

Työskentely Unix-ympäristössä

Unix-ympäristössä voi Matlabin komentoriveillä siirtyä kursorinäppäimillä ← ja → sekä `emacs`-tyylisillä komennoilla `^B` ja `^F`. Aikaisemmin annettuja komentoja saa näkyviin kursorinäppäimillä ↓ ja ↑ sekä vastaavilla `emacs`-tyylisillä komennoilla `^N` ja `^P`. Lisäksi merkkejä voi poistaa näppäimillä Backspace ja Del.

Näppäin `^C` keskeyttää käynnissä olevan laskennan ja näppäimellä `^Z` Matlab pysähtyy ja päästään komentotulkkiin. Komennolla `fg` pääsee komentotulkista takaisin pysäytettyyn ohjelmaan ja Unix-komennolla `jobs` näkee taustalla olevat työt.

Huutomerkin avulla voi Matlabissa antaa Unix-komentoja. Jos ei ole käytettävissä X-näyttöä, jolla voi pitää yhtä aikaa useita ikkunoita auki, voi ohjelmatiedostoja käydä editoimassa esimerkiksi Matlabin komennolla `!emacs` *tiedosto*.

Lisätietoa

Tässä jutussa olemme käyneet läpi vain Matlabin käytön alkeita. Matlabista ja muista matemaattisista ohjelmistoista kerrotaan CSC:n oppaassa *Matemaattiset ohjelmistot* [Haa98a]. Aiheesta on kirjoitettu myös @CSC-lehden aiemmissa artikkeleissa [Haa98b, Haa98c, Haa99, HS98, Lam97, Sav98a, Sav98b]. Numeerisista menetelmistä saa tietoa CSC:n oppaista [Haa95, HJ94, HHL+99].

Lisätietoja Matlabista saat käsikirjoista [Mat96, Mat97] sekä MathWorks-yhtiön www-sivuilta osoitteessa `http://www.mathworks.com`. Sivuilta löytyy myös käyttäjien kirjoittamia Matlab-ohjelmakoodeja.

CSC:n asiantuntijoilta voi kysellä tietoa Matlabista. Sähköpostiosoitteet ovat `Esa.Lammi@csc.fi` ja `Ville.Savolainen@csc.fi`.

Kirjallisuutta

[Haa95] Juha Haataja. *Optimointitehtävien ratkaiseminen*. CSC, 1995.

[Haa98a] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[Haa98b] Juha Haataja, Matlab 5.x — kysymyksiä ja vastauksia, *@CSC*, 2/1998.

[Haa98c] Juha Haataja, Optimointitehtävien ratkaisu Matlabilla, *@CSC*, 3/1998.

[Haa99] Juha Haataja, Geneettiset algoritmit ja Matlab, *@CSC*, 3/1999.

[HHL+99] Juha Haataja, Jussi Heikonen, Yrjö Leino, Jussi Rahola, Juha Ruokolainen ja Ville Savolainen. *Numeeriset menetelmät käytännössä*. CSC, 1999.

[HJ94] Jari Hämäläinen ja Jari Järvinen. *Elementtimenetelmä virtauslaskennassa*. CSC, 1994.

[HS98] Juha Haataja ja Ville Savolainen, Matlab 5.x — lisää kysymyksiä ja vastauksia, *@CSC*, 3/1998.

[Lam97] Esa Lammi, Matlabin uusi versio 5.0, *@CSC*, 3/1997.

[Mat96] The MathWorks, Inc. *Using Matlab, Version 5*, 1996.

[Mat97] The MathWorks, Inc. *Getting Started with Matlab, Version 5.1*, 1997.

[Sav98a] Ville Savolainen, Matlab 5.x — uusia kysymyksiä ja vastauksia, *@CSC*, 5/1998.

[Sav98b] Ville Savolainen, Tehokas ohjelmointi Matlabilla, *@CSC*, 1/1998.

Matlab 5.x ja grafiikka

Juha Haataja
Juha.Haataja@csc.fi

Matlab-ohjelmisto on interaktiiviseen numeeriseen laskentaan kehitetty työskentely-ympäristö. Numeeristen ominaisuuksien lisäksi Matlabissa on monipuoliset grafiikkaominaisuudet, joten sitä voi käyttää myös visualisointiin. Kerron seuraavassa johdatuksenomaisesti graafisten kuvaajien ja esitysten tuottamisesta Matlabilla.

Funktion kuvaajan piirtäminen

Komento	*Selitys*
`plot`	Tavallinen kuvaaja
`loglog`	Logaritmiset asteikot
`semilogx`	Logaritminen x-asteikko
`semilogy`	Logaritminen y-asteikko
`polar`	Napakoordinaattiesitys
`plotyy`	Vasemmalla ja oikealla y-akselit

Luomme aluksi x-arvoja sisältävän vektorin `x` komennolla `linspace`:

```
>> x = linspace(0, 2, 5)
x =
   0  0.5000  1.0000  1.5000  2.0000
```

Seuraavaksi laskemme funktion $f(x) = \sin(x)e^x$ arvot vektorin `x` sisältämissä pisteissä:

```
>> y = sin(x) .* exp(x)
y =
   0  0.7904  2.2874  4.4705  6.7188
```

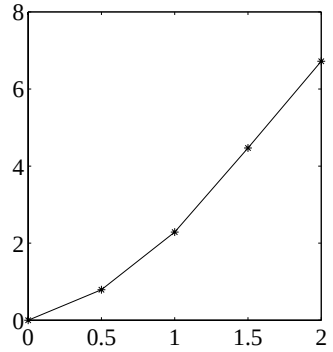
Huomaa, että laskutoimituksessa käytettiin alkioittaista kertolaskua `.*`, jossa lausekkeen `sin(x)` tuloksena saadun vektorin arvot kerrottiin alkioittain lausekkeen `exp(x)` tulosvektorin kanssa. Saman voi tehdä tietenkin (tehottomammin) myös silmukkarakenteella:

```
>> y = zeros(1,length(x));
>> for i = 1:length(x)
y(i) = sin(x(i)) * exp(x(i));
end
>> y
y =
   0  0.7904  2.2874  4.4705  6.7188
```

Seuraavaksi piirrämme funktion kuvaajan komennolla `plot`:

```
>> plot(x, y, '*-')
```

Tässä kuvapisteet merkitään symbolilla `*` ja lisäksi pisteet yhdistetään viivoilla. Tulos voisi näyttää seuraavalta:



Useamman funktion kuvaajat

Olkoon tiedoston `tiedot.dat` sisältö seuraava:

```
0.5 1
1   3
1.8 0.7
```

Voimme lukea datan sisään seuraavasti:

```
>> load tiedot.dat
>> tiedot
tiedot =
    0.5000    1.0000
    1.0000    3.0000
    1.8000    0.7000
```

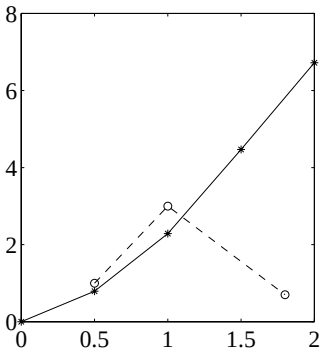
Siis tiedoston sisältö sijoitettiin muuttujaan `tiedot`. Tehdään nyt seuraavat sijoitukset muuttujiin `xx` ja `yy`:

```
>> xx = tiedot(:,1)'
xx =
    0.5000    1.0000    1.8000
>> yy = tiedot(:,2)'
yy =
    1.0000    3.0000    0.7000
```

Voimme nyt tulostaa tämän funktion kuvaajan yhdessä aiemman funktion kuvaajan kanssa:

```
>> plot(xx, yy, 'o--', x, y, '*-')
```

Tämä komento piirtää ensimmäisen kuvaajan käyttäen symbolia `o` ja katkoviivaa; toinen kuvaaja piirretään samoin kuin edellä. Tulos on seuraava:



Kuvaajan voi piirtää myös useassa vaiheessa käyttämällä komentoa `hold`, jolla voi jättää edellisen kuvaajan odottamaan seuraavan piirtämistä:

```
>> plot(x, y, '*-')
>> hold on
>> plot(xx, yy, 'o--')
>> hold off
```

Tulos näyttää samalta kuin edelläkin.

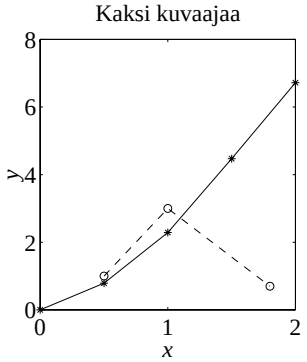
Kuvan ominaisuudet

Komento	*Selitys*
`title`	Kuvan otsikko
`xlabel`	Nimi *x*-akselille
`ylabel`	Nimi *y*-akselille
`legend`	Selitetekstit
`get`	Kuvan ominaisuuksien kysely
`set`	Kuvan ominaisuuksien asettaminen
`gcf`	Osoitin nykyiseen kuvaikkunaan
`gca`	Osoitin nykyisiin akseleihin
`axis`	Akselien skaalan asetus
`axes`	Luo uudet akselit kuvaikkunaan
`grid`	Piirtää ristikon kuvan taustalle
`subplot`	Erillisiä kuvaajia samaan kuvaan

Edelliseen kuvaajaan voi lisätä otsikon ja nimetä akselit seuraavasti:

```
>> title('Kaksi kuvaajaa')
>> xlabel('{\slx}'); ylabel('{\sly}')
```

Tulos on seuraava:



Kuvan ominaisuuksia voi selvittää komennon `get` avulla:

```
>> get(gcf)
        BackingStore = on
        CloseRequestFcn = closereq
        Color = [0.8 0.8 0.8]
...
        Position = [296 340 560 420]
...
```

Voimme muuttaa kuvaikkunan sijaintia ja kokoa seuraavasti:

```
>> set(gcf, 'Position', [600 300 300 300])
```

Vastaavasti voimme kysellä kuvan akselien ominaisuuksia:

```
>> get(gca)
        AmbientLightColor = [1 1 1]
        Box = on
        CameraPosition = [1 4 17.3205]
...
        XTick = [0 1 2]
...
```

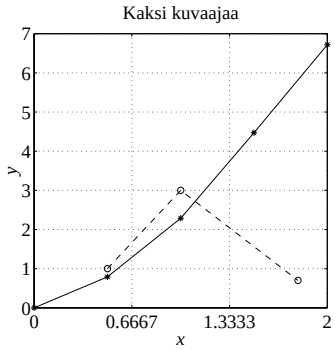
Voimme nyt muuttaa *x*-akselilla esitettyä numerointia:

```
>> set(gca, 'XTick', [0 2/3 4/3 2])
```

Komennolla

```
>> grid on
```

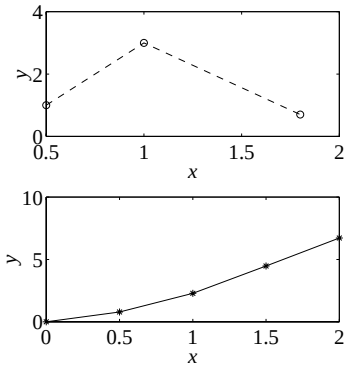
voimme piirtää ruudukon kuvaajan taustalle. Tulos näyttää seuraavalta:



Komennon `subplot` avulla voimme piirtää samaan kuvaan useamman kuvaajan. Otamme seuraavassa myös talteen osoittimet eri kuvaajiin:

```
>> h1 = subplot(2, 1, 1);
>> plot(xx, yy, 'o--')
>> xlabel('{\slx}'); ylabel('{\sly}')
>> h2 = subplot(2, 1, 2);
>> plot(x, y, '*-')
>> xlabel('{\slx}'); ylabel('{\sly}')
```

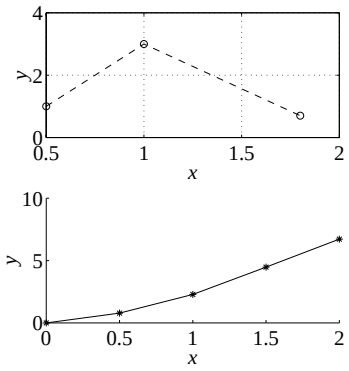
Tulos näyttää seuraavalta:



Voimme nyt muuttaa kuvaajien asetuksia käyttämällä osoittimia `h1` ja `h2`:

```
>> axes(h1)
>> grid on
>> axes(h2)
>> get(gca)
...
        Box = on
...
>> set(gca, 'Box')
[ on | {off} ]
>> set(gca, 'Box','off')
```

Tulos näyttää seuraavalta:



Kolmiulotteiset kuvaajat

Komento	*Selitys*
`mesh`	Kolmiulotteinen verkkokuvaaja
`meshgrid`	Hiladata kuvaajan piirtoa varten
`surf`	Kolmiulotteinen pintakaavio
`plot3`	Viivat ja pisteet 3D:ssä

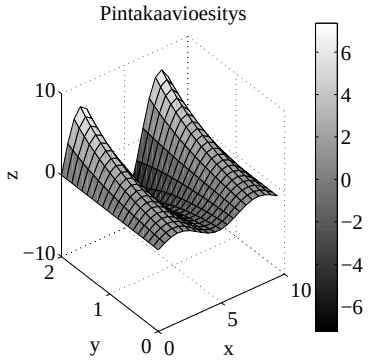
Voimme esittää funktion $g(x,y) = \sin(x)e^y$ Matlabin m-tiedostona `g.m`, jonka sisältö on seuraava:

```
function z = g(x,y)
z = sin(x) .* exp(y);
```

Voimme esittää tämän funktion pintakaaviona:

```
>> figure
>> x = linspace(0,3*pi,20);
>> y = linspace(0,2,20);
>> [X, Y] = meshgrid(x, y);
>> Z = g(X, Y);
>> colormap(gray)
>> surf(X, Y, Z)
>> xlabel('x')
>> ylabel('y')
>> zlabel('z')
>> title('Pintakaavioesitys')
>> colorbar vert
```

Tässä avattiin `figure`-komenolla uusi ikkuna, johon piirrettiin kuvaaja. Tulos näyttää seuraavalta:



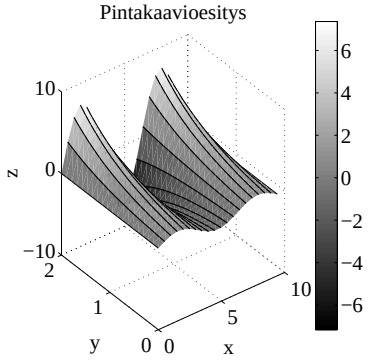
Saamme talteen osoittimen pintakaavion ominaisuuksiin seuraavasti:

```
>> hs = surf(X, Y, Z);
```

Tämän jälkeen voimme kysellä ja muuttaa kuvaajan ominaisuuksia:

```
>> set(hs,'MeshStyle')
[ {both} | row | column ]
>> set(hs,'MeshStyle','column')
```

Tulos näyttää seuraavalta:



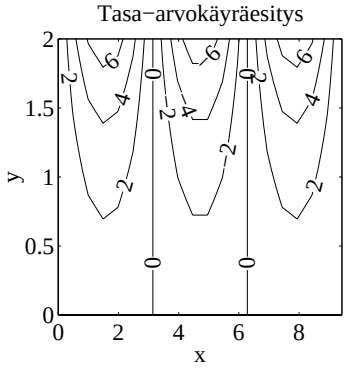
Tasa-arvokäyrät

Komento	*Selitys*
`contour`	Tasa-arvokäyrien piirto
`contourf`	Täytetyt tasa-arvokäyrien välit
`contour3`	Kolmiulotteiset tasa-arvokäyrät
`quiver`	Nuolikuvaaja

Voimme esittää edellä käytetyn datan myös tasa-arvokäyrien avulla:

```
>> colormap jet
>> [c,h] = contour(X, Y, Z);
>> xlabel('x')
>> ylabel('y')
>> title('Tasa-arvokäyräesitys')
```

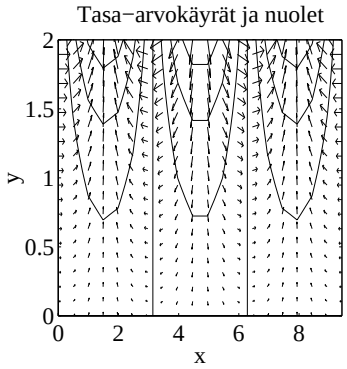
Tulos näyttää seuraavalta:



Tasa-arvokäyriin voi liittää myös gradienttia kuvaavat nuolet:

```
>> contour(X, Y, Z); hold on
>> [px,py] = gradient(Z,0.5,0.25);
>> quiver(X,Y,px,py); hold off
```

Tulos näyttää seuraavalta:



Tulostus

Kuvan saa tulostettua EPS-muodossa komennolla

```
>> print -deps kuva
```

Kuva on tämän jälkeen tiedostossa `kuva.eps`. Lisätietoja tulostuksesta saa komennolla `help print`. Komennolla `imwrite` voi tulostaa kuvia esimerkiksi JPEG-muodossa.

Tulostettavan kuvan ominaisuuksia voi muuttaa käyttämällä komentoa `get`:

```
>> get(gcf, 'PaperUnits')
ans =
inches
>> set(gcf, 'PaperUnits')
[ {inches} | centimeters | normalized
  | points ]
>> set(gcf, 'PaperUnits', 'centimeters')
>> get(gcf, 'PaperPosition')
ans =
    0.6345    6.3452   20.3046   15.2284
>> set(gcf, 'PaperPosition', [0.5 5 15 15])
>> print -loose -deps kuva
```

Tässä muutimme kuvan tulostuskokoa asettamalla kuvan määreelle `PaperPosition` uuden arvon.

Lisätietoja

Matlabin perusteista kerrotaan artikkelissa *Johdatus Matlab 5.x:n käyttöön* [Haa99b]. @CSC-lehdessä on ilmestynyt myös muita Matlab-artikkeleja [Sav98a, Sav98b, Haa98b, HS98, Haa98c, Haa99a]. CSC:n oppaassa *Matemaattiset ohjelmistot* [Haa98a] kerrotaan yleisemmin matemaattisten ohjelmistojen käytöstä.

Pikakatsaus Matlabin grafiikkaan löytyy www-osoitteesta

```
http://www.csc.fi/visualization/intensk99/
  matlab/
```

Matlabin käsikirjasta *Using Matlab Graphics* [Mat96] löytyy perusteellinen esitys Matlabin grafiikkaominaisuuksista.

Kirjallisuutta

[Haa98a] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[Haa98b] Juha Haataja, Matlab 5.x — kysymyksiä ja vastauksia, *@CSC*, 2/1998.

[Haa98c] Juha Haataja, Optimointitehtävien ratkaisu Matlabilla, *@CSC*, 3/1998.

[Haa99a] Juha Haataja, Geneettiset algoritmit ja Matlab, *@CSC*, 3/1999.

[Haa99b] Juha Haataja, Johdatus Matlab 5.x:n käyttöön, *@CSC*, 5/1999.

[HS98] Juha Haataja ja Ville Savolainen, Matlab 5.x — lisää kysymyksiä ja vastauksia, *@CSC*, 3/1998.

[Mat96] The MathWorks, Inc. *Using Matlab Graphics, Version 5*, 1996.

[Sav98a] Ville Savolainen, Matlab 5.x — uusia kysymyksiä ja vastauksia, *@CSC*, 5/1998.

[Sav98b] Ville Savolainen, Tehokas ohjelmointi Matlabilla, *@CSC*, 1/1998.

Matlabin tilastotiedettä

Esa Lammi
Esa.Lammi@csc.fi

Cedarilla olevassa Matlab-ohjelmistossa on mukana mm. Statistics toolbox. Se koostuu kaikkiaan yli sadasta Matlabin M-tiedostosta.

Tilastollisen käyttötarkoituksen mukaan ne voidaan jakaa viiteen luokkaan, jotka ovat seuraavat:

- todennäköisyysjakaumat
- tilastollisia tunnusluvut
- tilastolliset piirrokset
- lineaarisiset mallit
- hypoteesin testaus

Statistics toolbox tukee 14 todennäköisyysjakaumaa. Kustakin jakaumasta on viisi erilaista funktiota.

Todennäköisyysjakaumat

Matlabin Statistics toolboxissa on todennäköisyysjakaumia yhdeksän jatkuvaa ja viisi diskreettiä. Käytettävissä on seuraavat jakaumat:

- jatkuvat: beta, χ^2 eksponenttijakauma, F, Gamma normaalijakauma, T, tasajakauma Weibull
- diskreetit: binomi, diskreetti tasajakauma, geometrinen, hypergeometrinen, poisson.

Jokaiseen jakaumaan liittyy seuraavat funktiot:

- tiheysfunktio
- kertymäfunktio
- käänteiskertymäfunktio
- satunnaislukugeneraattori
- keskiarvo ja varianssi parametrien funktiona

Seuraavassa tarkastelemme lähemmin näitä funktioita.

Tiheysfunktio

Tiheysfunktiolla on eri merkitys jatkuville ja epäjatkuville jakaumille.

Diskreeteille jakaumille tiheysfunktio on tietyn havainnon pistetodennäköisyysfunktio. Jatkuville jakaumille tiheysfunktio ei ole todennäköisyys havaita tiettyä arvoa. Jokaisen tietyn arvon esiintymistodennäköisyys on nolla. Jatkuvien jakaumien todennäköisyydet saadaan integroimalla tiheysfunktio kiinnostavan välin yli.

Tiheysfunktiolla on kaksi teoreettista ominaisuutta:

- tiheysfunktio on nolla tai positiivinen jokaiselle mahdolliselle arvolle
- integraali yli koko tiheysfunktion arvojen joukon on yksi.

Matlabissa tiheysfunktiota kutsutaan jokaiselle jakaumalle saman yleisen formaatin mukaisesti. Esimerkiksi normaalijakauman, jonka keskiarvo on nolla ja keskihajonta on yksi, tiheysfunktio välillä -3:sta 3:een 0.1:n välein saadaan seuraavalla funktionkutsulla:

```
x = [-3:0.1:3];
f = normpdf(x,0,1);
```

Kertymäfunktio

Jos f on tiheysfunktio, niin siihen liittyvä kertymäfunktio F on

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt.$$

Kertymäfunktio $F(x)$ on todennäköisyys sille, että saatu tulos on pienempi tai yhtäsuuri kuin x. Kertymäfunktion teoreettisena perustana on seuraavat ominaisuudet:

- kertymäfunktion arvo vaihtelee 0:sta 1:een
- jos $y > x$, niin y:n kertymäfunktio on suurempi tai yhtäsuuri kuin x:n.

Normaalijakauman kertymäfunktion kutsu vastaavilla arvoilla kuin edellä on

```
x = [-3:0.1:3];
p = normcdf(x,0,1);
```

Muuttuja p sisältää normaalijakauman tiheysfunktioon parametreillä 0 ja 1 arvoilla x liittyvät todennäköisyydet.

Käänteiskertymäfunktio

Käänteiskertymäfunktio on hyödyllinen hypoteesien testauksessa ja luottamusvälien määräämisessä. Se palauttaa arvot annetuilla todennäköisyyksillä. Jatkuvan kertymäfunktion ja sen käänteisfunktion suhdetta tarkasteltaessa käytetään käänteisfunktiota:

```
x = [-3:0.1:3];
xuusi = norminv(normcdf(x,0,1),0,1);
```

Tuloksena saadaan tarkalleen annetut x:n arvot eli x ja $xuusi$ ovat samat.

Epäjatkuvilla jakaumilla kertymäfunktion ja sen käänteisfunktion riippuvuus on erilainen. Käänteiskertymäfunktio palauttaa tällöin ensimmäisen sellaisen x:n arvon, jolla kertymäfunktio saavuttaa arvoksi vähintään halutun todennäköisyyden.

Esimerkkinä tuotetaan normaalijakuman 95%:n luottamusvälit parametreillä 0 ja 1. Tässä x:n arvot määräävät välin, joka sisältää 95% standardin normaalijakauman todennäköisyysfunktiosta.

```
p = [0.025 0.975];
x = norminv(p,0,1)
```

Tuloksena saadaan

```
x =
   -1.9600    1.9600
```

Satunnaislukugeneraattori

Satunnaislukuja voidaan tuottaa kaikista jakaumista. Tasajakautunut satunnaislukugeneraattori voi tuottaa satunnaislukuja eri jakaumista kolmella eri menetelmällä: suoraan, käyttäen hyväksi käänteisyyttä tai hylkäämismenetelmällä.

Suorassa menetelmässä satunnaislukujen saanti perustuu kunkin jakauman määritelmään.

Käänteisessä menetelmässä tuotetaan satunnaislukuja jakaumasta käänteisfunktion avulla.

Hylkäämismenetelmässä halutaan tuottaa satunnaislukuja tietyn jakauman tiheysfunktiosta f. Sitä varten on löydettävä toinen tiheysfunktio g ja vakio c siten, että

$$f(x) \leq cg(x) \quad \text{kaikilla } x.$$

Tilastollisia tunnuslukuja

Tiheysfunktion keskiarvo ja varianssi ovat jakauman parametrien funktioita. Esimerkiksi betajakauman keskiarvo ja varianssi saadaan seuraavasti:

```
a = 1:6;
[m,v] = betastat(a,a)
```

Tuloksena saadaan keskiarvo m ja varianssi v:

```
m =
    0.5000    0.5000    0.5000    0.5000
    0.5000    0.5000
v =
    0.0833    0.0500    0.0357    0.0278
    0.0227    0.0192
```

Matlabin Statistics toolboxissa on kaikkiaan viisi erilaista keskiarvoa. Ne ovat seuraavat:

- `geomean` = geometrinen keskiarvo
- `harmmean` = harmoninen keskiarvo
- `mean` = aritmeettinen keskiarvo
- `median` = mediaani
- `trimmean` = keskiarvo, jossa pudotetaan pois haluttu määrä havaintojen suurimmasta ja pienimmästä päästä.

Hajontaa voidaan mitata myöskin viidellä eri funktiolla:

- `iqr` = kvartiiliväli
- `mad` = keskipoikkeama
- `range` = vaihteluväli
- `std` = keskihajonta
- `var` = varianssi

Tilastolliset kuvaajat

Graafisiin esityksiin on Matlabin Statistics tooboxissa viisi funktiota:

- `boxplot` havainnolistaa datan laatikoina.
- `fsurfht` mahdollistaa käyräpintojen piirron interaktiivisesti.
- `normplot` testaa normaalisuutta graafisesti.
- `qqplot` testaa graafisesti, ovatko kaksi otosta peräisin samasta todennäköisyysjakaumasta.
- `surfht`-funktiolla voidaan piirtää korkeuskäyriä interaktiivisesti.

Lineaarisia malleja

Lineaariset mallit käsittävät yksisuuntaisen varianssianalyysin, kaksisuuntaisen varianssianalyysin, polynomisen regression ja moninkertaisen lineaarisen regression.

Lineaariset mallit voidaan esittää muodossa

$$y = X\beta + \varepsilon,$$

missä y on $n \times l$ -havaintovektori, X on selittävien muuttujien $n \times p$ -matriisi, β on parametrien $p \times l$ -vektori ja ε on satunnaisvirheen $n \times l$ -vektori.

Yksisuuntainen varianssianalyysi

Yksisuuntaisen varianssianalyysin ANOVAn tarkoituksena on selvittää, onko eri ryhmien havaintoarvoilla sama keskiarvo. Se palauttaa p-arvon nollahypoteesille, että X:n sarakkeiden keskiarvot ovat yhtäsuuria. Jos p-arvo on lähellä nollaa, niin on ilmeistä, että nollahypoteesi on väärä. Tällöin tulkitaan X:n sarakkeiden keskiarvojen olevan erisuuret.

ANOVA suoritetaan käskyllä

```
p = anova1(X)
```

tai käskyllä

```
p = anova1(x,group)
```

Jälkimmäisessä tapauksessa verrataan parametrin `group` indeksoimien muuttujien keskiarvoja x:ssä. Tuloksena saadaan kaksi taulua, joista toisessa on numerollinen selitys ja toisessa vastaava graafinen esitys.

Kaksisuuntainen varianssianalyysi

Funktio `anova2(X,reps)` suorittaa tasapainoiselle aineistolle kaksisuuntaista varianssianalyysiä. Se tehdään vertaamalla X:ssä kahden tai useamman sarakkeen ja kahden tai useamman rivin keskiarvoja. Eri sarakkeissa olevat arvot kuvaavat vaihteluja yhdessä faktorissa. Vastaavasti eri riveillä olevat arvot esittävät vaihteluja toisessa faktorissa. Mikäli rivi-sarake-parilla on enemmän kuin yksi havainto, niin funktion kutsussa oleva `reps` ilmoittaa havaintojen määrän.

Funktio `anova2` palauttaa p-arvon nollahypoteesille, että X:n sarakkeiden keskiarvot ja rivien keskiarvot ovat yhtäsuuria. Jos p-arvo on lähellä nollaa, on nollahypoteesi hylättävä.

Moninkertainen lineaarinen regressio

Moninkertaisen lineaarisen regression tarkoituksena on aikaansaada määrällinen yhteys ennustavien muuttujien ja vastemuuttujan välillä. Tätä yhteyttä voidaan käyttää selventämään, mitkä ennustavat muuttujat ovat tehokkaimmat. Se ilmaisee myöskin vaikutuksen suunnan, kuinka y:n arvo muuttuu x:n muuttuessa.

Lineaarinen malli on muotoa

$$y = X\beta + \epsilon,$$

missä y on havaintovektori kooltaan $n*l$, X on selittävien muuttujien $n*p$-matriisi, β parametrien $p*l$-suuruinen vektori ja ϵ on $n*l$:n suuruinen satunnaishäiriövektori.

Hypoteesin testausta

Hypoteesin testauksella määritetään, onko väite populaation luonteenpiirteestä järkevä.

Hypoteesin testauksen käsitteitä

Nollahypoteesi on alkuperäinen väite. Sitä merkitään lausekkeella

$$H_0 : \mu = a,$$

missä μ on keskiarvo ja a on testattava arvo.

Vaihtoehtoinen hypoteesi edelliselle voi sisältää kolme erilaista hypoteesia. Ne ovat

$$H_1 \ : \ \mu > a \text{ tai}$$
$$H_1 \ : \ \mu < a \text{ tai}$$
$$H_1 \ : \ \mu \neq a.$$

Merkitsevyystasolla määritetään se varmuuden aste, millä nollahypoteesi voidaan hylätä ja valita vaihtoehtoinen hypoteesi tarkastelun kohteeksi.

Saadun otostuloksen todennäköisyyttä oletuksesta, että nollahypoteesi on tosi, kuvaa *p-arvo*.

Hypoteesin testauksessa saadaan myös halutun suuruiset *luottamusvälit* lasketuille arvoille. Jos $100(1-\alpha)$%:n luottamusväli ei sisällä testattua arvoa, niin nollahypoteesi hylätään merkitsevyystasolla α.

Esimerkki hypoteesin testauksesta

Tarkastellaan seuraavassa esimerkissä viime eduskuntavaalien voittajan SDP:n vaalimenestystä kahdessa eri vaalipiirissä. Uudenmaan ja Hämeen pohjoisen vaalipiireistä valitaan kuuden kunnan otos kummastakin. Niillä pyritään testaamaan, onko SDP:n menestys näissä vaalipiireissä samanlainen kuin oli koko maassa eli 28.3% annetuista äänistä. Seuraavassa tarkastelemme saatua tulosta:

```
>> load sdp
>> uusimaa
uusimaa =
  0.3897   0.3285   0.1744   0.2874   0.5035   0.2110
>> hame
hame =
  0.0954   0.2770   0.3470   0.2380   0.1757   0.3037
```

Tiedot on talletettu aiemmassa Matlab-istunnossa tiedostoon `sdp.mat`. Lukuarvot esittävät SDP:n prosenttiosuutta kunnan annetuista äänistä. Mukaan otetut Uudenmaan läänin kunnat ovat: Hyvinkää, Karkkila, Liljendal, Mäntsälä, Pohja ja Ruotsinpyhtää. Hämeen pohjoisesta on mukana seuraavat kunnat: Kihniö, Lempäälä, Nokia, Ruovesi, Vesilahti ja Ylöjärvi.

```
>> [h,parvo,ci] = ztest(uusimaa,0.283,0.1)
h =
     0
parvo =
    0.4224
ci =
    0.2357    0.3958
```

Tässä testattiin `ztest`-funktiolla, onko Uudenmaan läänissä SDP:n ääniosuus saman suuruinen kuin koko maassa eli 28.3%. Keskihajonnaksi oletettiin 10%-yksikköä.

Parametrin *h* arvo 0 tarkoittaa, että ei voida hylätä nollahypoteesiä: SDP:n äänimäärä Uudenmaan läänissä on keskimäärin 28.3%. Luotettavuusrajat SDP:n ääniosuudelle Uudenmaan läänissä osoittavat, että puolueen osuus Uudenmaan äänistä on 23.6%:n ja 39,6%:n välillä.

```
>> [h,parvo,ci] = ttest(hame,0.283)
h =
     0
parvo =
    0.2969
ci =
    0.1434    0.3356
```

SDP:n ääniosuutta Hämeen pohjoisen vaalipiirin äänistä verrattiin puolueen valtakunnalliseen keskiarvoon `ttest`-funktiolla. Se ei oleta keskihajontaa tunnetuksi.

Tässäkin tapauksessa saadaan parametrin arvoksi 0. Se hyväksyy nollahypoteesin, että SDP:n ääniosuus Hämeen pohjoisessa vaalipiirissä on sama kuin puolueella koko maassa. Äänimäärän osuus Hämeen pohjoisessa vaalipiirissä on 14.3%:n ja 33.6%:n välillä.

```
>> [h,sig,ci] = ttest2(uusimaa,hame)
h =
     0
sig =
    0.2454
ci =
   -0.0614    0.2140
```

Funktiolla `ttest2` testataan SDP:n ääniosuutta edellä tarkastelluissa vaalipiireissä. Parametri *h* ilmoittaa arvollaan 0, että nollahypoteesi ääniosuuksien yhtäsuuruuksista on hyväksyttävä. Keskiarvojen erotus Uudenmaan ja Hämeen pohjoisen vaalipiirien välillä vaihtelee -6.1%-yksikön ja 21.4%-yksikön välillä.

Lähdeluettelo

[1] Statistics Toolbox, User's Guide, September 1993

Regressioanalyysi Matlabilla

Esa Lammi
Esa.Lammi@csc.fi

Matlab tarjoaa Statistics toolboxin myötä välineet regressioanalyysiin. Useamman muuttujan regressioanalyysissä määritetään riippuva muuttuja riippumattomien muuttujien avulla. Selitettävä muuttuja voidaan silloin ilmaista muodossa

$$y = a_0 + a_1 x_1 + \ldots + a_n x_n.$$

Mallin tuntemattomat kertoimet $a_0, a_1, ..., a_n$ määritetään käyttäen pienimmän neliösumman menetelmää.

Esimerkkiajo

Tarkastellaan 24 satunnaisesti valitun kunnan veroäyrejä. Pyritään selittämään niitä kunnan asukasluvulla, pinta-alalla, kauppojen lukumäärällä ja kunnat menoilla. Veroäyrit ja menot ovat tuhansina. Matlab käynnistyy komennolla `matlab`, jonka jälkeen voimme lukea sisään datat:

```
>> load kunnat
```

Tarvittava data-aineisto sijaitsee tiedostossa kunnat. Määritellään tämän jälkeen muuttujat: ayrit, aslkm, pala, kaupat ja menot.

```
>> ayrit=kunnat(:,1);
>> aslkm=kunnat(:,2);
>> pala=kunnat(:,3);
>> kaupat=kunnat(:,4);
>> menot=kunnat(:,5);
```

Määritetään riippumattomien muuttujien muodostama matriisi X ja sen jälkeen kertoimet.

```
>> X=[ones(size(aslkm)) aslkm pala kaupat menot];
>> a=X\ayrit
a =
   1.0e+03 *

    4.6562
    0.1042
   -0.8918
   -5.0326
    0.0003
```

Täten pienimmän neliösumman antama malli on seuraava:

$$\text{ayrit} = 4656.2 + 104.2\,\text{aslkm} - 891.8\,\text{pala} - 5032.6\,\text{kaupat} + 0.3\,\text{menot}.$$

Määääritetään tämän jälkeen havaintoaineiston maksimipoikkeama mallista.

```
>> Y=X*a;
>> MaxErr = max(abs(Y-ayrit))
MaxErr =
   9.2589e+05
```

Regression määrittäminen komennolla `stepwise`

Stepwise-funktio määrittää regressiomallin käyttäen riippumattomien muuttujien muodostamaa matriisia. Olkoon tämä matriisi `XX`, jolloin

```
>> XX=[aslkm pala kaupat menot];
```

Antamalla nyt komento

```
>> stepwise(XX,ayrit)
```

saadaan kolme eri taulua, jotka antavat informaatiota mallista. Kertoimet esitetään 95%:n luottamusväleillä. Napauttamalla hiirellä kerrointaulukon muuttujaa voidaan mallista poistaa muuttuja, jolloin jäljellä jäävien muuttujien arvot muuttuvat. Mallissa mukana olevat muuttujat ovat vihreällä värillä ja mallista poistetut punaisella värillä. Muuttuja saadaan takaisin malliin näpäyttämällä hiirellä mallista poistettua muuttujaa.

Toinen taulu kertoo muuttujien merkitsevyyden. Mikäli muuttujan arvon yhteydessä on yhtenäinen viiva, on muuttuja merkittävä. Niiden muuttujien arvo, jotka eivät merkittävästi eroa nollasta, merkitään pisteviivoilla.

Regression määrittäminen funktiolla `regress`

Funktio `regress` palauttaa pienimmän neliösumman menetelmän sovituksen riippuvalle muuttujalle annettujen riippumattomien muuttujien avulla. Malli on tällöin muotoa

$$y = X\beta + \epsilon,$$

jossa ϵ kuvaa satunnaisvaihtelua.

Edellä olevaa esimerkkiä hyväksikäyttäen saadaan:

```
>> [B,BINT,R,RINT,STATS] = ...
   regress(ayrit,X,0.05)
B =
   1.0e+03 *
```

```
    0.1043
   -0.8897
   -5.0175
    0.0003
```

Mallin kertoimet ovat nyt B:ssä ja siihen liittyvät 95%:n luottamusvälit ovat BINTin arvoina.

```
BINT =
   1.0e+03 *

    0.0818    0.1268
   -1.5957   -0.1837
   -6.7144   -3.3206
   -0.0007    0.0014

R =
   1.0e+05 *

    5.1023
    5.7106
   -9.2647
    ⋮
    0.4714
```

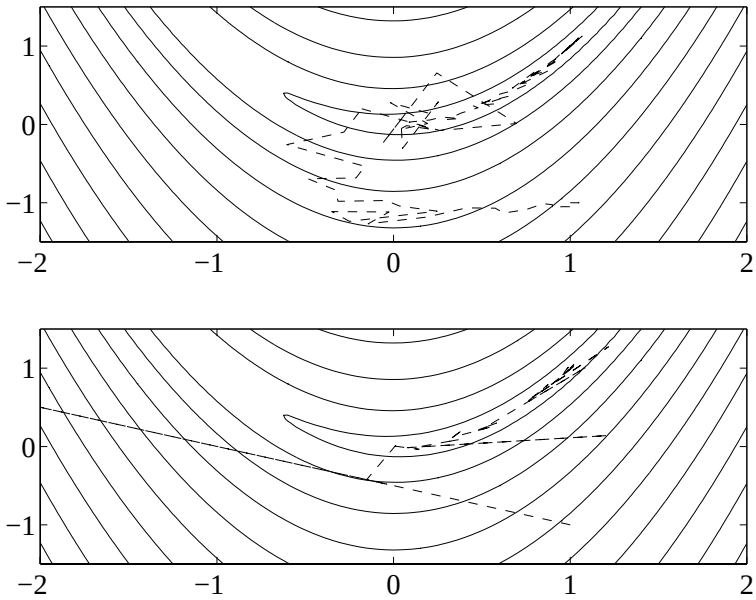
Muuttuja R pitää sisällään residuaalin arvot ja RINT sen 95%:n luottamusvälit.

```
RINT =
   1.0e+06 *

    0.3671    0.6534
    0.0746    1.0675
   -1.6592   -0.1937
    ⋮
   -0.8324    0.9267

STATS =
    0.9894  601.2134         0
```

Muuttuja STATS sisältää selitysasteen, F- ja p-arvot.

Optimointitehtävien ratkaisu Matlabilla

Juha Haataja
Juha.Haataja@csc.fi



Kuva 1: *Rosenbrockin funktion minimointi rutiineilla `fmins` (ylempi kuvaaja) ja `fminu` (alempi kuvaaja).*

Esittelen tässä artikkelissa optimointitehtävien ratkaisua Matlabin rutiinikokoelmalla Optimization Toolbox [Gra93], joka on asennettu CSC:n Cypress- ja Caper-koneille. Optimointitehtävien ratkaisua on käsitelty CSC:n oppaassa [Haa95]. Matemaattisia ohjelmistoja on esitelty CSC:n oppaassa [Haa98], joka on saatavissa WWW-järjestelmästä.

Ohjelmointi Matlabilla

Matlab-kieli mahdollistaa varsin tehokkaan tavan ohjelmoida matemaattisia algoritmeja. Matlabilla voi esimerkiksi rakentaa algoritmin prototyypin ja testata sen toimintaa. Tämän jälkeen lopullisen toteutuksen voi tehdä esimerkiksi C- tai Fortran-kielellä. Matlabin etuna ohjelmankehityksessä on

- nopea, interaktiivinen matriisikieli, ei tarvita ohjelman kääntämistä
- m-tiedostojen avulla voi helposti rakentaa tarvittavia funktioita ja käyttää niitä rakennuspalikoina
- Matlabin työkalupakeista (Toolbox) löytyy usein valmiita rutiineita algoritmin toteuttamiseen.

Rajoitteeton optimointi

Matlabin Optimization Toolboxin sisältämiä rutiineja on esitelty taulukossa 1. Kokeilemme yksinkertaisena esimerkkinä Rosenbrockin funktion

$$f(\mathbf{x}) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$$

minimointia. Kirjoitetaan aluksi funktion määrittely tiedostoon `rbrock.m`:

```
function z = rbrock(x)
z = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
```

Haemme Rosenbrockin funktion minimin rutiineilla `fmins` ja `fminu`. Iteraation alkupiste on $(1,-1)$:

```
>> flops(0); rs = fmins('rbrock',[1 -1])
rs =
    1.0000    1.0001
>> flops
ans =
        2820
>> flops(0); rs = fminu('rbrock',[1 -1])
rs =
    0.9999    0.9998
>> flops
ans =
        4522
```

Kumpikin rutiini löysi ratkaisun $x = (1,1)$. Polytooppihakua käyttävä `fmins`-rutiini näytti olevan tehokkaampi tässä tehtävässä.

Voimme muuttaa kohdefunktion `rbrock` tallettamaan läpikäydyt iteraatiopisteet, jolloin voimme tutkia algoritmin käyttäytymistä:

```
function z = rbrock(x)
global X  % talletettavat arvot
X = [X; x];
z = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
```

Seuraavassa piirrämme ratkaisussa tarvitut iteraatiopisteet. Lisäksi piirrämme kohdefunktion tasa-arvokäyrät ja tulostamme kuvan tiedostoon PostScript-muodossa (katso kuvaa 1):

```
global X;  % arvojen talletusta varten
x1 = -2:0.05:2;
x2 = -1.5:0.05:1.5;
[xx1 xx2] = meshgrid(x1, x2);
z = 100*(xx2 - xx1.^2).^2 + (1 - xx1).^2;
subplot(2,1,1);
axis([-2 2 -1.5 1.5]);
contour(x1,x2,z,(0:1.4:50).^3); hold on
X = [];
res = fmins('rbrock',[1 -1]);
plot(X(:,1)',X(:,2)','--'); hold off
subplot(2,1,2);
axis([-2 2 -1.5 1.5]);
contour(x1,x2,z,(0:1.4:50).^3); hold on
X = [];
res = fminu('rbrock',[1 -1]);
plot(X(:,1)',X(:,2)','--'); hold off
print -deps rbrock
```

Kuvasta 1 nähdään, että `fmins`-rutiini tekee ensimmäisillä iteraatioilla polytooppihaulle tyypillistä haarukointia, kunnes löytää tiensä Rosenbrockin funktion ”laaksoon”, jonka jälkeen suppeneminen on melko nopeaa. Rutiini `fminu` puolestaan menee aluksi nollakohdan ohi ja palaa sitten takaisin. Tässä käytettiin rutiinin oletusmenetelmää, joka on BFGS-algoritmi.

Rajoitteinen optimointi

Ratkaistaan Rosenbrockin funktion ja rajoitteen $h(x) = x_1^2 + x_2^2 - 1 = 0$ muodostama minimointitehtävä (kuva 2) käyttäen toistettua kvadraattista optimointia (SQP, sequential quadratic programming). Ratkaisurutiinina on Optimization Toolboxin funktio `constr`.

Seuraavassa on määritelty kohdefunktio rajoitteineen Matlabin funktiotiedostoon `rbconstr.m`:

```
function [f,h] = rbconstr(x)
f = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
h(1) = x(1)^2 + x(2)^2 - 1;
```

Asetamme ratkaisualgoritmin tulostamaan tietoja ratkaisun kulusta (valitsin 1), toleranssit kohdefunktiolle ja rajoitusehdoille (valitsimet 3 ja 4) sekä yhtäsuuruusrajoitteiden lukumäärän (valitsin 13):

```
>> options(1) = 1;
>> options(3) = 1e-8; options(4) = 1e-10;
>> options(13) = 1;
>> res = constr('rbconstr',[1 -1],options)
f-COUNT  FUNCTION  MAX{g}       STEP
    3         400      -1          1
   ...
   78       100.99      4.17e-13  -6.1e-05
Optimization Terminated Successfully
Optimization Converged Successfully
Active Constraints:
     1
res =
    0.0099   -1.0000
>> [f,h] = rbconstr(res)
f =
  100.9901
h =
   4.1700e-13
```

Rutiini `constr` siis löysi minimin. Kyseessä on kuitenkin lokaali eikä globaali minimi, mikä havaitaan käynnistettäessä iteraatio eri alkuarvauksella:

```
>> res = constr('rbconstr',[1 0],options)
    ...
res =
    0.7864    0.6177
>> [f,h] = rbconstr(res)
f =
    0.0457
h =
   1.4788e-11
```

Varastomallin optimointi

Seuraavassa tutkimme erästä klassista varastomallia [HW63] esimerkkinä Matlabin käytöstä ohjelmankehitykseen. Optimistrategia haetaan dynaamisella ohjelmoinnilla.

Varastomme koko y_j, $j = 1,\ldots,n$, käyttäytyy deterministisen differenssiyhtälön

$$y_{j+1} = y_j + q_j - d_j$$

mukaan, missä $d_j \geq 0$ on (ennalta tiedetty) asiakkaan tilaus ja $q_j \geq 0$ on ohjaus (uusi valmistusmääräys), jolla varaston tilaa voidaan muuttaa. Lisäksi oletetaan, että $y_0 = 0$.

Kustannusfunktio olkoon

$$K = \sum_{j=1}^{n} \left[A_j \delta_j + I_j C y_{j+1} \right],$$

missä $I_j C$ on tavarayksikön varastointikustannukset jaksolla j, ja A_j on tilaamisesta aiheutuva kertakustannus sekä

$$\delta_j = \begin{cases} 0 & \text{jos } q_j = 0, \\ 1 & \text{jos } q_j > 0. \end{cases}$$

Varaston optimaalinen täydennyspolitiikka q_j saadaan laskemalla kustannusfunktiot $Z_k, k = 1,\ldots,n$:

$$Z_k(0) = \min_w Y_k(w), \quad w = 1,2,\ldots,k.$$

Tässä on

$$Y_k(w) = A_w + C\sum_{j=w}^{k-1}\left[I_j \sum_{i=j+1}^{k} d_i\right] + Z_{w-1}(0).$$

Siis $Y_k(w)$ on periodien $1,\ldots,k$ kustannus, kun oletetaan ettei, jakson k lopussa ole varastoa. Lisäksi varasto loppuu hetkellä w, ja samalla saapuu valmistusmääräys, joka täyttää kysynnän hetkeen k asti. Z_k:n määrittelevässä yhtälössä ei tarvitse lähteä liikkeelle alkuhetkestä, jos tiedetään, että Z_{k-1}:n kysynnän täyttävä tilaus saapui hetkellä v.

Kaavat funktioille $Z_k(0)$ ja $Y_k(w)$ on helppo koodata Matlabilla. Seuraavassa on tiedosto `Z.m`:

```
function [z,yy] = Z(k,v,A,C,I,d,zold)
if k == 0
  z = 0; yy = 0;
else
  yy = [];
  for w = v:k
    yy = [yy Y(k,w,v,A,C,I,d,zold)];
  end
  z = min(yy);
end
```

Seuraavassa on tiedosto `Y.m`:

```
function y = Y(k,w,v,A,C,I,d,zold)
if w > k - 1
  if length(zold) >= w
    y = A(w) + zold(w);
  else
    y = A(w) + Z(w-1,v,A,C,I,d,zold);
  end
else
  t = 0;
  for j = w:(k-1)
    t = t + I(j)*sum(d((j+1):k));
  end
  if length(zold) >= w
    y = A(w) + C*t + zold(w);
  else
    y = A(w) + C*t + Z(w-1,v,A,C,I,d,zold);
  end
end
```

Varsinaisen optimoinnin suorittaa rutiini `qopt.m`, joka palauttaa tilausvektorin `q` ja optimikustannuksen `z`:

```
function [q, z] = qopt(d,A,C,I,iter)
v = 1; z0 = 0;
for k = 1:iter
  [z,yy] = Z(k,v,A,C,I,d,z0);
  z0 = [z0 z];
  idx = 1:length(yy);
  ind = idx(yy==z) + v - 1;
  indexes(k,1) = v;
  for j = 1:length(ind)
    indexes(k,j+1) = ind(j);
  end
  maxind(k) = length(ind);
  v = ind(length(ind));
end
q = zeros(1,iter);
for k = iter:-1:1
  w0 = indexes(k,2);
  if indexes(k,1) == w0
    w = w0;
  else
    w = indexes(k,maxind(k)+1);
  end
  for j = w:k
    q(w) = q(w) + d(j); d(j) = 0;
  end
end
```

Algoritmissa haetaan optimaaliset kustannukset lähtien alkutilasta, jolloin varasto on nollassa. Kun optimikustannukset ovat selvillä, voidaan kerätä yhteen niitä vastaavat optimaaliset tilausmääräykset q_j.

Tehtävän lähtotiedot voidaan syöttää seuraavasti:

```
d = [80 50 125 200 0 200];
n = length(d); id = ones(1,n);
C = 120; I = 0.2*id/n;
```

Tässä muuttujaan `d` sijoitettiin asiakkaiden tilaukset periodeille $1,\ldots,6$. Varaston päivityksen hinta on 120 mk yksiköltä. Tilausperiodit ovat tasavälisiä, ja varastokustannus koko tarkasteluvälillä on 20% varaston arvosta.

Varaston päivityksen kertamaksu on joko 3000 mk tai 300 mk. Optimaalisen varaston täydennyspolitiikan $q_j, j = 1,\ldots,n$ voi nyt laskea seuraavasti:

```
>> d
d =    80    50   125   200     0   200
>> A = 3000*id;
>> [q,z] = qopt(d,A,C,I,n)
q =   255     0     0   400     0     0
z =        8800
>> A = 300*id;
>> [q,z] = qopt(d,A,C,I,n)
q =   130     0   125   200     0   200
z =        1400
```

Siis jos varaston päivittämisestä aiheutuvia kertamaksuja A_j pienennetään, päivitetään varastoa pienemmissä erissä ja useammin.

Muita ohjelmistoja ja lisätietoja

CSC:n tarjoamia optimointiohjelmistoja on esitelty teoksessa [Haa95]. Jos pelkkä Matlabin numeriikka ei riitä, vaan tarvitaan interaktiivista symbolinkäsittelyä, voi käyttää esimerkiksi CSC:n Maple- tai Mathematica-ohjelmistoja. Tehokkaita optimointitehtä-

vien ratkaisualgoritmeja löytyy mm. IMSL- ja NAG-aliohjelmakirjastoista. Myös Netlib-verkkoarkistosta voi etsiä optimointiohjelmistoja [Haa98].

Kirjallisuutta

[Gra93] Andrew Grace. *Optimization Toolbox For Use with MATLAB*. The MathWorks, Inc, 1993.

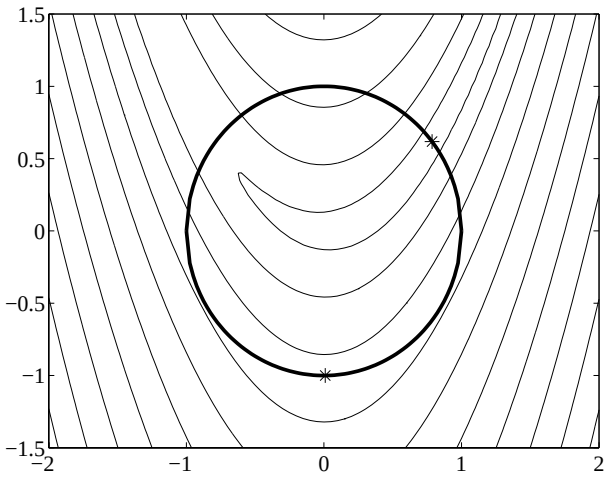
[Haa95] Juha Haataja. *Optimointitehtävien ratkaiseminen*. CSC, 1995.

[Haa98] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[HW63] G. Hadley ja T. M. Whitin. *Analysis of Inventory Systems*. Prentice-Hall, 1963.

Tehtävä	*Matem. esitys*	*Funktiokutsu*	*Menetelmä*
Skalaarifunktio	$\min f(x), x \in [a,b] \subset \mathbb{R}$	`fmin('f',a,b)`	Interpolaatio
Lineaarinen optimointi	$\min \mathbf{c}^T\mathbf{x}, A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n$	`lp(c,A,b)`	*Simplex*-algoritmi
Kvadraattinen optimointi	$\min \frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{c}^T\mathbf{x}$, $A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{R}^n$	`qp(H,c,A,b)`	Projektiomenetelmä (active set)
Rajoitteeton optimointi	$\min f(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n$	`fminu('f',x)` `fmins('f',x)`	BFGS, DFP tai jyrkin suunta Polytooppihaku
Rajoitteinen optimointi	$\min f(\mathbf{x}), \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \mathbf{x} \in \mathbb{R}^n$	`constr('fg',x)`	SQP-menetelmä
Monitavoiteoptimointi	$\min \gamma$, $\mathbf{x} \in \mathbb{R}^n, \mathbf{f}(\mathbf{x}) - W\gamma \leq \mathbf{g}$	`attgoal('f',x,g,w)`	SQP ja ansiofunktio
Minimax-optimointi	$\min \max_{1 \leq i \leq p} f_i(\mathbf{x})$, $G(\mathbf{x}) \leq \mathbf{0}$, $\mathbf{x} \in \mathbb{R}^n$	`minimax('fg',x)`	SQP ja ansiofunktio
NLSQ	$\min \sum_{i=1}^{m} f_i(\mathbf{x})^2$, $\mathbf{x} \in \mathbb{R}^n$	`leastsq('f',x)`	Levenberg-Marquardt tai Gauss-Newton
Epälineaariset yhtälöt	$\mathbf{f}(\mathbf{x}) = \mathbf{0}, \mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^n$	`fsolve('f',x)`	Levenberg-Marquardt tai Gauss-Newton

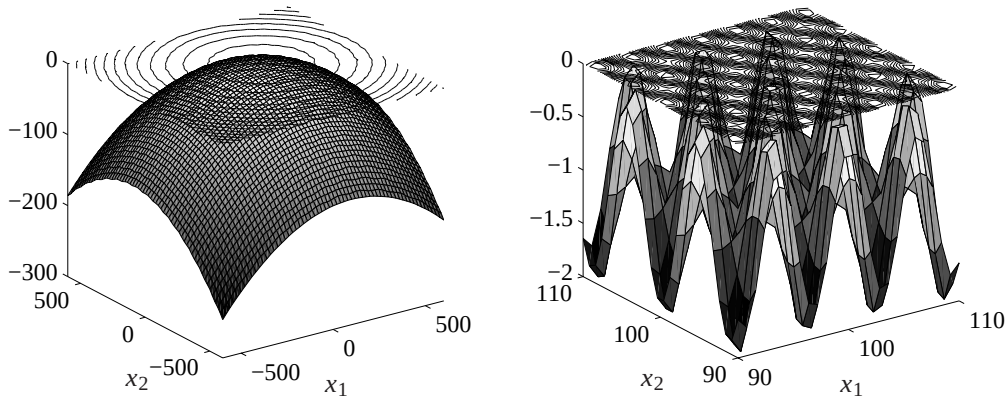
Taulukko 1: *Eräitä Matlabin optimointirutiineita. Funktiokutsuissa annetaan optimoitavan funktion (m-tiedosto) nimi merkkijonona, esimerkiksi `fminu('rbrock',[1 -1])`. Argumentti `x` on minimointitehtävän alkuarvaus.*



Kuva 2: *Rajoitteisen optimointitehtävän ratkaiseminen. Tähdellä merkityt kaksi pistettä ovat paikallisia minimejä. Löydätkö lisää paikallisia minimejä?*

Geneettiset algoritmit ja Matlab

Juha Haataja
Juha.Haataja@csc.fi



Kuva 3: *Griewankin funktio. Oikeanpuoleisessa kuvaajassa on esitetty tarkempia yksityiskohtia optimipisteen lähistöltä.*

Geneettiset algoritmit (GA) ovat luonnon evoluutiomekanismeja matkivia optimointimenetelmiä. GA-menetelmät eivät vaadi kohdefunktion tai rajoitefunktioiden jatkuvuutta, joten niitä voidaan käyttää perinteisillä menetelmillä vaikeasti ratkaistaviin ongelmiin. Lisäksi geneettiset algoritmit soveltuvat luonnostaan rinnakkaislaskentaan.

Geneettiset algoritmit sopivat tilanteisiin, joissa tarvoitteena on ”riittävän hyvän” optimikohdan löytäminen. Geneettisten algoritmien tehokkuuteen vaikuttaa suuresti mm. ongelman koodaus sopivaan esitysmuotoon.

Geneettisiä algoritmeja ei kannata käyttää (ainakaan ainoana menetelmänä) jatkuvasti differentioituvien optimointitehtävien ratkaisemiseen: mikäli saatavilla on hyvä alkuarvaus, perinteiset menetelmät ovat yleensä paljon tehokkaampia. Toisaalta jos optimointitehtävässä on epäjatkuvuuksia ja paljon paikallisia minimejä, geneettiset algoritmit voivat olla käyttökelpoisia.

GA-menetelmien rakenne

Geneettisen algoritmin toimintaperiaate on seuraava:

Alusta populaatio $\{\mathbf{x}^i\}$, $i = 1,\ldots,l$.
Laske kohdefunktion arvot: $f(\mathbf{x}^i)$, $i = 1,\ldots,l$.
repeat
  for $i = 1,2,\ldots,l$
    Risteytä kaksi valittua vektoria $\mathbf{x}^a$ and $\mathbf{x}^b$ vektoriksi $\mathbf{x}'$.
    Tuota mutaatioita: $\mathbf{x}' \to \mathbf{x}''$.
    Laske kohdefunktion arvo: $f_i = f(\mathbf{x}'')$.
    Tallenna vektori $\mathbf{x}''$ uuteen populaatioon.
  end
until löytyy tarpeeksi hyvä ratkaisu

Valitsemme alkupopulaatioksi siis joukon vektoreita $\{\mathbf{x}^i\}$, $i = 1,\ldots,l$. Populaation koko säilyy samana sukupolvesta toiseen. Kaksi sopivalla mekanismilla valittua yksilöä risteytetään keskenään vektoriksi $\mathbf{x}'$. Tätä vektoria puolestaan muutetaan satunnaisen mutaation avulla. Ratkaisuehdokkaiden paremmuuden vertaamiseen käytetään kohdefunktiota $f(\mathbf{x})$.

Esimerkkitehtävä

Etsimme maksimikohtaa jatkuvalle funktiolle $f(\mathbf{x})$: $\mathbb{R}^n \to \mathbb{R}$. Optimointitehtävällä on laatikkorajoitteet $\mathbf{x}^l \le \mathbf{x} \le \mathbf{x}^u$. Esimerkkinä käytämme ns. Griewankin funktiota

$$f(\mathbf{x}) = \left((x_1-100)^2+(x_2-100)^2\right)/4000 - \cos(x_1-100)\cos\left((x_2-100)/\sqrt{2}\right)+1.$$

Funktion kuvaaja löytyy kuvasta 3. Funktiolla on paljon paikallisia maksimikohtia. Globaali maksimipiste on $\mathbf{x}^* = (100,100)^T$.

Reaalikoodattu GA

Matlabilla on helppo toteuttaa geneettinen algoritmi, joka käsittelee reaalilukuvektoreista koostuvaa populaatiota $\{\mathbf{x}^i\}$, $\mathbf{x}^i \in \mathbb{R}^n$, $i = 1,2,\ldots,l$. Tässä artikkelissa käytetyt Matlab-rutiinit löytyvät tilan säästämiseksi www:stä [Haa].

Kaksi populaation alkiota risteytetään seuraavasti:

$$x_i' = (1-c)x_i^a + cx_i^b.$$

Tässä vektorit $\mathbf{x}^a$ ja $\mathbf{x}^b$ ovat valitut vanhemmat ja vektori $\mathbf{x}'$ on uusi yksilö. Risteytys tehdään kullakin indeksillä i todennäköisyydellä P_{cross}. Muuten käytetään vanhemman $\mathbf{x}^a$ arvoa. Kerroin c on tasajakautunut satunnaisluku väliltä $(-0.5, 1.5)$.

Valitsemme vanhemmat turnausperiaatteella. Aluksi poimimme kaksi satunnaista alkiota populaatiosta. Valitsemme näistä paremman vektorin todennäköisyydellä P_{tour} ja huonomman todennäköisyydellä $1 - P_{\text{tour}}$. Toinen risteytettävä vektori valitaan samalla periaatteella. Tämän valintamenettelyn etuna on se, että kohdefunktion arvoja ei tarvitse skaalata, sillä turnauksessa tarvitsee vain verrata arvoja keskenään.

Mutaatiossa muutamme vektorin $\mathbf{x}'$ alkioita:

$$x_i'' = x_i' + r(x_i^u - x_i^l).$$

Tässä satunnaisluku r on normaalijakautunut (keskiarvo nolla ja hajonta 0.1). Vektorit $\mathbf{x}^l$ ja $\mathbf{x}^u$ ovat tehtävän laatikkorajoitteet. Kutakin alkiota $\mathbf{x}'$ muutetaan todennäköisyydellä P_{mut}.

Käytämme myös elitismiä: otamme vanhan populaation parhaan alkion ja vertaamme sitä uuden populaation satunnaiseen alkioon. Jos uusi alkio on huonompi, korvaamme sen vanhan populaation parhaalla alkiolla.

Menetelmän käyttäytyminen

Käytämme edellä kuvattua geneettistä algoritmia kuvassa 3 esitetyn Griewankin funktion maksimointiin. Geneettistä algoritmia toistetaan 300 sukupolven ajan; populaation koko on 30 alkiota. Algoritmin parametrien arvoiksi on valittu $P_{\text{cross}} = 0.8$, $P_{\text{mut}} = 0.02$ ja $P_{\text{tour}} = 0.7$. Algoritmi ei kuitenkaan ole kovin herkkä näiden parametrien arvoille. Vertailukohtana käytämme satunnaisotantaa: valitsemme tasajakautumasta $30 \times 300 = 9000$ näytettä, joista paras otetaan maksimikohdan approksimaatioksi. Kumpaakin menetelmää toistetaan sata kertaa toisistaan riippumattomasti.

Tulokset löytyvät taulukosta 2. Sadasta geneettisen algoritmin toistokerrasta on paras ja huonoin tulos esitetty kuvassa 4. Geneettinen algoritmi löysi globaalin optimikohdan noin $2/3$ testeistä; satunnaisotanta ei onnistunut tässä kertaakaan. Toisaalta yhdistämällä satunnaisotanta ja jokin lokaalin optimoinnin menetelmä saadaan kohtuullisen tehokas menetelmä [Haa99].

Lisätietoja

Geneettisiä algoritmeja on käytetty muilla menetelmillä vaikeasti ratkaistavissa optimointitehtävissä [Haa95]. Vaihtoehtoisia menetelmiä GA:lle ovat perinteiset suorahakumenetelmät, mm. polytooppihaku. Vahvoja kilpailijoita ovat myös jäähdytysmenetelmät (*simulated annealing*), jotka perustuvat tilastollisessa fysiikassa tutkittuihin jäähdytysilmiöihin.

Mitchellin tiivis katsaus geneettisiin algoritmeihin on hyvin ajan tasalla [Mic98]. Hyödyllisiä ovat myös Hollandin, Goldbergin, Davisin, Michalewiczin ja Schwefelin teokset [Hol92, Gol89, Dav91, Mic96, Sch95].

CSC on julkaissut optimointia ja geneettisiä algoritmia käsitteleviä teoksia [HKR93, Haa95]. Fortran 90/95 -oppikirjasta löytyy differentiaalievoluution toteutus [HRR98]. Matemaattisia ohjelmistoja on esitelty teoksessa [Haa98].

Geneettisiä algorimeja käsitellään mm. Web-palvelimessa *The Hitch-Hiker's Guide to Evolutionary Computation*:

`http://surf.de.uu.net/encore/www/top.htm`

Myös arkistosta *The Genetic Algorithms Archive* löytyy tietoa menetelmistä:

`http://www.aic.nrl.navy.mil:80/galist/`

Kirjallisuutta

[Dav91] Lawrence Davis, toim. *Handbook of Genetic Algorithms*. Van Hostrand Reinhold, 1991.

[Gol89] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[Haa] Juha Haataja. *Examples of using genetic algorithms*. Web address `http://www.csc.fi/Movies/GA.html`.

[Haa95] Juha Haataja. *Optimointitehtävien ratkaiseminen*. CSC, 1995.

[Haa98] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[Haa99] Juha Haataja. Using Genetic Algorithms for Optimization: Technology Transfer in Action. Teoksessa: K. Miettinen, M. M. Mäkelä, P. Neittaanmäki ja J. Periaux, toim., *Evolutionary Algorithms in Engineering and Computer Science*. Wiley, 1999.

[HKR93] Juha Haataja, Juhani Käpyaho ja Jussi Rahola. *Numeeriset menetelmät*. CSC, 1993.

[Hol92] John H. Holland. *Adaptation in Natural and Artificial Systems*. Bradford Books, 1992.

[HRR98] Juha Haataja, Jussi Rahola ja Juha Ruokolainen, toim. *Fortran 90/95*. CSC, 1998.

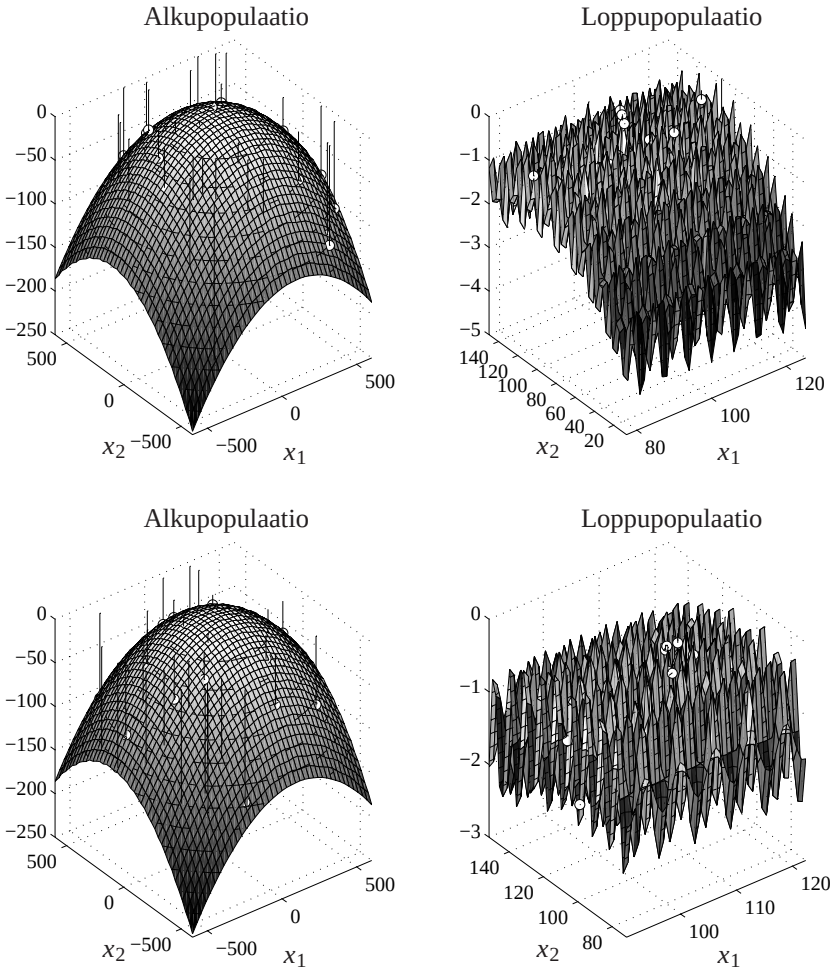
[Mic96] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[Mic98] Melanie Michell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[Sch95] Hans-Paul Schwefel. *Evolution and Optimum Seeking*. Wiley, 1995.

Menetelmä	*Paras*	*Huonoin*	*Keskiarvo*	*Onnistumispros.*
GA	-3.6e-7	-0.0099	-0.0033	63%
Satunnaisotanta	-0.033	-0.57	-0.23	0%

Taulukko 2: *Griewankin tehtävän ratkaiseminen eri menetelmillä.*



Kuva 4: *Geneettisen algoritmin käyttö maksimointiin. Kohdefunktio on esitetty pintakaaviona ja populaation alkiot on merkitty symbolilla ∘. Menetelmää toistettiin sata kertaa. Ylhäällä oikealla menetelmä löytää optimikohdan* $(100, 100)^T$*. Huonoin tapaus näkyy alhaalta oikealta: maksimikohta löytyy pisteestä* $(106.3, 100.0)^T$*.*

Tavalliset differentiaaliyhtälöt Matlabilla

Ville Savolainen
Ville.Savolainen@csc.fi

Matlab on interaktiivinen sovellusohjelmisto numeeriseen laskentaan ja visualisointiin. Matlab soveltuu yksinkertaisten ongelmien ratkaisuun ja kokeiluihin ikään kuin kynän ja paperin jatkeena. Sillä kannattaa myös prototyypittää laajempia ongelmia ennen tehokkaampaa ja mahdollisesti yksityiskohtaisempaa toteutusta käännettävällä ohjelmointikielellä.

Matlabin peruskäyttöä on käsitelty @CSC:n numerossa 1/1998 [VS98]. Lisätietoja Matlabin käytöstä saa mm. oppaista [Mat96a, Mat96b].

Matlabin peruspaketilla voidaan ratkaista tavallisia differentiaaliyhtälöryhmiä alkuarvotehtävissä. Ajan ollessa riippumaton muuttuja näissä tunnetaan systeemin alkutila tietyllä ajanhetkellä t_0 sekä systeemin muutosta kuvaavat differentiaaliyhtälöt (DY:t). Systeemin tila halutaan määrätä hetkellä t. Nämä ovat verrattain yksinkertaisia ja Matlabilla helposti formuloitavia tehtäviä. Fysikaalisella ja matemaattisella intuitiolla sekä symbolisten ohjelmistojen (Mathematica, Maple) avulla näitä voidaan käyttää monien fysikaalisten ongelmien alustavaan tarkasteluun.

Reuna-arvotehtävien ratkaiseminen on huomattavasti vaikeampaa eikä alkuarvotehtävien tapaan automatisoitavissa Matlabilla.

Monet fysiikan tärkeät DY:t ovat osittaisdifferentiaaliyhtälöitä (ODY), joissa riippumattomia muuttujia on useampia, tyypillisesti kolme avaruuden dimensiota. ODY:jä voidaan esim. sopivien symmetrioiden vallitessa palauttaa tavallisiksi DY-ryhmiksi.

Matlabin elementtimenetelmään perustuvalla PDE-työkalupakilla voidaan ratkaista kahden riippumattoman muuttujan ODY-ryhmiä. Työkalupakki sisältää graafisen esikäsittelijän tehtävän muotoilemiseksi sekä valmiit käyttöliittymät mm. eräisiin lujuuslaskentaan, sähkömagneettisiin kenttiin ja lämmönsiirtoon liittyviin tehtävätyyppeihin. PDE-työkalupakkia on esitelty @CSC:n numerossa 4/1996 [YL96].

Differentiaaliyhtälöryhmä määritellään *kankeaksi*, kun Jacobin matriisin ominaisarvojen suurin ja pienin reaaliosa eroavat useita kertaluokkia. Kankeissa tehtävissä systeemin kannalta usein merkityksettömillä, nopeasti häviävillä osilla, on askelpituutta rajoittava vaikutus. Tällaisia DY-ryhmiä syntyy esim. mallinnettaessa kemiallisia reaktioita, joissa eri reaktioiden tyypilliset aikaskaalat eroavat huomattavasti. Kankeiden tehtävien ratkaisu ei-kankeille tehtäville tarkoitetuilla ratkaisijoilla on vaikeaa.

Tässä artikkelissa käsittelemme tavallisten DY-ryhmien alkuarvotehtävien ratkaisua Matlabilla kahden esimerkin valossa. Ensimmäinen on yksinkertainen ei-kankea tehtävä, ja toinen klassinen kankea ongelma. Molempiin liittyvät koodit löytyvät WWW-osoitteesta

```
http://www.csc.fi/programming/
  examples/matlab_dy/
```

Differentiaaliyhtälöiden numeerista ratkaisemista käsitellään tarkemmin teoksissa [HKR93, HW87, HW91, Str86].

Resepti alkuarvotehtäville

Näin ratkaiset helposti ja nopeasti alkuarvotehtäviä Matlabilla:

1. Kirjoita tavallinen differentiaaliyhtälö tai -yhtälöt ensimmäisen kertaluvun yhtälöiden ryhmänä:

   $$y_i'(t) = f(t, y_1, \ldots, y_n), i = 1, \ldots, n.$$

   Yleinen n:nnen kertaluvun differentiaaliyhtälö voidaan aina palauttaa n:ksi ensimmäisen kertaluvun differentiaaliyhtälöksi. Alkuarvotehtävässä on annettu $y_i(t_0) = y_{i0}$.

2. Kirjoita DY-ryhmä M-tiedostoon funktioksi, jonka tulosargumentti on vektori $\vec{y}'$ ja muodollisina syöttöparametreina ajanhetki t ja tuntematon vektori $\vec{y}$.

3. Matlab kutsuu DY-ryhmän määrittelevää funktiota käyttäjän valitsemasta ratkaisijafunktiosta `odexxx`. Ratkaisijafunktion kutsussa määritellään integroitava aikaväli ja alkuarvot. Kutsumuoto on ratkaisijasta riippumaton, ja käyttäjä voi funktion nimeä muuttamalla kokeilla eri menetelmien tehokkuutta annettuun tehtävään.

Hölkkääjä ja koira

Seuraava esimerkki, joka on mukaillen lainattu kirjasta [GH95], on ei-kankea alkuarvotehtävä ensimmäisen asteen tavalliselle DY-parille. Tehtävä on hyvin yksinkertainen, mutta esittelee kaikki alkuarvotehtävän ratkaisemisen askeleet käytännössä.

Olkoon $\vec{X}(t)$ määrätty hölkkääjän rata tasossa. Hetkellä t_0 vihainen koira näkee hölkkääjän ja hyökkää tämän kimppuun. Koiran paikka $\vec{x}(t)$ on ratkaistava, kun tiedetään, että koiran nopeus $\vec{v} = \dot{\vec{x}}$ on itseisarvoltaan vakio w ja suoraan kohti juoksijaa, ja koiran alkupaikka $\vec{x}(t_0)$ tunnetaan. Juoksija ei hädissään reagoi koiran rataan, vaan $\vec{X}(t)$ on annettu ja ainoastaan $\vec{x}(t)$ tuntematon. Toisin sanoen

$$\begin{aligned} |\dot{\vec{x}}(t)| &= w, \\ \dot{\vec{x}}(t) &= \lambda(\vec{X}(t) - \vec{x}(t)), \end{aligned}$$

mistä λ voidaan sieventää pois ja ratkaistavaksi differentiaaliyhtälöpariksi saadaan:

$$\dot{\vec{x}}(t) = \frac{w}{|\vec{X}(t) - \vec{x}(t)|} \left(\vec{X}(t) - \vec{x}(t) \right).$$

Muuttujat on esitetty vektorimuodossa $\vec{x} = (x, y)$ ja $\vec{X} = (X, Y)$, kuten niitä Matlabilla käsitellään.

Ratkaisu Matlabilla

Yhtälöryhmä on valmiiksi ensimmäistä astetta, joten määritellään sitä vastaava M-funktio `hurtta` tiedostoon `hurtta.m`

```
function dx = hurtta(t,x)
global w
X = juoksija(t);
h = X-x; nh = norm(h);
if nh < 1e-3
   dx = NaN*h
else
   dx = (w/nh)*h;
end
```

Koiran vakionopeus `w` välitetään globaalilla muuttujalla. Kun koiran ja juoksijan välinen etäisyys saavuttaa riittävän pienen arvon, aikaintegrointi lopetetaan `NaN`-arvoon (*Not-a-Number*). Ilman ehtolausetta singulariteetti saa Matlabin vaikeuksiin, eikä tehtävä ole enää fysikaalisestikaan mielekäs koiran saatua juoksijan hampaisiinsa. Juoksijan määrätty rata $\vec{X}(t)$ annetaan erillisessä M-tiedostossa `juoksija.m`. Olkoon ensimmäisessä tapauksessa määritelty juoksijan nopeus vakio $\dot{\vec{x}} = (8, 0)$ pitkin x-akselia:

```
function s = juoksija(t);
s = [8*t; 0];
```

Pääohjelma `puskii.m` näyttää seuraavalta:

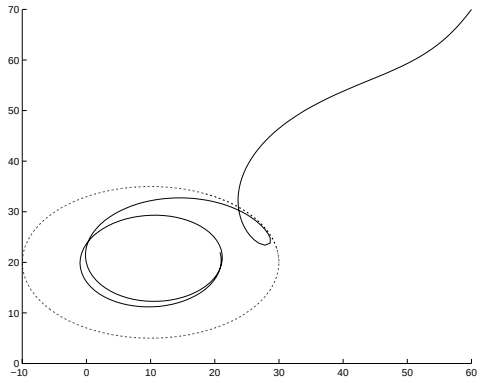
```
global w; w = 10;
x0 = [60; 70];
tspan = 0:0.1:20;
[t,x] = ode23('hurtta',tspan,x0);
steps = size(find(x(:,1)<1e6),1);
disp(t(steps))
hold on
plot(x(1:steps,1),x(1:steps,2))
J = zeros(steps,2);
for i = 1:steps,
  J(i,:) = juoksija(t(i))';
end
plot(J(:,1),J(:,2),':')
```

Tässä määriteltiin koiran nopeus `w`, lähtöpiste `x0` ja integrointiväli `tspan`. Vektorin `tspan` jako on merkityksellinen vain tulostuksen kannalta eikä vaikuta aikaintegroinnissa käytettyyn askellukseen. Määrittely `tspan = [0 20]` olisi ratkaisun kannalta samanarvoinen. Funktiolla `ode23` valitaan eksplisiittinen Runge-Kutta $(2,3)$ -ratkaisija ja lopuksi piirretään koiran (yhtenäinen viiva) ja juoksijan (katkoviiva) radat. Lisäksi tulostetaan hetki, jolloin koira saavuttaa juoksijan tai integrointi lopetetaan.

Ajamalla ohjelma komennolla `puskii` nähdään, että koira saavuttaa itseään hitaamman juoksijan ennen hetkeä $t = 12.3$. Tehtävää on helppo varioida juoksijan radan ja nopeuden tai koiran nopeuden suhteen. Esimerkiksi hölkkääjän juostessa parametrisoitua ellipsirataa koiraa nopeammin, Laika-koira juuttuu lopulta pienemmälle stationääriselle ellipsiradalle. Määrittelyllä

```
function s = juoksija(t);
s = [10+20*cos(t); 20+15*sin(t)];
```

radat näyttävät seuraavalta:



Brysseleraattori

Brysseleraattori on nimensä mukaisesti kankea ja potentiaalisesti suuri systeemi, esimerkissämme $2N$ differentiaaliyhtälön ryhmä. Brysseleraattori kuvaa kuuden kemiallisen yhdisteen kinetiikkaa ja yksiulotteista diffuusiota neljässä reaktiossa, ja ongelmaa kuvaa siis alunperin ODY-ryhmä. Tehtävä on esitelty teoksissa [HW87, HW91]. Näissä esitetyin yksinkertaistuksin ratkaistavaksi jää kaksi yhdistettä u ja v, jotka on diskretoitu paikan suhteen N pisteen hilassa $x_i = i/(N+1)$, $i = 1, \ldots, N$:

$$\begin{aligned} \dot{u}_i &= 1 + u_i^2 v_i - 4u_i + c(u_{i-1} - 2u_i + u_{i+1}), \\ \dot{v}_i &= 3u_i - u_i^2 v_i + c(v_{i-1} - 2v_i + v_{i+1}), \end{aligned}$$

missä $c = \alpha(N+1)^2$. Yhtälöryhmä on sitä kankeampi, mitä suurempi N on. Alkuarvot ovat

$$u_i(0) = 1 + \sin(2\pi x_i),$$
$$v_i(0) = 3.$$

Reuna-arvot koskevat hilan ulkopuolisia pisteitä:

$$u_0(t) = u_{N+1}(t) = 1,$$
$$v_0(t) = v_{N+1}(t) = 3.$$

Ratkaisu Matlabilla

DY-ryhmän määrittely on suoraviivaista ja toteutettu oppaasta [Mat96b] muokatussa funktiossa `bryssel.m` käyttäen hyväksi DY-ryhmän harvaa rakennetta. Hilapisteiden lukumäärä N on määritelty globaaliksi muuttujaksi, ja muuttujavektori on indeksoitu järjestyksessä $\vec{y} = (u_1, v_1, \ldots, u_N, v_N)$. Päätepisteissä $i = 1$ ja $i = N$ diffuusiotermeissä olevat reunaehdot on sijoitettu erikseen määrittelyihin.

Ratkaisijaa kutsutaan pääohjelmalla, joka määrittelee systeemin koon, tarkasteltavan aikavälin ja alkuarvot:

```
>> global N; N = 20;
>> tspan = [0; 10];
>> x0 = [1+sin((2*pi/(N+1))*(1:N)); ...
3+zeros(1,N)];
>> x0 = x0(:);
>> options = odeset('Vectorized', ...
'on','Jacobian','on');
>> [t,x] = ode15s('bryssel', ...
tspan,x0,options);
>> plot(t,x)
```

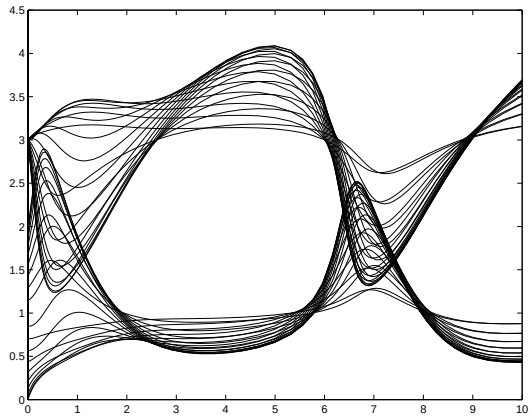
Pääohjelma löytyy tiedostosta `mainbrs.m`. Alkuarvot määritellään ensin $2 \times N$-matriisina ja muunnetaan sitten vektoriksi, joka on indeksoitu edellä esitetyllä tavalla. Funktiolla `ode15s` ratkaisijaksi on valittu Matlabin kankeiden tehtävien moniaskelmenetelmäkokoelma.

Funktiolla `odeset` annetaan `options`-argumentille `Jacobian` arvo `on`. Tällä ilmoitetaan ratkaisijafunktion kutsussa, että DY-ryhmän Jacobin matriisi määritellään analyyttisesti tiedostossa `bryssel.m`. Tämä on mahdollista vain kankeille tehtäville ja voi olla välttämätöntä yhtälöryhmän ratkaisemiseksi tai nopeuttaa sitä. Derivaattojen analyyttisessa laskemisessa voidaan käyttää apuna esimerkiksi *Symbolic Math* -työkalupakkia tai symbolisen laskennan Mathematica-ohjelmistoa.

Jacobin matriisin esittämisessä on lisäksi käytetty hyväksi sen nauharakennetta ja Matlabin harvamatriisioperaatiota `spdiags`. @CSC:n numeron 3/1998 Matlab-niksipalstalla esitetään, kuinka kyseiset analyyttiset lausekkeet lasketaan Mathematicalla ja palautetaan Matlabiin. Palstalla annetaan myös vinkkejä nauha- ja harvamatriisien käsittelyyn Matlabilla.

Oletusarvoisesti Matlab approksimoi DY-ryhmän Jacobin matriisia numeerisesti, eikä käyttäjän tällöin tarvitse puuttua sen laskentaan. Jos Jacobin matriisi on harva, kuten brysseleraattorin tapauksessa, ohjelmoijan kannattaa kuitenkin käyttää valitsinta `Jpattern` ja välittää nollasta poikkeavien alkioiden indeksit ratkaisijalle.

Yhdisteiden pitoisuudet ajan funktiona hilapisteiden lukumäärällä $N = 20$ on esitetty alla:



Ratkaisijafunktiot

Sokratesta mukaillen: Tunne yhtälösi!

Ratkaisijafunktio tulee valita tehtävän mukaisesti. Jos tehtävä ei ole kankea, ensiksi kannattaa kokeilla eksplisiittisiä Runge-Kutta-ratkaisijoita `ode45` ja `ode23`. Moniaskelmenetelmistä ei-kankeille tehtäville on tarjolla Adams-Bashforth-Moulton-ratkaisija `ode113`, joka saattaa olla edellisiä tehokkaampi.

Kankeille tehtäville on kaksi ratkaisijafunktiota, `ode15s` ja `ode23s`. Funktio `ode15s` tarjoaa valikoiman moniaskelmenetelmiä 1–5 asteen NDF- ja BDF-ratkaisijoista. Käyttäjän tarvitsee ainoastaan määritellä korkein sallittu ratkaisijan aste valitsimella `MaxOrder` sekä ilmoittaa, käytetäänkö BDF-menetelmiä valitsimella `BDF`. Näiden oletusarvot ovat `5` ja `off`. Matalamman asteen ratkaisijat voivat olla joissakin tapauksissa stabiilimpia. Funktio `ode23s` käyttää toisen asteen modifioitua Rosenbrock-ratkaisijaa, joka on yksiaskelmenetelmä.

Jos tehtävä tiedetään kankeaksi tai ei-kankeat ratkaisufunktiot eivät siihen pure, ensiksi kannattaa kokeilla oletusarvoista `ode15s`-kutsua. Ratkaisijafunktio `ode23s` voi joskus, erityisesti karkeilla toleranssikriteereillä, olla tehokkaampi.

Kankeita ratkaisijoita voidaan kutsua samalla syntaksilla kuin ei-kankeita ratkaisijoita, mutta tehokkuus voi parantua huomattavasti välittämällä niille

lisätietoja ongelmasta `options`-argumentilla. Argumenttiluokista tärkeimmät koskevat aihepiirejä virherajat, Jacobin matriisin generointi, askelpituus, massamatriisi ja tapahtumat (*events*).

Lisätietoja

Matlabin käytössä saa apua komennoilla `help matlab` ja `matlabdoc` sekä Matlabin komennolla `help`. Matlabin käsikirjat löytyvät Matlab-komennolla `helpdesk`.

Matlabin esimerkkejä kankeista differentiaaliyhtälöistä löytyy hakemistosta

```
/v/osf40_alpha/appl/math/matlab/
  matlab5.1/toolbox/matlab/demos
```

CSC:n oppaassa [JH98a] käsitellään myös Matlabin käyttöä. Tästä @CSC:n numerosta alkaa myös Matlabin käyttöön vinkkejä antava artikkelisarja [JH98b].

Kirjallisuutta

[GH95] Walter Gander ja Jiří Hřebíček. *Solving Problems in Scientific Computing Using Maple and MATLAB*. Springer, 1995.

[HW87] E. Hairer, S. P. Nørsett ja G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems*. Springer-Verlag, 1987.

[HW91] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II, Stiff and Differential-Algebraic Problems*. Springer-Verlag, 1991.

[JH98a] Juha Haataja (toim.). *Matemaattiset ohjelmistot*. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`. CSC – Tieteellinen laskenta Oy, 1998.

[JH98b] Juha Haataja. *Matlab 5.x — kysymyksiä ja vastauksia*. @CSC, 2/1998.

[HKR93] Juha Haataja, Juhani Käpyaho ja Jussi Rahola. *Numeeriset menetelmät*. CSC – Tieteellinen laskenta Oy, 1993.

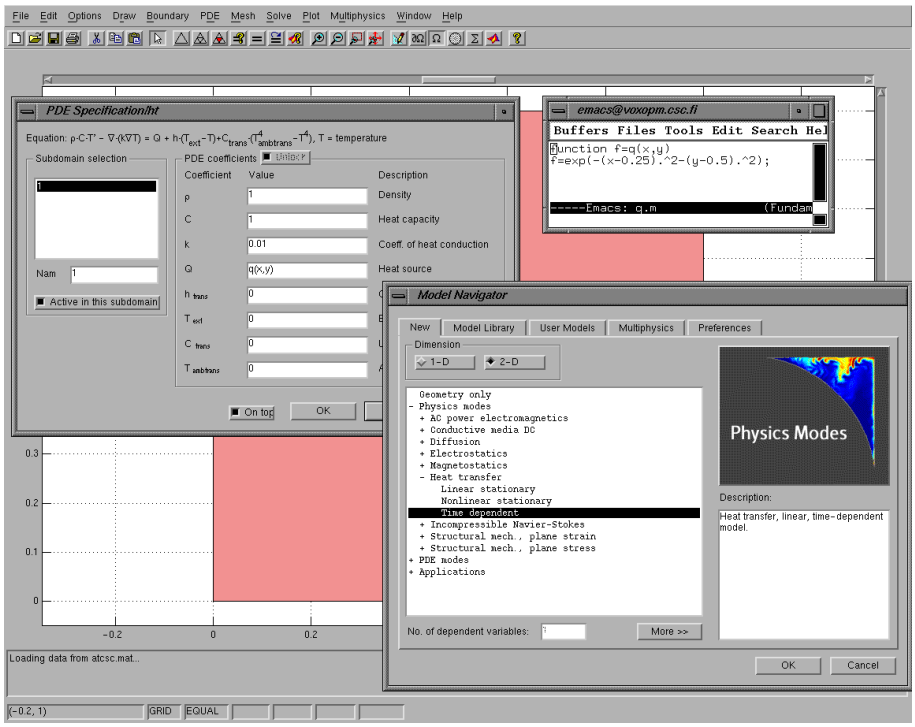
[YL96] Yrjö Leino. *Millaisia yhtälöitä voit ratkoa PDE Toolboxilla?* @CSC, 4/1996.

[Mat96a] The MathWorks, Inc. *Getting Started with Matlab, Version 5*, 1996.

[Mat96b] The MathWorks, Inc. *Using Matlab, Version 5*, 1996.

[Str86] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[VS98] Ville Savolainen. *Tehokas ohjelmointi Matlabilla*. @CSC, 1/1998.



Kuva 1: Femlabin pääikkuna, *Model Navigator* -ikkuna ja yhtälön kertoimien valintaikkuna.

FEMLAB monifysikaaliseen mallinnukseen

Jussi Heikonen
Jussi.Heikonen@csc.fi

FEMLAB on MATLABiin perustuva graafisella käyttöliittymällä varustettu osittaisdifferentiaaliyhtälöratkaisija, joka soveltuu erinomaisesti myös monifysikaalisten tehtävien käsittelyyn. Kuten nimestä voi päätellä, yhtälöiden numeeriseen ratkaisemiseen käytetään elementtimenetelmää.

Toistaiseksi FEMLABilla voi ratkaista yksi- ja kaksiulotteisissa alueissa määriteltyjä tehtäviä, mutta ohjelman 3D-versio on jo kehitteillä. Laskenta-alueen eri osissa voi ratkaista eri yhtälöryhmiä, mikä on oleellista monifysikaalisessa mallinnuksessa.

Käyttöä helpottaa se, että FEMLABissa on valmiina yhtälöitä moniin eri sovelluksiin, kuten sähkömagnetiikkaan, diffuusioilmiöihin, lämmönsiirtoon, virtausmekaniikkaan ja rakenneanalyysiin. Näin käyttäjän tarvitsee vain määritellä tehtävään liittyvät fysikaaliset vakiot ja parametrit. Luonnollisesti on mahdollista ratkaista myös yleisiä osittaisdifferentiaaliyhtälöryhmiä.

FEMLAB on asennettu Cedarille. Tällä hetkellä käyttäjiä voi olla korkeintaan viisi yhtä aikaa, mutta tarpeen vaatiessa voidaan ohjelmalle hankkia useamman käyttäjän lisenssi.

Seuraavassa esitellään FEMLABin käytön alkeet ja annetaan ohjeita lisätietojen hakemiseksi.

FEMLABin peruskäyttö

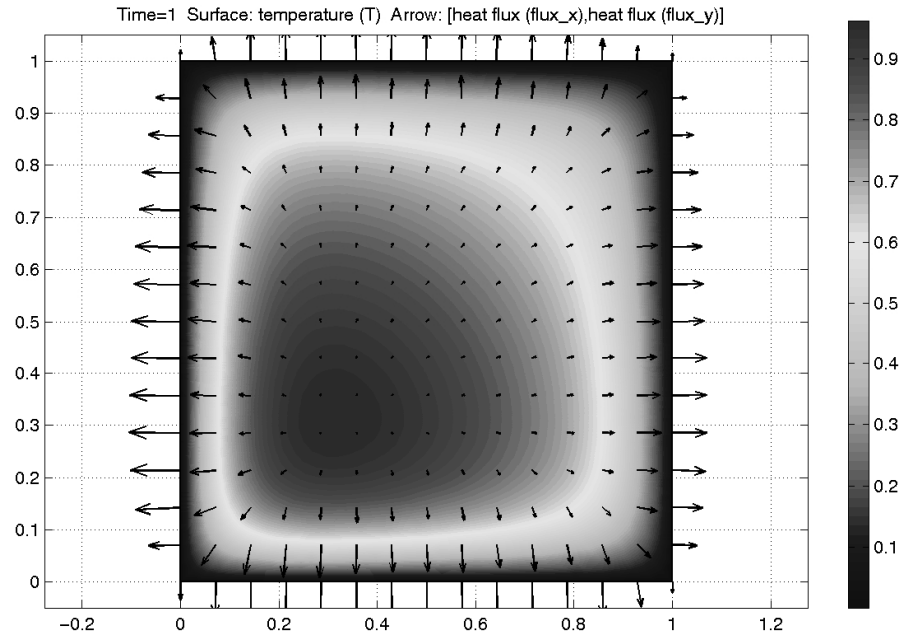
Istunto aloitetaan käynnistämällä MATLAB:

```
cedar% matlab
```

Tämän jälkeen FEMLAB käynnistetään MATLABista komennolla

```
>> femlab
```

Käynnistyessään FEMLAB avaa kaksi ikkunaa. Suuremmassa pääikkunassa on valikoita ja painikkeita mm. laskenta-alueen, reunaehtojen ja yhtälöiden määritystä, verkongenerointia, tehtävän ratkaisemista ja tulosten jälkikäsittelyä varten. Liikkeelle kuitenkin lähdetään pienemmästä *Model Navigator* -ikkunasta, jossa valitaan ratkaistavat yhtälöt painikkeen *New* tai *Multiphysics* kautta.



Kuva 2: Esimerkkitehtävän ratkaisu.

Tämän ikkunan saa näkyviin myös pääikkunan valikon *File* komennon *Open* avulla. Kuvassa 1 on taustalla pääikkuna painikkeineen ja oikeassa alakulmassa näkyy *Model Navigator* -ikkuna.

FEMLABin peruskäyttö on yksinkertaista. Ratkaistaan seuraava lämmönjohtumistehtävä yksikköneliössä:

$$\frac{\partial T}{\partial t} = -\nabla \cdot (-0.01 \nabla T) + Q; \;\; x, y \in (0,1); \;\; t \in (0,1],$$

missä lämpölähde on muotoa

$$Q = \exp\{-(x-0.25)^2 - (y-0.25)^2\}.$$

Alueen reunalla $T = 0$ ja hetkellä $t = 0$ koko alueessa $T = 0$.

Valitaan ikkunasta *Model Navigator New* ja kaksoisnäpäytetään listasta kohtaa *Physics modes*, jolloin näkyviin tulee luettelo valmiista sovelluksista. Valitaan edelleen *Heat transfer* ja kohta *Time dependent*. Näppäin *OK* varmistaa valinnan ja sulkee ikkunan.

Nyt tehtävän voi määritellä ja ratkaista etenemällä pääikkunan yläreunan valikoissa vasemmalta oikealle: *Draw*, *Boundary*, . . . , *Plot*.

Piirretään ensin yksikköneliö joko valikon *Draw* komennoilla tai ikkunan vasemman reunan painikkeilla, jotka ovat näkyvissä piirtomoodissa. Hilaviivoihin ym. piirtämiseen liittyviin asioihin voi vaikuttaa sekä valikon *Draw* komennolla *Properties* että tietyillä valikon *Options* komennoilla.

Reunaehdot määritellään valikon *Boundary* komennolla *Specify Boundary Conditions*. . . Käsiteltävän reunan voi valita hiirellä joko piirroksesta tai komennon avaamasta ikkunasta kohdasta *Boundary selection*. Valitulla reunalla voi kiinnittää lämpötilan tai lämpövuon. Lämpövuon tapauksessa voi käyttää myös konvektio- tai säteilytyyppistä reunaehtoa. Valitaan kaikille reunoille lämpötilaksi nolla.

Reunaehtojen jälkeen määrätään itse yhtälön kertoimet valikkokomennolla *PDE/PDE Specification*. . . , joka avaa kuvan 1 vasemmassa yläkulmassa näkyvän yhtälöikkunan. Alueen voi jälleen valita joka kuvasta tai valikosta. Tyydytään muuten oletuskertoimiin, mutta asetetaan lämmönjohtumiskertoimeksi $k = 0.01$ ja lämpölähteeksi

$$Q = \exp\{-(x-0.25)^2 - (y-0.25)^2\}.$$

Reunaehdoissa ja kertoimissa on käytettävä MATLABin mukaista syntaksia. Niinpä lämpölähteen Q kohdalle kirjoitetaan

```
exp(-(x-0.25).^2-(y-0.25).^2)
```

Jos reunaehdot tai yhtälön kertoimet ovat monimutkaisia tai niissä toistuu samoja lausekkeita, kannattaa käyttää apuna MATLABin m-funktioita. Äskeinen lämpölähde voidaan toteuttaa myös kirjoittamalla tiedosto `q.m` seuraavasti:

```
function f=q(x,y)
f=exp(-(x-0.25).^2-(y-0.25).^2);
```

Nyt kertoimen Q kohdalle kirjoitetaan luonnollisesti `q(x,y)`. Kuvassa 1 on tehty näin ja mukana on myös tiedosto `q.m`. Jotta FEMLAB käyttäisi ajan tasalla olevaa versiota funktiosta, on tiedoston editoimisen jälkeen aina syytä poistaa funktio muistista. Tässä tapauksessa siis annetaan MATLABissa mahdollisten muutosten jälkeen komento

```
>> clear q
```

Reunaehdoissa ja kertoimissa käytettävissä olevia vakioita voi määritellä valikon *Options* komennolla *Add/Edit Variables...*

Elementtimenetelmän tarvitsema verkko luodaan valikon *Mesh* komennolla *Initialize Mesh*. Verkkoa voi tarvittaessa tihentää komennolla *Refine Mesh*. Paikallisen tihennyksen voi tehdä komennolla *Refine Selected Triangles* ja valitsemalla hiirellä tihennettävän alueen. Tihennys tapahtuu valitsemalla komento uudelleen. Tihennetään hilaa kerran tasaisesti ja edelleen vasemmasta alakulmasta lähteen luota.

Ennen tehtävän ratkaisemista on vielä annettava alkuehto valikon *Solve* komennolla *Specify Initial Cond's*. Valitaan alkuehdoksi nolla. Kohdasta *Solver Parameters / Timestepping* voi valita alku- ja loppuajan sekä halutut välitulokset. Valitaan oletusarvo, eli ratkaistaan tehtävä ajanhetkillä nollasta yhteen ja talletetaan tulokset kymmenesosasekunnin välein. Ratkaistaan tehtävä komennolla *Solve Problem*.

Kun tehtävä on ratkaistu, tulokset voi visualisoida monin tavoin avaamalla valikosta *Plot* kohdan *Parameters...* Kuvassa 2 näkyy harmaasävyillä esitetty lämpötilajakauma ja nuolilla kuvattu lämpövuo hetkellä $t = 1$. Kuva on tulostettu valikon *File* komennolla *Print...* Ajasta riippuvien tehtävien tuloksista voi tehdä myös animaatioita.

Monifysiikkatehtävät

Monifysiikkatehtävien määrittelemisessä ja ratkaisemisessa käydään läpi samat vaiheet kuin yllä, mutta mallissa on useampia yhtälöitä ja usein on tarpeen ratkaista eri yhtälöt alueen eri osissa.

Malliin voi valita useampia yhtälöitä *Model Navigator* -ikkunasta kohdasta *Multiphysics*. Mallinnettavia ilmiöitä voi lisätä tai poistaa valitsemalla ne hiirellä ja siirtämällä niitä keskellä olevilla nuolipainikkeilla. Vastaavan ikkunan saa näkyviin pääikkunan valikon *Multiphysics* komennolla *Add/Edit Modes...*

Yhtälö, jonka reunaehtoja tai kertoimia halutaan asettaa, valitaan nyt joko suoraan *Multiphysics*-valikon alareunasta tai komennolla *Set Application Mode...* Laskenta-alue kullekin yhtälölle valitaan kertoimia määrättäessä. Tällaisissa tapauksissa alue on luonnollisesti piirrettävä siten, että se koostuu erillisistä osista.

Käyttö MATLABin komentoriviltä

Yksinkertaisia tehtäviä ratkaistaessa on mukavinta käyttää graafista käyttöliittymää, mutta toisinaan on kätevämpää käyttää FEMLABin komentoja suoraan MATLABista ja kirjoittaa sopivia m-tiedostoja. Monen kirjastossa *Model Library* olevan esimerkin kohdalla on annettu myös vastaavat MATLABista käsin annettavat komennot.

Graafisella käyttöliittymällä laadittu malli ja sen ratkaisu on mahdollista siirtää MATLABin työtilaan joko kokonaan tai osittain valikon *File* komennolla *Export to Workspace*. Käyttäjä voi esimerkiksi tehdä geometrian ja elementtiverkon graafisella käyttöliittymällä, siirtää ne MATLABiin ja kirjoittaa sopivan m-tiedoston yhtälöiden määrittelyä ja ratkaisua varten.

Käyttöliittymällä tehdyn työn voi tallentaa myös suoraan m-tiedostoon, jota voi tarvittaessa muuttaa. Jos tiedoston haluaa lukea takaisin käyttöliittymään, sen muoto on säilytettävä. Suoraa MATLAB-käyttöä varten tiedostoa voi muunnella vapaammin.

Puutteita

Vaikka FEMLABia on hyvin helppo käyttää, toivomisen varaa on vielä. Valmiita yhtälöitä voi käyttää vain karteesisessa koordinaatistossa. Jos haluaa mallintaa jotain esimerkiksi sylinterikoordinaatistossa, on käytettävä yleisessä muodossa annettuja yhtälöitä ja valittava kertoimet sopivasti.

FEMLABissa on vain yksi elementtityyppi, kolmioelementti ensimmäisen asteen kantafunktiolla. Tuloksen tarkkuutta voi siten parantaa ainoastaan hilaa tihentämällä.

Myös virheilmoitukset voivat olla varsin epämääräisiä.

Dokumentaatio ja esimerkkejä

FEMLABin täydelliseen dokumentaatioon pääsee käsiksi valikkokomennolla *Help / FEMLAB Help Desk (HTML)...* Linkin *Table of Contents* takaa löytyvät oppaat *User's Guide and Introduction*, *Model Library*, *Reference Guide*, *The Graphical User Interface* ja *Function Reference*.

Varsinkin monipuolisia ja yksityiskohtaisia esimerkkejä sisältävä *Model Library* on erittäin hyödyllinen. Esimerkkeihin pääsee myös suoraan käsiksi *Model Navigator* -ikkunan kohdasta *Model Library*.

FEMLABin kotisivu on osoitteessa

```
http://www.femlab.com
```

Tehokas ohjelmointi Matlabilla

Ville Savolainen
Ville.Savolainen@csc.fi

Matlab on alunperin matriisien käsittelyyn tehty laskentaympäristö, joka on kehittynyt yleiskäyttöiseksi työkaluksi numeeriseen laskentaan, visualisointiin ja mallittamiseen. Matlab on interaktiivinen sovellusohjelmisto, jolla voi ratkoa yksinkertaisia ongelmia nopeasti tai prototyypittää laajempia ongelmia ennen tehokkaampaa toteutusta käännettävällä ohjelmointikielellä.

Matlabia voidaan laajentaa kirjoittamalla aliohjelmia (*scriptejä*) tai uusia funktioita M-tiedostoihin tai kääntämällä ulkopuolisia Fortran- tai C-ohjelmia MEX-tiedostoihin. Matlabissa on myös sovelluskohtaisia *toolbox*-funktiopaketteja eli työkalupakkeja. Symboliseen laskentaan Matlabissa ei ole mahdollisuutta.

Matlabin ohjelmointikielen tulkattavuuden vuoksi ohjelmointityylin tehokkuuteen on kiinnitettävä erityistä huomiota. Tässä artikkelissa annetaan yksinkertaisia vinkkejä Matlab-koodin tehostamiseksi.

Matlabin versio 5.1 on käytettävissä CSC:n koneista Caperillä, Cypressillä ja Cypress2:lla. Version 5 uusista ominaisuuksista kerrotaan oppaassa [Mat96c]. Työkalupakit ovat käytettävissä vain Cypressillä.

Kaikki tässä artikkelissa esitettyihin esimerkkeihin liittyvät tiedostot löytyvät WWW-osoitteesta

```
http://www.csc.fi/programming/
  examples/matlab
```

Vektorointi

Matlabin perustietotyyppi on matriisi ja tämän erikoistapauksina vektori ja skalaari. Versio 5 on tuonut uusina tietotyyppeinä useampidimensioiset taulukot (eivät käytössä harvojen matriisien operaatioille), solutaulukot ja rakenteiset tietotyypit.

Matlab on tehokas matriisikieli, ja koodi on syytä kirjoittaa matriisi- ja vektorioperaatioiden avulla eli *vektoroituna*. Esim. `for`- ja `while`-silmukoiden käyttö hidastaa ohjelman suoritusta huomattavasti. Jos silmukkarakennetta joudutaan käyttämään, kannattaa sijoituslauseissa arvon saavat vektorit ja matriisit allokoida etukäteen funktiolla `zeros`, ettei Matlab joudu kasvattamaan näiden kokoa joka askeleella.

Esimerkiksi seuraavaa koodia olisi hankala toteuttaa matriisioperaatioiden avulla. Koodi laskee parametritaulukon `m` yksikköhalkaisijaisen virtasilmukan magneettikentän määrittämiseksi elliptisten ingegraalien avulla.

```
r=0.001:0.001:0.25; z=0.001:0.001:0.5;
m=zeros(250,500);
for i=1:250, for j=1:500
m(i,j)=2*r(i)./((0.5+r(i))^2+z(j)^2);
end,end
```

Saman ohjelman voi suorittaa myös ilman etukäteen tehtyä muistinvarausta. Seuraavassa kohdassa tarkastelemme, kuinka näiden ajankäyttöä voi vertailla.

Profilointi

Yksinkertaisin tapa komentojen, aliohjelmien tai funktioiden ajankäytön mittaamiseksi on ympäröidä suoritettavat lausekkeet kellokomennoilla `tic` ja `toc`. Edellisen kohdan koodinpätkä löytyy tiedostosta `mloop.m` ja ajastetaan komennolla:

```
>> tic, mloop, toc
```

Matlab 5 tarjoaa monipuolisemman profilointityökalun M-tiedostojen ajankäytön mittaamiseen. Seuraavat komennot tulostavat saman aliohjelman ajankäytön riveittäin:

```
>> profile mloop
>> mloop
>> profile report
```

Graafinen versio samasta saadaan komennoilla

```
>> t=profile
>> pareto(t.count)
```

Komentojen käyttämä aika vaihtelee hieman suorituskerrasta toiseen, ja vertailuja tehdessä on hyvä tyhjentää työmuisti käskyllä `clear all` ennen testattavan funktion suoritusta.

Muistinvaraus

Matlab käyttää eri työmuistialueita komentorivi- tai *script*-tasolla ja kunkin kutsutun M-funktion sisällä. Funktion argumentteina olevat muuttujat kopioidaan kyseisen funktion työmuistiin, jos ja vain jos niitä muutetaan funktion suorituksen aikana. Muistia voi siis säästää välttämällä suurille taulukoille `y` seuraa-

van tyyppisiä funktiokutsuja:

```
y=func(x,y)
```

Komennolla `global Y` määritellään globaali muuttuja `Y`, joka on käytettävissä kaikissa funktioissa, joissa `Y` on määritelty globaaliksi samalla käskyllä. Tällä voidaan välttää muuttujan kopionti sitä käyttävien funktioiden työmuisteihin.

Funktio käännetään M-tiedostosta *välikoodiksi*, mikä tapahtuu kutsuttaessa sitä ensimmäisen kerran istunnon aikana tai tiedostoon tehtyjen muutosten jälkeen. Funktion `func` voi kääntää etukäteen välikoodiksi työmuistiin komennolla `pcode func`, mutta tämä on harvoin tarpeen.

Komento `who` tulostaa käytetyn työmuistin muuttujat, komento `whos` lisäksi näiden koon, vaatiman tilan ja tyypin. Komennolla `clear` vapautetaan työmuistista eri valitsimilla osa tai kaikki muuttujista, globaaleista muuttujista, käännetyistä M-funktioista tai MEX-linkeistä. Myös muuttujan `x` asettaminen tyhjäksi matriisiksi komennolla

```
x=[];
```

vapauttaa muistitilan.

Komento `pack` pakkaa fragmentoituneen muistin yhtenäiseksi. Suuret matriisit kannattaa allokoida etukäteen komennolla `zeros` muistin fragmentoitumisen vähentämiseksi. Matriisit kannattaa allokoida järjestyksessä suurimmasta pienimpään.

MEX-ohjelmat

M-tiedostojen sijaan voi olla hyödyllistä käyttää ulkopuolisia Fortran- tai C-ohjelmia, jotka käännetään MEX-tiedostoiksi. Näin voidaan välttää valmiiden aliohjelmien kirjoittaminen uudestaan Matlabilla. Lisäksi MEX-ohjelmilla voidaan nopeuttaa esim. `for`-silmukoista johtuvia pullonkauloja.

MEX-tiedostoja kutsutaan Matlabista kuten M-tiedostoja, ja MEX-funktion argumentit välitetään samalla tavalla kuin Matlab-funktioiden. Käsitellään esimerkkinä MEX-funktiota `loop`, joka laskee a-säteisen silmukan, jossa kulkee virta `I`, magneettikentän. Kentän komponentit `Br` ja `Bz` lasketaan vektorien `r` ja `z` määrittämässä säännöllisessä hilassa. Funktiota kutsutaan seuraavasti:

```
>> a=0.50; I=2.5e5;
>> r=0.01:0.01;0.25; z=0.01:0.01:0.50;
>> [Br, Bz] = loop(a, I, r, z);
```

Tuloksena olevien magneettikentän komponenttien halutaan olevan matriiseja, jotka on indeksoitu siten, että alkio (i, j) vastaa pistettä $(r(i), z(j))$.

MEX-funktio toteutetaan kahdessa osassa sekä C- että Fortran-kielisenä: Varsinaisen laskennan suorittavana aliohjelmana `loop` ja *"päällystakki"*-aliohjelmana `mexFunction`, joka

1. lukee syötettävien matriisien (tai niiden erikoistapauksien vektorin ja skalaarin) koon funktioilla `mxGetM` ja `mxGetN`,
2. luo palautettavat matriisit esimerkiksi komennolla `mxCreateFull`,
3. luo osoitinmuuttujat sekä syöttö- että tulosmatriisien alkioihin funktioilla `mxGetPr` ja `mxGetPi`,
4. kutsuu suorittavaa aliohjelmaa.

Esimerkissä luetaan vektorien `r` ja `z` pituudet ja määritetään näistä riippuvat matriisien koot.

Oletusarvoisesti Matlab käyttää Caperin ja Cypressien f77- tai C-kääntäjiä, mutta myös Fortran 90 tai C++ voidaan valita. Vaikka laskennallinen aliohjelma olisi toteutettu Fortran 90 -kielisenä, päällystakkirutiini kannattaa kirjoittaa Fortran 77:lla. Tässä tapauksessa MEX-funktioita käännettäessä joudutaan muutenkin hiukan temppuilemaan.

Tarkastellaan ensin aliohjelman `mexFunction` toteutusta esimerkissä. Caperin ja Cypressien f77-kääntäjät tuntevat `%VAL`-rakenteen, joka yksinkertaistaa osoittimien käyttöä. Tämän lisäksi muuttujatyyppi `POINTER` otetaan käyttöön `#include`-lauseella.

```
#include <fintrf.h>
      SUBROUTINE mexFunction
     1   (nlhs, plhs,nrhs, prhs)
      IMPLICIT NONE
      INTEGER nlhs, nrhs
      POINTER plhs(*), prhs(*)
      INTEGER m, n
      INTEGER mxGetM, mxGetN
      POINTER mxGetPr, mxCreateFull
      POINTER ap, Ip, rp, zp, Brp, Bzp

C Vaakavektorien r ja z koko
      m = mxGetN(prhs(3))
      n = mxGetN(prhs(4))
C Luo m x n -matriisit Br ja Bz
      plhs(1) = mxCreateFull(m,n,0)
      plhs(2) = mxCreateFull(m,n,0)
C Osoittimet kaikkiin argumentteihin
      ap = mxGetPr(prhs(1))
      Ip = mxGetPr(prhs(2))
      rp = mxGetPr(prhs(3))
      zp = mxGetPr(prhs(4))
      Brp = mxGetPr(plhs(1))
      Bzp = mxGetPr(plhs(2))
C Kutsu laskennallista aliohjelmaa
      CALL loop(%VAL(Brp), %VAL(Bzp),
     1     %VAL(ap), %VAL(Ip),
     2     %VAL(rp), %VAL(zp), m, n)
```

```
      RETURN
      END
```

Ohjelma on tiedostossa `loopgate.F`. Osoitintaulukko `plhs` viittaa funktionkutsun järjestyksessä tulosargumentteihin ja `rlhs` syöttöargumetteihin. Aliohjelmaan olisi hyvä sisällyttää myös argumenttien virheellisten dimensioiden ja muiden mahdollisten ongelmatilanteiden tarkistus. Yllä `r` ja `z` on oletettu vaakavektoreiksi ilman tarkistusta.

Seuraava Fortran 90 -aliohjelma `loop` suorittaa itse laskennan. Argumenttien järjestys aliohjelmassa vastaa MEX-funktionkutsua Matlabista siten, että ensin ovat tulos- ja sitten syöttöargumentit ja lopuksi ylimääräiset argumentit, jotka on määritelty vasta aliohjelmassa `mexFunction`. Viimeksi mainittuja ovat tässä vektorien pituudet `m` ja `n`.

```
SUBROUTINE loop(Br, Bz, a, current, &
                r, z, m, n)
  IMPLICIT NONE
  INTEGER, PARAMETER :: &
    double=SELECTED_REAL_KIND(12,50)
  INTEGER, INTENT(IN) :: m, n
  REAL(KIND=double), &
    DIMENSION(m,n) :: Br, Bz
  REAL(KIND=double), DIMENSION(m) :: r
  REAL(KIND=double), DIMENSION(n) :: z
  REAL(KIND=double) :: a, current
  REAL(KIND=double) :: k2, K, E, C
  INTEGER :: i, j

  DO i = 1, m
    DO j = 1, n
      k2 = 4*a*r(i)/ &
        ((a+r(i))**2+z(j)**2)
      CALL ELLIPKE(k2, K, E)
      C = 2.d-7*current/ &
        SQRT((a+r(i))**2+z(j)**2)
      Br(i,j) = C*z(j)/r(i)* &
        (-K+(a**2+r(i)**2+z(j)**2)/ &
        ((a-r(i))**2+z(j)**2)*E)
      Bz(i,j) = C* &
        (K+(a**2-r(i)**2-z(j)**2)/ &
        ((a-r(i))**2+z(j)**2)*E)
    END DO
  END DO

END SUBROUTINE loop
```

Aliohjelma on tiedostossa `loop.f90`. Funktion argumentteina olevat reaali- ja kompleksimuuttujat on määriteltävä Matlabin konekohtaisina kaksoistarkkuuden lukuina. Tämä voidaan varmistaa käyttämällä Fortran 77 -standardista peräisin olevaa tyypiä `DOUBLE PRECISION`. Jos käytetään Fortran 90:n lajimäärittelyä `SELECTED_REAL_KIND`, on argumenttien välitys Matlabista testattava erikseen. Yllä oleva määrittely `double` toimii sekä Caperillä että Cypresseillä.

Aliohjelma `loop` käännetään Matlabista erikseen komennolla

```
>> !f90 -c loop.f90
```

ja MEX-funktio käännetään ja linkitetään suorituskelpoiseksi komennolla

```
>> mex loop.o ellipke.f loopgate.F
```

Tuloksena on Caperillä tiedosto `loop.mexaxp` tai Cypresseillä `loop.mexsg64`. Vaikka Matlabin käännöskomennossa `mex` voidaan valita kääntäjä `FC=f90`, tämä ei tunnista Fortran 90:n vapaata sarakemuotoa. Elliptiset integraalit laskeva aliohjelma `ELLIPKE` on yksinkertaisuuden vuoksi erillisessä tiedostossa `ellipke.f`. Laskentaa suorittavasta ohjelmasta voidaan myös kutsua Matlabin omia funktioita kuten `ellipke` komennolla `CALL mexCallMATLAB`. Kutsun käytöstä kerrotaan oppaassa [Mat96d].

MEX-funktiota `loop` kutsutaan edellä määritellyllä tavalla. Tuloksia voi tarkastella piirtämällä vektorikentän:

```
>> [x, y] = meshgrid(r, z);
>> quiver(x, y, Br', Bz')
```

Hakemistosta

```
/p/appl/doc/matlab/extern/examples/mex
```

löytyy Fortran 77 ja C-kielinen MEX-esimerkki `yprime`, jota on käsitelty oppaassa [Mat96d]. Näillä kielillä kirjoitettujen ohjelmien käytöstä on kerrottu myös SuperMenun numerossa 1/1995 [LR95].

Käyttäjä voi kopioida hakemistosta

```
/v/osf40_alpha/appl/math/matlab/
  latest/bin
```

konfiguraatiotiedoston `mexopts.sh` ja muuttaa käännösasetuksia pysyvästi itsellään.

Eräajokäyttö

Vaikka Matlab on suunniteltu interaktiiviseksi ohjelmistoksi, sitä tulee ajaa erätyönä raskasta laskentaa vaativissa tapauksissa vasteaikojen parantamiseksi ja käyttäjien tasapuoliseksi kohtelemiseksi. Interaktiivisille töille on asetettu seuraavat CPU-aikarajat: Caper: 1 h; Cypress: 8 h ja Cypress2: 15 min. Pitkille töille annetaan vähemmän keskusyksikköaikaa kuin vastaaville erätöille. Interaktiiviselle käytölle on myös muistirajoitukset. Molemmat rajoitukset voidaan tarkistaa komennolla `limit`.

Erätyönä suoritettavat Matlab-komennot kirjoitetaan tiedostoon, joka loppuu lauseeseen `quit`. Magneettikenttäesimerkkiä jatkaaksemme tiedostossa `cusp` oleva Matlab-ohjelma laskee ns. *cusp*-kentän, joka

luodaan kahdella samalla akselilla sijaitsevalla identtisellä virtasilmukalla, joissa kulkevat vastakkaiset virrat. Piirretty kuva kirjoitetaan EPS-tiedostoksi ohjelman lopussa komennolla

```
print -deps cusp.ps
```

Ohjelmaa kutsutaan seuraavasta eräajotiedostosta:

```
#!/bin/sh
#QSUB -q prime
#QSUB -lT 900
#QSUB -e job_err
#QSUB -o job_out
#QSUB -re
#QSUB -ro
cd $WRKDIR
matlab < cusp
```

Tässä määritellään jono `prime`, oletusarvoa pienempi varattu CPU-aika 900 s ja tulostiedostot. Ohjelman oletetaan olevan käyttäjän työhakemistossa. Tiedosto `cusp.job` lähetetään suoritukseen Caperillä ja Cypress2:lla

```
% qsub cusp.job
```

Cypressillä ei Matlab-ohjelmia voi ajaa eräajoina.

Erätöitä voidaan seurata komennoilla `cqstat` (edellyttää X-ikkunointijärjestelmää), `nqeq`, `qstat`, `top` ja `ps`. Esimerkiksi komento

```
% qstat -a -u käyttäjätunnus
```

listaa kaikki käyttäjän erätyöt.

Lisätietoja komennon `qsub` valitsimista sekä erätyöjärjestelmän seuraamisesta saa CSC:n helpjärjestelmästä, metakoneen käyttöoppaasta [KL96] sekä `man`-järjestelmästä.

Lisätietoja

Lisätietoja Matlabin käytöstä saa komennoilla `help matlab` ja `matlabdoc`, Matlabin komennolla `help` sekä oppaista [Mat96a, Mat96b]. Matlabin käsikirjat löytyvät myös Matlab-komennolla `helpdesk`.

CSC:n oppaassa [JH98] käsitellään myös Matlabin käyttöä.

Kirjallisuutta

[JH98] Juha Haataja. *Matemaattiset ohjelmistot*. CSC – Tieteellinen laskenta Oy, 1998.

[JR94] Jussi Rahola. *Matematiikan ohjelmistot ja erätyöt*. SuperMenu, 3/1994.

[KL96] Kirsti Lounamaa. *Metakoneen käyttöopas*. CSC – Tieteellinen laskenta Oy, 1996.

[LR95] Esa Lammi ja Jussi Rahola. *Omien ohjelmien mukaanotto Matlabissa*. SuperMenu, 1/1995.

[Mat96a] The MathWorks, Inc. *Getting Started with Matlab, Version 5*, 1996.

[Mat96b] The MathWorks, Inc. *Using Matlab, Version 5*, 1996.

[Mat96c] The MathWorks, Inc. *Matlab 5 New Features*, 1996.

[Mat96d] The MathWorks, Inc. *Application Program Interface Guide, Version 5*, 1996.

Ohjelmointikielen valinta ja tehokas ohjelmointi

Juha Haataja
Juha.Haataja@csc.fi

CSC:n tietokoneilla käytetään ohjelmankehitykseen seuraavia välineitä: *käännettävät kielet* (Fortran 90/95, FORTRAN 77, C, C++), *numeeriset laskentaympäristöt* (Matlab, IDL, Splus, SAS) ja *symbolinkäsittelyjärjestelmät* (Mathematica, Macsyma, Maple, Reduce).

Taulukossa 1 on kerrottu eräiden ohjelmointikielten ja matemaattisten ohjelmistojen soveltumisesta eri tehtävätyyppeihin. Erikoisohjelmistoja (ryhmäteoria jne.) ei ole mukana vertailussa.

Tehokkaan ja luotettavan ohjelmointityylin käyttö on tärkeää muidenkin kuin käännettyjen kielten yhteydessä. Kannattaa myös miettiä, voiko ainakin osan tulkittavilla kielillä suoritetuista laskuista toteuttaa käännettävillä kielillä, jotka voivat olla kymmeniä kertoja nopeampia.

Ohjelmointikielten piirteitä

Monissa sovellusohjelmistoissa on oma ohjelmointikieli matemaattisten mallien määrittelyyn ja ratkaisemiseen. Esittelen seuraavassa tärkeimpien CSC:ssä käytettyjen ohjelmointikielien ominaisuuksia.

Matlab

Matlabin perustietorakenne on matriisi eli kaksiulotteinen taulukko. Vektorit ja skalaarit ovat matriisin erikoistapauksia. Kuvankäsittelyssä matriisi tulkitaan digitaalikuvaksi. Matlabin versiossa 5 on myös mahdollisuus käsitellä useampiulotteisia taulukoita. Lisäksi voimme määritellä *solutaulukoita* (cell array) ja *rakenteisia tietotyyppejä* (structure array), joiden alkiot eivät välttämättä ole samaa tyyppiä. Myöskin olio-ohjelmointi on mahdollista.

Keskeinen Matlabin puute verrattuna symbolisen matematiikan ohjelmistoihin on se, että tietojenkäsittely on pääsääntöisesti numeerista. Esimerkiksi kuvaajan piirtämistä varten on funktiosta ensin laskettava näytematriisi.

Matlabia voidaan laajentaa kirjoittamalla Matlab-funktioita niin sanottuihin M-tiedostoihin eli `.m`-loppuisiin tiedostoihin sekä liittämällä ulkopuolisia Fortran- tai C-kielisiä aliohjelmia Matlabiin (MEX-tiedostot). Matlabissa on monipuolinen kaksi- ja kolmiulotteinen grafiikka.

Mathematica ja Maple

Mathematican ja Maplen monipuolinen perustietorakenne on Lisp-kielestä tuttu lista. Lista on hierarkkinen tietorakenne, jonka alkiot voivat olla mitä ta-

Kieli	*Numeriikka*	*Symb.käs.*	*Grafiikka*	*Tilasto*
Fortran 90	+++ ●●●	–	●●	●●
FORTRAN 77	++ ●●●	–	●●	●●●
C	+ ●●	–	●●●	●
C++	++ ●●	–	●●●	●
Matlab	+++	–	+++	++
IDL	++	–	+++	++
Splus	++	–	++	+++
SAS	++	–	+	+++
Mathematica	++	+++	+++	++
Macsyma	+	+++	++	+
Maple	++	+++	+++	++
Reduce	+	++	+	+

Taulukko 1: *Ohjelmankehitykseen käytettyjen kielten ominaisuuksia eri tehtävissä: numeerisessa laskennassa, symbolinkäsittelyssä, grafiikassa ja tilastollisessa analyysissä. Merkintä +++ tarkoittaa erittäin hyvää soveltuvuutta, ++ hyvää ja + välttävää soveltuvuutta. Käännettävien kielien kohdalla käytetty merkintä ●●● tarkoittaa, että kyseiseen tarkoitukseen on saatavissa runsaasti aliohjelmakirjastona toteutettuja rutiinikokoelmia; merkintä ●● tarkoittaa kohtuullista ja ● vähäistä rutiinikokoelmien saatavuutta. Taulukossa on pyritty kuvailemaan tilannetta CSC:n laskentaympäristössä.*

hansa ohjelmistojen tuntemaa tietotyyppiä, mukaanlukien toisia listoja.

Mathematica esittää vektorit, matriisit ja joukot listoina. Maplessa on omat tietorakenteet joukkoja, assosiatiivisia taulukoita ja matriiseita varten.

Maplen monenlaiset tietorakenteet saattavat sekoittaa käyttäjää, mutta verrattuna Mathematicaan Maplessa on paremmat mahdollisuudet tehdä virhetarkistuksia eri tietorakenteita käsiteltäessä. Symbolinkäsittelyjärjestelmien tietoalkiot voivat sisältää numeerista tai symbolista tietoa, mikä mahdollistaa lausekkeiden symbolisen käsittelyn, kuten sieventämisen, derivoinnin tai integroinnin.

Mathematicassa ja Maplessa on monipuoliset grafiikkaominaisuudet. Etenkin funktioiden kuvaajien piirtäminen on helppoa.

Mathematicaa ja Maplea voi laajentaa määrittelemällä funktioita ja sääntöjä tiedostoissa ja lukemalla nämä sisään ohjelmistoon. Mathematicaan voi liittää ulkoisia aliohjelmia MathLink-protokollaa käyttäen.

Puhtaasti numeerista tietoa käsiteltäessä symbolisen laskennan ohjelmistot voivat olla tehottomia, koska algoritmien ja tietorakenteiden pitää pystyä käsittelemään sekä numeerista että symbolista tietoa.

Fortran 90/95 ja FORTRAN 77

Fortran on tärkein tieteellisen ja teknisen laskennan ohjelmointikieli. Uusimmissa standardeissa (Fortran 90 ja Fortran 95) on poistettu useimmat vanhan FORTRAN 77:n ongelmat. Käytössä on paljon ohjelmointia helpottavia piirteitä, kuten aliohjelman paikallisten taulukoiden automaattinen dynaaminen muistinhallinta.

Fortran-kielellä on kirjoitettu valtavasti aliohjelmakirjastoja eri tarkoituksiin. Useimmissa tietokoneympäristöissä voidaan Fortran-kielisiä aliohjelmakirjastoja kutsua muista ohjelmointikielistä.

C ja C++

C-ohjelmointikielessä on taulukkorakenteen lisäksi tietuerakenne sekä osoitinmuuttujat. C-kielen eräs puute on kompleksimuuttujien puuttuminen. Lisäksi osoitinmuuttujien liiallisella käytöllä voi kirjoittaa sekavaa ja vaikeasti optimoitavaa koodia.

C++-kieli on kokonaan oliokeskeinen C-ohjelmointikielen laajennus. Sillä voidaan joustavasti määrittää uusia tietotyyppejä ja kirjoittaa niitä käsittelevät operaattorit siten, että tietotyyppien käyttö on mahdollisimman läpinäkyvää. Uusien tietotyyppien tehokas muistinhallinta voi kuitenkin olla vaikeaa. Aloittelijan kirjoittama C++-koodi tuottaa paljon ajonaikaisia tilapäismuuttujia, joiden käyttö hidastaa suoritusta.

Numeerisen laskennan kannalta C++-standardin olisi suonut sisältävän matriisityypin. C++:lla onkin kirjoitettu monia keskenään yhteensopimattomia matriisiluokkia.

Tehokas ohjelmointityyli

Tulkittavissa kielissä (Matlab, IDL) jokainen lause analysoidaan jokaisella suorituskerralla uudestaan, minkä vuoksi sama asia vie paljon enemmän aikaa kuin käännetyllä ohjelmalla toteutettuna. Parhaimmillaan tulkittavat kielet ovat kaikenlaisissa kokeiluissa ja pienehköissä töissä. Tosin Cedarilla Matlab-ohjelmistoon on asennettu suoritusta nopeuttava Matlab-kääntäjä. Raskaimmat tehtävät olisi parempi laskea joko Fortranilla tai C:llä. Matlab- ja IDL-ohjelmien muuntaminen Fortraniksi on varsin suoraviivaista.

Tulkittavissa kielissä ohjelmointityylin merkitys korostuu. Esimerkiksi Matlabin matriisioperaatiot ovat erittäin tehokkaita, mutta silmukoiksi aukikirjoitettuina operaatiot hidastuvat tuntuvasti.

Fortranissa silmukoiden ja taulukko-operaatioiden välillä ei ole juuri eroa, koska kääntäjä tuottaa molemmista oleellisesti saman koodin. Silti taulukko-operaatioita kannattaa käyttää ohjelmien suunnitteluvaiheessa. Kun algoritmin ajattelee näiden operaatioiden avulla, tuloksena on todennäköisesti siisti ja myös tehokkaasti toimiva ohjelma. Mikäli ohjelma toteutetaan C:llä tai sellaisella Fortranilla, jossa näitä operaatioita ei ole, ne on joka tapauksessa hyvin helppo kirjoittaa auki silmukoiksi.

Käyttöesimerkki: lineaarialgebraa

Seuraavassa käymme läpi lineaarialgebran perusoperaatioiden toteutukset eri ohjelmointikielillä. Olkoot $\mathbf{x}$, $\mathbf{b}$ ja $\mathbf{r}$ vektoreita avaruudessa $\mathbb{R}^n$, matriisi A kooltaan $n \times n$ ja kerroin c reaaliluku. Kullakin kielellä tehdään seuraavat laskutoimitukset:

$$\begin{aligned} \mathbf{r} &= \mathbf{b} - A\mathbf{x}, \\ c &= \mathbf{r}^T\mathbf{r} = \sum_{i=1}^{n} r_i^2, \\ \mathbf{x} &= \mathbf{x} + c\,\mathbf{r}. \end{aligned}$$

Fortran 90/95

Fortran 90/95:n taulukkosyntaksia käyttämällä varsinainen ohjelma mahtuu kolmelle riville. Alla on lisäksi alustettu taulukot satunnaisluvulla. Muissa käännettävissä kielissä satunnaislukugeneraattorien kutsut ovat toteutuskohtaisia, joten en esitä niitä.

```
PROGRAM taulukot
  IMPLICIT NONE
  INTEGER, PARAMETER :: n = 10
  REAL :: c
  REAL, DIMENSION(n) :: x, b, r
```

```
  REAL, DIMENSION(n,n) :: a
  CALL RANDOM_NUMBER(b)
  CALL RANDOM_NUMBER(x)
  CALL RANDOM_NUMBER(a)
  r = b - MATMUL(a,x)
  c = DOT_PRODUCT(r,r)
  x = x + c*r
END PROGRAM taulukot
```

Jos matriisit ja vektorit ovat suuria, kannattaa aliohjelmakutsut `MATMUL` ja `DOT_PRODUCT` usein korvata vastaavilla BLAS-aliohjelmakirjaston kutsuilla, sillä nämä ovat usein optimaalisen tehokkaita (tosin hiukan hankalampia käyttää). Jos taas matriisit ovat pieniä (esim. 3×3), voi paljon CPU-aikaa kuluttavissa kohdissa harkita operaatioiden aukikirjoitusta turhien aliohjelmakutsujen minimoimiseksi.

FORTRAN 77

Voimme toteuttaa edelliset operaatiot FORTRAN 77:llä seuraavasti:

```
      PROGRAM TAULU
      INTEGER N
      PARAMETER (N = 10)
      REAL X(N), B(N), R(N), A(N,N)
      ...
      DO 20 I=1,N
        S=0.0
        DO 10 J=1,N
          S=S+A(I,J)*X(J)
10      CONTINUE
        R(I)=B(I)-S
20    CONTINUE
      C=0
      DO 30 I=1,N
        C=C+R(I)**2
30    CONTINUE
      DO 40 I=1,N
        X(I)=C(I)+C*R(I)
40    CONTINUE
      END
```

Fortranissa taulukot talletetaan muistiin siten, että ensimmäinen indeksi muuttuu nopeimmin. Kaksiulotteisten taulukkojen tapauksessa alkiot ovat siten muistissa sarakkeittain. Talletustavasta johtuen on tehokkaampaa kirjoittaa silmukat siten, että sisimmissä silmukoissa muutetaan taulukon ensimmäistä indeksiä eli edetään sarakkeittain. Tämän vuoksi suuret taulukot kannattaa joskus tehokkuussyistä tallettaa transponoituina.

C-kieli

C-kielinen toteutus lineaarialgebran operaatioista olisi esimerkiksi seuraava:

```
int n = 10;
float x[n], b[n], r[n], a[n][n];
  ...
for (i=0; i<n; i++)
{
   for(s=0.0, j=0; j<n; j++)
       s += a[i][j]*x[j];
   x[i]=b[i]-s;
}
for (c=0.0, i=0; i<n; i++)
    c += r[i]*r[i];
for (i=0; i<n; i++)
    x[i] += c*r[i];
```

C-kielessä taulukoiden indeksit alkavat aina nollasta. Määrittely `float a[10]` luo taulukon, jossa on kymmenen alkiota. Kelvollisia indeksejä ovat siten 0,1,...,9.

Matlab

Matlabin perustietotyyppejä ovat vektorit ja matriisit, joten lineaarialgebran operaatioiden suoritus on suoraviivaista. Seuraavassa käytämme dimensiota $n = 10$. Aluksi luomme satunnaiset vektorit $\mathbf{x}$ ja $\mathbf{b}$ sekä matriisin A.

```
n = 10;
x = rand(n,1);
b = rand(n,1);
A = rand(n,n);
r = b - A*x;
c = r'*r;
x = x + c*r;
```

Matlabin symboli * tarkoittaa matriisikertolaskua. Toisaalta matriisi, jonka koko on 1×1, tulkitaan skalaariksi. Skalaarin ja vektorin kertolasku tehdään komponenteittain.

IDL

IDL:ssä voi käsitellä tehokkaasti matriiseja:

```
n = 10
seed = 123
x = randomu(seed,n,1)
b = randomu(seed,n,1)
a = randomu(seed,n,n)
r = b - a#x
c = transpose(r) # r
x = x + r#c
```

IDL:n matriisitulon symboli on siis #.

Mathematica

Seuraavassa laskemme Mathematicalla lineaarialgebran perusoperaatioita:

```
n = 10;
x = Table[Random[Real], {i,1,n}];
b = Table[Random[Real], {i,1,n}];
A = Table[Random[Real], {i,1,n}, {j,1,n}];
r = b - A . x;
c = r . r;
x = x + c * r;
```

Mathematicassa merkki . (piste) tarkoittaa siis matriisi- ja pistetuloa, vaikka varsinaista tietotyyppiä matriiseille ja vektoreille ei olekaan olemassa.

Maple

Maplessa voimme käyttää `linalg`-pakettia lineaarialgebran operaatioihin:

```
with(linalg):
n := 10:
x := randvector(n):
b := randvector(n):
```

```
A := randmatrix(n,n):
r := evalm(b - A &* x):
c := dotprod(r,r):
x := evalm(x + c*r):
```

Sijoituslauseissa käytämme `evalm`-funktiota, jotta matriisioperaatioita sisältävät lausekkeet suoritettaisiin. Matriisitulon symbolina on Maplessa `&*`.

Toinen käyttöesimerkki: kuvankäsittely

Toisena esimerkkitehtävänä on 100×100 -kokoisen kuvadatan käsittely. Aluksi laskemme kuvadatan keskiarvon. Lopuksi vertailemme kunkin pikselin suuruutta keskiarvon suhteen (kynnystys). Toteutamme laskennan sekä Matlabilla että Mathematicalla käyttäen sekä taulukko- että silmukkaoperaatioita. Tarkoituksena on paitsi verrata ohjelmistoja keskenään myös tutkia eri tyyppisten toteutusten vaikutusta kunkin ohjelmiston osalta.

Matlab

Ensin laskemme operaatiot Matlabin taulukko-operaatioiden avulla ja tämän jälkeen silmukkarakenteilla. Aluksi luomme satunnaisen 500×500 -matriisin:

```
data = floor(256*rand(500));
```

Tämän jälkeen mittaamme operaatioihin kuluvan ajan:

```
time = cputime;
ave = sum(data(:))/prod(size(data));
res = data > ave;
t1 = cputime - time
```

Matlabissa loogisia arvoja vastaavat numeroarvot 0 ja 1. Skalaarin `ave` ja matriisin `data` vertailu tehtiin edellä komponenteittain. Seuraavassa on `for`-silmukkarakennetta käyttävä toteutus:

```
time = cputime;
[ni, nj] = size(data);
s = 0;
for i = 1:ni
  for j = 1:nj
    s = s + data(i,j);
  end
end
ave2 = s/(ni*nj);
for i = 1:ni
  for j = 1:nj
    res2(i,j) = data(i,j) > ave2;
  end
end
t2 = cputime - time
```

Tuloksiksi saamme CSC:n Caper-koneella (Compaq AlphaServer) esimerkiksi seuraavaa:

```
t1 =    0.0333
t2 =    6.0500
```

Täten taulukko-operaatioiden käyttö oli noin 200 kertaa nopeampaa kuin `for`-silmukoiden. Jos kuitenkin tulostaulukko `res2` luodaan ennen sijoitusoperaatioita, kuluu aikaa hiukan vähemmän:

```
time = cputime;
  ...
ave3 = s/(ni*nj);
res3 = zeros(size(data));
for i = 1:ni
  for j = 1:nj
    res3(i,j) = data(i,j) > ave3;
  end
end
t3 = cputime - time
```

Ainoa ero siis on taulukon luominen `zeros`-funktiolla ennen sijoitusoperaatioita. Tämä koodi on noin 50% nopeampi kuin aikaisempi silmukkaversio. Taulukko-operaatioiden tehoon ei tietenkään päästä.

Mathematica

Voimme laskea kuvadatan keskiarvon Mathematicalla seuraavasti:

```
average[l_List] := Apply[Plus,Flatten[l]] /
  Apply[Times,Dimensions[l]];

average2[l_List] :=
  Block[{ni, nj, sum = 0, m},
    {ni,nj} = Dimensions[l];
    For[i = 1, i <= ni, i++,
      For[j = 1, j <= nj, j++,
        sum = sum + l[[i,j]]]];
    m = sum/(ni*nj)]
```

Tässä määrittelimme kaksi rutiinia, joista ensimmäinen on toteutettu Mathematican listojenkäsittelyrutiinilla `Apply` ja toinen `For`-silmukkarakenteella. Seuraavassa on esimerkki rutiinien testauksesta:

```
data = Table[Random[Integer, {0,255}],
               {i,1,500}, {j,1,500}];
t1 = First[Timing[ ave = average[data] ]];
t2 = First[Timing[ ave2 = average2[data] ]];
t3 = First[Timing[ res = Outer[(# > ave)&, data] ]];
t4 = First[Timing[ res2 = Array[(data[[##]]>ave)&,
                        Dimensions[data]] ]];
{t1, t2, t3, t4}
```

Mathematica-lausekkeet tuottavat listarakenteet `res` ja `res2`, jotka sisältävät loogisia arvoja `True` ja `False`. Käytämme ajanmittaukseen `Timing`-funktiota. Tuloksiksi saamme Cedar-koneessa (SGI Origin 2000) esimerkiksi seuraavaa:

```
{0.67 Second, 12.7 Second,
    6.67 Second, 12.29 Second}
```

Siten silmukkarakennetta hyödyntävä laskurutiini on noin 20 kertaa hitaampi kuin listaoperaatioita käyttävä. Tämä on tyypillistä tulkittaville kielille. Lisäksi ulkotulo-operaatiota `Outer` käyttävä lauseke on `Array`-funktiota käyttävää nopeampi.

Prototyyppityöskentely

Tutkimuksessa on usein käyttökelpoista ratkaista matemaattisia ongelmia interaktiivisten ohjelmisto-

jen avulla. Tämä toimii parhaiten pienille tehtäville. Kun olemme kirjoittaneet prototyypin ratkaisurutiinista, voimme jatkossa toteuttaa saman algoritmin perinteisillä ohjelmointikielillä. Siirtoa helpottaa, jos interaktiivisessa ympäristössä ja ohjelmointikielessä on käytössä samankaltaisia tietorakenteita (vertaa esim. Matlab ja Fortran 90/95).

Prototyypin laatimiseen kannattaa käyttää interaktiivista matriisikieltä kuten Matlabia tai symbolisen laskennan ohjelmistoa kuten Mathematicaa tai Maplea. Interaktiivisessa ympäristössä voi nopeasti toteuttaa ja testata prototyyppiohjelmia. Tässä ympäristössä voi tutkia erilaisten parametrien vaikutusta ohjelman suoritukseen sekä tarkastella laskentatuloksia graafisesti. Myös eri laskenta-algoritmien vertailu on helppoa.

Prototyypin testaus pitää yleensä tehdä pienidimensioisilla testitapauksilla, sillä varsinainen laskenta kestää usein liian kauan näissä ympäristöissä. Interaktiivisen ympäristön laskentaa voi nopeuttaa joissakin tapauksissa optimoimalla koodia, kääntämällä koodi konekieliseksi tai käyttämällä ulkopuolisia kirjastoja.

Raskaissa laskentatehtävissä on ohjelman lopullinen versio kirjoitettava jollakin käännettävällä ohjelmointikielellä kenties käyttäen hyväksi aliohjelmakirjastoja. Prototyypin ohjelmakoodia voi ehkä sellaisenaan muuntaa ohjelmointikielten koodiksi. Eräät operaatiot, kuten yhtälöryhmien ratkaisu, voi korvata kutsulla aliohjelmakirjastoihin. Testauksessa on helppo verrata prototyypin ja varsinaisen laskentaympäristön antamia tuloksia.

Eri ympäristöjen yhteiskäyttö

Usein on mahdollista rakentaa tehokas ja helppokäyttöinen laskentaympäristö yhdistämällä esimerkiksi jokin matriisikieli aliohjelmakirjastoihin.

Interaktiiviset laskentaympäristöt tarjoavat helppokäyttöisen käyttöliittymän raskaaseen numeeriseen laskentaan. Tulosten graafinen tarkastelu ja eri parametrien muuntelu on helppoa, kunhan näkee sen vaivan, että liittää oman laskentarutiininsa tällaiseen ympäristöön.

Symbolisen laskennan ohjelmien käyttö helpottaa monia laskentatehtäviä. Esimerkiksi optimointialgoritmeja varten voi funktion gradientin ja Hessen matriisin muodostaa automaattisesti symbolisen laskennan ohjelmistoilla. Lisäksi lausekkeet voi tulostaa esimerkiksi Fortran- tai C-kielisinä versioina, jotka voi sijoittaa osaksi ohjelmakoodia.

Matlabiin voi liittää Fortran- ja C-kielisiä aliohjelmia käyttäen ns. MEX-tiedostoja, joilla määritellään kutsuliittymä. Esimerkiksi Matlabin matriisit siirtyvät tehokkaasti ohjelmointikieliin. Matlabiin on saatavilla myös liitäntä symbolinkäsittelyyn. *Symbolic Math* ja *Extended Symbolic Math Toolbox* liittävät Matlabin Maple-ohjelmiston laskentaytimiin.

Lisätietoja

CSC:n julkaisemassa oppaassa [Haa98] on esitelty matemaattisia ohjelmistoja sekä mm. aliohjelmakirjastojen käyttöä. Luonnollisesti eri ohjelmistoilla ja ohjelmointikielillä on omat käsikirjansa, joita on lueteltu seuraavassa kirjallisuusluettelossa.

Kirjallisuutta

[Haa98] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[HHR96] K.M. Heal, M.L. Hansen ja K. M. Rickard. *Maple V, Learning Guide*. Hamilton Printing Company, 1996.

[HRR98] Juha Haataja, Jussi Rahola ja Juha Ruokolainen, toim. *Fortran 90/95*. CSC, 1998.

[Mat92] The MathWorks, Inc. *MATLAB User's Guide*, 1992.

[MGH+96] M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn ja S. M. Vorkoetter. *Programming Guide*. Hamilton Printing Company, 1996.

[Res] Research Systems, Inc. *IDL User's Guide*.

[Wol96] Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Cambridge University Press, 1996.

Matlab 5.x — kysymyksiä ja vastauksia

Juha Haataja
Juha.Haataja@csc.fi

Matlab ("Matrix Laboratory") on käytetyimpiä sovellusohjelmia CSC:n koneilla. Monipuolisuutensa ansiosta se soveltuu yhtä hyvin numeeriseen laskentaan kuin visualisointiin.

Matlabia koskevia kysymyksiä voi lähettää CSC:hen Esa Lammille (e-mail `Esa.Lammi@csc.fi`), Ville Savolaiselle (`Ville.Savolainen@csc.fi`) ja Juha Haatajalle (`Juha.Haataja@csc.fi`).

Seuraavissa esimerkeissä käytetään Matlabin versiota 5.1. Taulukossa 2 on esitetty joitakin keskeisiä Matlab-komentoja.

Matlabin avustustekstit

K: *Miten saan Matlabin `help`-komennon tekstit näkymään ruutu kerrallaan?*

V: Anna Matlabin komento `more on`, jolloin avustustekstit sivutetaan. Lisätietoja saat Matlabin komennolla `help more`.

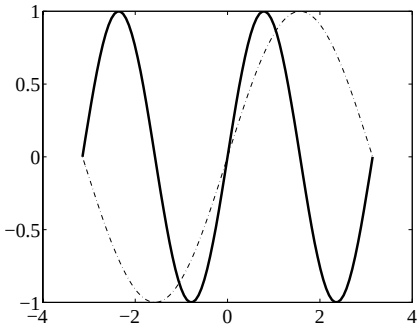
Viivan paksuus

K: *Miten saan Matlabin tulostamaan kuvaajaan piirretyistä käyristä yhden paksummalla viivalla kuin muut?*

V: Matlabin piirtämien kuvaajien ominaisuuksia voi tutkia ja muuttaa komennolla `set`. Seuraavassa on yksinkertainen esimerkki:

```
x = -pi:(pi/50):pi;
plt1 = plot(x,sin(x),'-.'); hold on
plt2 = plot(x,sin(2*x),'-');
set(plt2,'LineWidth',3); hold off
print -deps plt
```

Lisätietoja saa komennoilla `help set` ja `help get`. Tiedostoon `plt.eps` saadaan seuraava PostScript-kuva:



Matlab-koodi	*Selitys*
`help` *fun*	Komennon tai funktion *fun* avustusteksti.
`quit`	Lopetetaan Matlabin käyttö.
`type` *fun*	Tulostetaan rutiinin *fun* ohjelmakoodi.
`more on`	Näytetään avustustekstit sivu kerrallaan.
`lookfor` *sana*	Haetaan avainsanaan *sana* liittyviä komentoja.
`who`, `whos`	Lista määritellyistä muuttujista.
`x = 1:n`	Vektori $(1,2,\ldots,n)$.
`x = (1:n).^2`	Vektori $(1,4,\ldots,n^2)$.
`x = i:j:k`	Vektori $(i,i+j,i+2j,\ldots,k)$.
`a = [a11 a12; a21 a22]`	Matriisi $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$.
`a = [1 2 3; 4 5 6]`	Muodostetaan matriisi a.
`x = a(:)`	Matriisin a muuttaminen vektoriksi x.
`a = [a; [7 8 9]]`	Rivin lisääminen matriisiin a.
`a = [a [10 11 12]']`	Sarakkeen lisääminen matriisiin a.
`x = a(:,k)`	Haetaan matriisin a kaikki rivit sarakkeelta k.
`x = a(i:j,k)`	Haetaan matriisin a rivit $i,i+1,\ldots,j$ sarakkeelta k.
`x = a([1 2 3],[2 3])`	Haetaan matriisin a annetut rivit annetuilta sarakkeilta.
`a(:,[1 3]) = []`	Poistetaan matriisista a sarakkeet 1 ja 3.

Taulukko 2: *Yllä esitetyt Matlab-lauseet ja komennot ovat usein hyödyllisiä.*

Viipaloitua dataa

K: *Miten saan visualisoitua 3D-volyymidatan "viipaloituna"?*

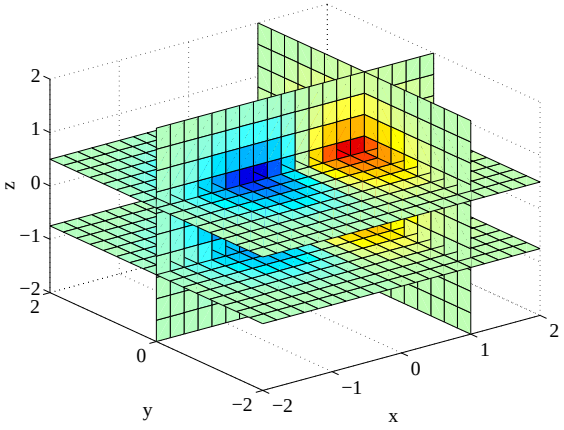
V: Seuraavassa lasketaan funktion $f(x,y,z) = x\exp(-x^2-y^2-z^2)$ arvot kolmiulotteisen suorakulmaisen hilan pisteissä $x,y,z \in [-2,2]$. Aluksi määritellään kolmiulotteinen taulukko $v(i,j,k)$, $i = 1,\ldots,m$, $j = 1,\ldots,n$, $k = 1,\ldots,p$:

```
r1 = -2:.2:2; r2 = -2:.25:2;
r3 = -2:.4:2;
[x,y,z] = meshgrid(r1, r2, r3);
v = x.*exp(-x.^2 - y.^2 - z.^2);
```

Matlabin rutiini `slice` ymmärtää tässä luodun kolmiulotteisen taulukon `v`. Alla olevan kaltaisen kuvan voi piirtää esim. seuraavasti:

```
colormap(jet); brighten(0.5);
slice(r1,r2,r3,v, ...
      [1.0],[0.0],[-0.75,0.5])
xlabel('x'); ylabel('y'); zlabel('z');
```

Lisätietoja saa Matlabin komennoilla `help slice` ja `help meshgrid`.



Kirjasimen tyyppi

K: *Miten saan muutettua Matlabin grafiikan kirjasimen tyypin ja koon toiseksi?*

V: Seuraavassa visualisoidaan edellä muodostetun kolmiulotteisen taulukon `v` sisältämää dataa. Oletuskirjasimeksi valitaan Times ja kooksi 18 pistettä.

```
set(0, 'DefaultAxesFontName', 'times')
set(0, 'DefaultTextFontName', 'times')
set(0, 'DefaultAxesFontSize', 18)
set(0, 'DefaultTextFontSize', 18)
colormap(jet); brighten(0.5);
slice(r1,r2,r3,v, ...
      [1.0],[0.0],[-0.75,0.5])
xlabel('x'); ylabel('y'); zlabel('z');
```

Komennolla `set` asetettiin oletusfontin tyyppi ja koko. Kolmiulotteisen taulukon `v` sisältämää dataa havainnollistettiin poikkileikkauskuvia piirtävällä funktiolla `slice`. Kuvan voi tulostaa PostScript-muodossa tiedostoon `dim.eps` komennolla

```
print -deps dim
```

Grafiikan oletusasetuksia saa tulostettua komennoilla `get(0)` ja `get(0,'nimi')`, esimerkiksi seuraavasti:

```
get(0,'DefaultAxesFontSize')
```

3D-kuvaaja epäsäännölliselle datalle

K: *Miten saan piirrettyä 3D-pinnan, kun käytettävissäni on arvot solmupisteissä sekä solmupisteiden koordinaatit?*

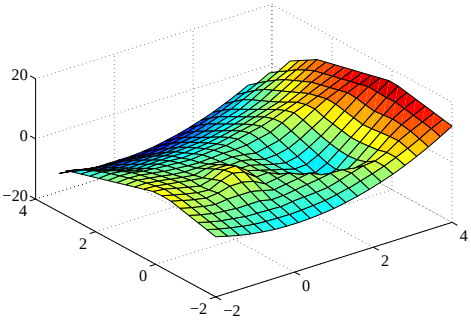
V: Olkoon data ASCII-tiedostossa `hila.dat`:

```
-2  -2   0
-2  -0.5  3.75
-2   0   4
 ...
 4   2.5   9.75
 4   3.5   3.75
```

Puuttuvat arvot voi interpoloida komennolla `griddata`. Samalla komennolla voi myös interpoloida epäsäännöllisessä hilassa annetun datan säännölliseen hilaan. Seuraavassa on tästä esimerkki:

```
load hila.dat;
x = hila(:,1); y = hila(:,2);
z = hila(:,3);
minx = min(x); maxx = max(x);
miny = min(y); maxy = max(y);
resx = 20; resy = 20;
xi = [minx:((maxx-minx)/resx):maxx]';
yi = [miny:((maxy-miny)/resy):maxy];
zi = griddata(x,y,z,xi,yi);
surf(xi,yi,zi) % piirretään kuva
set(gca,'CameraPosition',[-15 -20 180])
```

Tuloksena voisi olla seuraavan näköinen kuvaaja:



Pisteiden poistaminen kuvaajasta

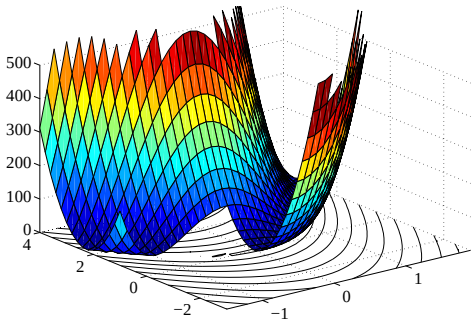
K: *Miten saan piirrettyä pintakaavion siten, että osa pinnasta jää piirtämättä?*

V: Käytä Matlabin käsikirjassa mainittua "`NaN`-

temppua": jos kuvapisteen arvona on NaN ("not a number"), ei kyseistä pistettä piirretä. Seuraavassa on esimerkki Rosenbrockin funktion $100(x_2 - x_1^2)^2 + (1 - x_1)^2$ pintakaavion ja tasa-arvokäyrien piirtämisestä:

```
x1 = -1.5:0.1:1.9; x2 = -3:0.2:4;
[xx1 xx2] = meshgrid(x1, x2);
z = 100*(xx2-xx1.^2).^2 + (1-xx1).^2;
nan = NaN;
z0 = z;
% pisteiden poisto pintakaaviosta:
z0(z0 > 600) = nan*z0(z0 > 600);
z0(1:20,1:15) = nan(ones(20,15));
% piirretään pintakaavio:
surf(x1,x2,z0)
axis([-1.5 1.9 -3 4 0 500])
hold on
set(gcf, 'DefaultLineLineWidth', 2)
contour(x1,x2,z,(0:1.4:50).^3);
hold off
caxis([0,500])
```

Tässä siis jätettiin pintakaaviossa piirtämättä ne pisteet, joissa funktion arvot olivat suurempia kuin 600. Lisäksi pintakaavion etunurkasta jätettiin osa piirtämättä. Tuloksena on seuraava kuva:



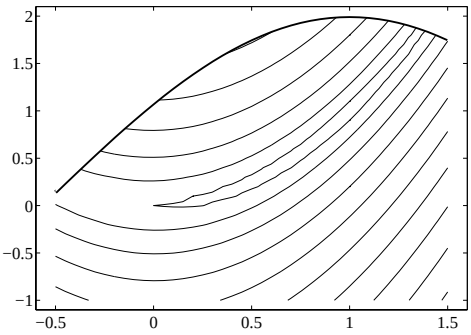
Kuvan osan piilottaminen

K: *Miten voi piilottaa Matlab-kuvaajasta esimerkiksi annetun käyrän $y = f(x)$ yläpuolisen alueen?*

V: Kuvan osan peittämisen voi tehdä esimerkiksi käyttämällä Matlabin `fill`-komentoa. Seuraavassa piirretään Rosenbrockin funktion tasa-arvokäyriä. Kuvaajasta poistetaan käyrän $y(x) = 2\cos x - 1$ yläpuolinen alue:

```
% piirretään tasa-arvokäyrät
x0 = -0.5:0.05:1.5; y0 = -1:0.1:2;
[xx yy] = meshgrid(x0, y0);
z = 100*(yy - xx.^2).^2 + (1 - xx).^2;
contour(x0,y0,z,(0:1:50).^3);
axis([-0.6 1.6 -1.1 2.1]); hold on
% näytetyn alueen yläreuna
x = x0; y = 2*cos(x-1); plt = plot(x,y);
% paksumpi viiva:
set(plt, 'LineWidth', 4)
% piilotetaan osa kuvaajasta
xl = get(gca,'xlim');
yl = get(gca, 'ylim');
% etäisyys kuvan reunasta:
bd = .02*(xl(2)-xl(1));
xpts = [(xl(1)+[bd bd]) x ...
        (xl(2)-[bd bd])];
ypts = [(yl(2)-[bd bd]) y ...
        y(length(y)) (yl(2)-bd)];
% piirretään tausta:
fh = fill(xpts, ypts, 'w');
% poistetaan reuna:
set(fh, 'EdgeColor', 'none')
hold off
```

Edellä siis käytettiin komentoa `fill` peittämään käyrän $y(x)$ yläpuolinen alue. Seuraavassa on edellisten komentojen tuottama kuva:



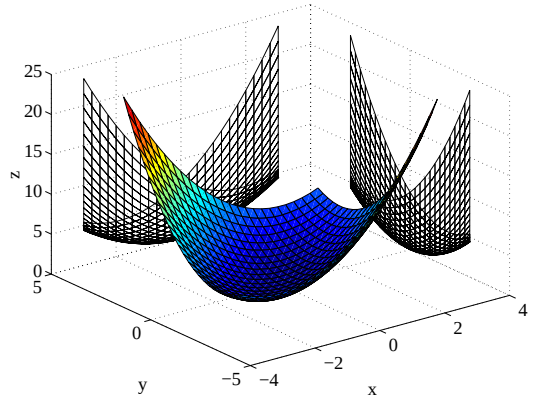
Projektiot 3D-kuvaajaan

K: *Haluaisin lisätä 3D-pintakaavioon datan projektiot yz- ja xz-tasoille. Miten tämän voi tehdä Matlabissa?*

V: Seuraavassa on esimerkki kvadraattisen funktion $f(x) = x^T Qx, x \in \mathbb{R}^n$, kuvaajan piirtämisestä:

```
Q = [0.8, -0.5; -0.5, 0.8];
X = -3:0.25:3; Y = X;
n = length(X);
[XX,YY] = meshgrid(X,Y);
Z = Q*[XX(:)'; YY(:)'];
Z = reshape(Z(1,:).*XX(:)'+ ...
    Z(2,:).*YY(:)', n, n);
surf(X, Y, Z);
xlim = get(gca,'xlim');
ylim = get(gca,'ylim');
hold on
h = mesh(X,ylim(2)*ones(size(Y)),Z);
set(h,'facecolor','none')
h = mesh(xlim(2)*ones(size(Y)),Y,Z);
set(h,'facecolor','none')
hold off
xlabel('x'); ylabel('y'); zlabel('z')
```

Tuloksena on seuraavan näköinen kuvaaja:



Lisätietoja saa komennoilla `help mesh`, `help get` ja `help set`.

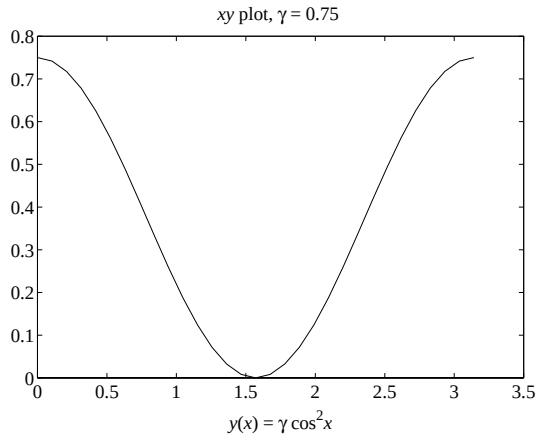
Ylä- ja alaindeksit sekä kreikkalaiset kirjaimet

K: *Miten saan Matlab-kuvien otsikkoon ylä- ja alaindeksit ja kreikkalaiset kirjaimet?*

V: Matlabin versio 5 mahdollistaa mm. ylä- ja alaindeksien ja kreikkalaisten kirjainten käytön:

```
x = 0:(pi/30):pi;
plt = plot(x,0.75*cos(x).^2);
lh = xlabel(['{\ity}({\itx}) = ' ...
   '{\gamma} cos^2{\itx}']);
th = title(['{\itxy} plot, ' ...
            '{\gamma} = 0.75 ']);
set(gca,'Position',...
  [0.10 0.15 0.8 0.75])
set(lh,'Position',[1.75 -0.08 0])
```

Halutut tekstityypit valitaan siis TEXiä muistuttavalla syntaksilla. Tuloksena on seuraavan näköinen kuva:



Lisätietoja

Matlabin käyttöesimerkkejä löytyy kirjallisuusluettelossa mainituista teoksista. Matlabia on esitelty mm. CSC:n oppaassa *Matemaattiset ohjelmistot*. Tietolähteenä voi käyttää myös Usenet News -uutispalstaa `comp.soft-sys.matlab`. Myös CSC:n asiantuntijoilta voi kysellä Matlabiin liittyviä vinkkejä.

Kirjallisuutta

- Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC – Tieteellinen laskenta Oy, 1998. 3. painos. 138 s. ISBN 952-9821-46-8. Web-osoite on `http://www.csc.fi/oppaat/mat.ohj/`.
- Simo K. Kivelä. *MATLAB-opas*. Otakustantamo, 1991.
- The MathWorks, Inc. *Getting Started with Matlab, Version 5*, 1996.
- The MathWorks, Inc. *Using Matlab, Version 5*, 1996.
- The MathWorks, Inc. *Matlab 5 New Features*, 1996.
- Ville Savolainen. *Tehokas ohjelmointi Matlabilla*. @CSC 1/1998.

Matlab 5.x — lisää kysymyksiä ja vastauksia

Juha Haataja ja Ville Savolainen
Juha.Haataja@csc.fi, Ville.Savolainen@csc.fi

Kerromme tässä artikkelissa Matlab 5.x:n käytöstä — tätä kirjoittaessamme käytössämme on versio 5.1. Myös edellisissä @CSC-lehdissä oli Matlabin käyttövinkkejä [Haa98b, Sav98].

Matriisin muodostaminen vektorista

K: *Haluaisin muodostaa n -alkioisesta sarakevektorista `x` matriisin `m`, jonka jokainen sarake (`p` kpl) on sama kuin `x`.*

V: Seuraava tapa on tehokas varsinkin, jos *n* on iso:

```
n = 500; x = rand(n,1);
m = x(:,ones(1,p));
```

Lisää sarakkeita tai rivejä

K: *Kuinka saa lisättyä olemassa olevaan matriisiin uuden rivin tai sarakkeen, jonka arvot ovat vakioita?*

V: Myös skalaariarvon lisäyksen voi tehdä käyttämällä Matlabin taulukko-operaatioita:

```
>> m = [1 2 3; 4 5 6]
m =
     1     2     3
     4     5     6
>> m = [m; 100 + zeros(size(m(1,:)))]
m =
     1     2     3
     4     5     6
   100   100   100
>> m = [m -100 + zeros(size(m(:,1)))]
m =
     1     2     3  -100
     4     5     6  -100
   100   100   100  -100
```

Lisättävissä riveissä ja sarakkeissa täytyy olla oikea määrä alkioita. Lisäksi on syytä muistaa, että matriisin riviä vastaa vaakavektori ja saraketta pystyvektori. Seuraavassa on toinen tapa tehdä samat operaatiot:

```
>> m = [1 2 3; 4 5 6];
>> a = 100;
>> m = [m; a(1,ones(size(m(1,:))))]
m =
     1     2     3
     4     5     6
   100   100   100
>> m = [m -a(ones(size(m(:,1))),1)]
m =
     1     2     3  -100
     4     5     6  -100
   100   100   100  -100
```

Kolmas tapa hoitaa lisääminen on käyttää matriisin indeksointia. Tämä tapa toimii myös useampiulotteisissa tapauksissa:

```
>> m(:,5) = 123*ones(size(m(:,1)))
m =
     1     2     3  -100   123
     4     5     6  -100   123
   100   100   100  -100   123
```

Matlab 5:ssä voi käyttää sijoituslauseessa myös skalaariarvoa Fortran 90:n tapaan:

```
>> m(:,6) = -321
m =
     1     2     3  -100   123  -321
     4     5     6  -100   123  -321
   100   100   100  -100   123  -321
```

Taulukoiden alkioiden muuttaminen

K: *Miten saisin muutettua kerralla vektorin kaikki ne alkiot, joiden arvo on tietty luku?*

V: Seuraavassa on tästä esimerkki:

```
>> x = [3 4 5 4 5 1];
>> ind = find(x == 5)
ind =
     5     7
```

Vektorissa `ind` on nyt vektorin `x` arvoa 5 vastaavien alkioiden indeksit. Korvaaminen onnistuu esimerkiksi seuraavasti:

```
>> x(ind) = 999
x =
     3     4   999     4   999     1
```

Koska indeksivektorin pituus on kaksi, voidaan korvauksessa käyttää kaksialkioista vektoria:

```
>> x(ind) = [-1 -2]
x =
     3     4    -1     4    -2     1
```

Alkioiden valinta

K: *Haluaisin löytää vektorista x ensimmäisen alkion, jonka arvo on suurempi kuin edeltävän alkion.*

V: Funktio `diff` laskee peräkkäisten alkioiden erotukset:

```
>> x = [7 6 5 6 7 4 5 3];
>> diff(x)
ans =
   -1   -1    1    1   -3    1   -2
```

Funktio `find` palauttaa loogista ehtoa vastaavat vektorin alkion indeksit. Ratkaisun voi koodata vaikkapa seuraavasti:

```
>> ind = min(find(diff(x) > 0)) + 1
ind =
     4
```

Tässä esimerkissä neljäs alkio oli edeltäjäänsä suurempi.

K: *Haluaisin löytää arvoja nolla ja yksi sisältävästä vektorista maksimimäärän peräkkäisiä ykkösiä.*

V: Seuraavassa eräs ratkaisu:

```
>> x = [1 1 1 1 0 1 1 0 1 1 1];
>> ind = find([0 x 0] ~= 1);
>> maxnr = max(diff(ind)) - 1
maxnr =
     4
```

Tässä haettiin vertailuoperaattorin `~=` avulla paikat, joissa vektori arvo ei ollut yksi. Lisäksi täytyy käsitellä vektorin alku ja loppu, jonka vuoksi lisättiin nolla vektorin kumpaankin päähän. Funktion `diff` avulla lasketiin nollien välimatkat, jonka jälkeen vastaukseksi otetaan maksimiarvo välimatkoista.

Seuraavassa toinen esimerkki ja yhteen lauseeseen lyhennetty ratkaisu:

```
>> x = [1 0 1 0 0 0 1 1 0 1];
>> max(diff(find([0 x 0] ~= 1))) - 1
ans =
     2
```

Silmukoiden vektorointi

K: *Haluaisin nopeuttaa ehtolauseen sisältävän silmukan toimintaa. Seuraavassa x ja x2 ovat vektoreita.*

```
for ind=1:length(x),
  if (x(ind)>pi)
    x2(ind)=x(ind);
  else
    x2(ind)=(2*pi)-x(ind);
  end
end
```

V: Silmukoiden kirjoittamisen voi usein välttää komentoa `find` käyttämällä:

```
ind = 1:length(x);
x2(ind) = x(ind);
idx = find(x(ind) <= pi);
x2(idx) = 2*pi - x2(idx);
```

Toinen tapa vektoroida silmukka olisi seuraavan kaltainen:

```
testi = x<=pi;
x2 = testi.*(2*pi-x) + (~testi).*x;
```

tai vielä lyhyemmin

```
x2 = x + (x<=pi).*(2*pi-2*x);
```

Tridiagonaalimatriisi

K: *Haluaisin luoda Matlabin avulla matriisin*

$$m = \begin{pmatrix} 4 & -1 & 0 & & 0 \\ -1 & 4 & -1 & \cdots & 0 \\ 0 & -1 & 4 & & 0 \\ \vdots & & & \ddots & -1 \\ 0 & 0 & 0 & -1 & 4 \end{pmatrix}.$$

V: Käytä funktiota `diag` matriisin muodostamiseen:

```
>> n = 4;
>> m = 4*diag(ones(n,1)) ...
   - diag(ones(n-1,1),1) ...
   - diag(ones(n-1,1),-1)
m =
     4    -1     0     0
    -1     4    -1     0
     0    -1     4    -1
     0     0    -1     4
```

Harvat diagonaalimatriisit

K: *Lasken $N \times N$-matriisin `J`, jolla on nollasta eroavia alkioita vain päädiagonaalilla ja määrätyillä $M-1$ sivudiagonaaleilla. Haluaisin laskea vain nämä alkiot ja tallettaa `J`:n harvana matriisina.*

V: Laske pää- ja sivudiagonaalit $N \times M$-apumatriisin `B` pystyriveihin siten, että `B`:n rivi-indeksi vastaa matriisin `J` sarakeindeksiä. Muunna `B` komennolla `spdiags` harvamatriisiksi `J`. Olkoon esimerkkinä edellinen yksinkertainen tridiagonaalimatriisi:

```
>> N = 4;
>> B = zeros(N,3);
>> B(1:N-1,1) = -ones(N-1,1);
>> B(1:N,2) = 4*ones(N,1);
>> B(2:N,3) = -ones(N-1,1);
>> J = spdiags(B,-1:1,N,N);
```

Funktion `spdiags` toisena argumenttina oleva vektori siis kertoo diagonaalien etäisyydet päädiagonaa-

lista (positiiviset arvot yläpuolella) ja kaksi viimeistä argumenttia `J`:n koon.

Harvat matriisit

K: *Minulla on ASCII-tiedostossa dataa kolmisarakkeisessa muodossa*

$$\begin{array}{ll} i_{11} & i_{12} \quad arvo_1 \\ i_{21} & i_{22} \quad arvo_2 \\ \ldots & \end{array}$$

ja haluaisin muodostaa datasta harvan matriisin m, jonka nollasta poikkeavien alkioiden arvot ovat $m(i_{11}, i_{12}) = arvo_1$ *jne.*

V: Lue data sisään `load`-komennolla ja muunna syntyvä kolmisarakkeinen matriisi harvaksi matriisiksi. Olkoon data tiedostossa `test.dat`:

```
>> type test.dat
1 1  4
2 1 -1
  ...
8 8  4
>> load test.dat
>> m = spconvert(test)
m =
   (1,1)        4
   (2,1)       -1
   (1,2)       -1
     ...
   (7,8)       -1
   (8,8)        4
```

Funktio `spconvert` siis muunsi kolmisarakkeisen taulukon `test` harvaksi matriisiksi `m`.

Käyrän sovitus

K: *Miten saan sovitettua mittausdataan toisen asteen polynomin? Entä lineaarikombinaation annetusta funktiokannasta?*

V: Polynomien sovitukseen on oma komento `polyfit` sekä komento `polyval` sovituksen arvojen laskemiseen. Yleisemmässä tapauksessa pienimmän neliösumman sovitus käy ratkaisemalla kertoimet \-operaattorin avulla. Seuraavassa on esimerkki tällä suoritetusta kolmannen asteen polynomin

$$y = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

sovituksesta dataan (t, y):

```
>> t=[1.0 2.7 3.2 4.8 5.6]';
>> y=[14.2 17.8 22.0 38.3 51.7]';
>> X=[ones(size(t)) t t.^2 t.^3];
>> a=X\y
a =
   15.7580
   -2.8467
    1.1644
    0.0870
```

Siis haluamamme polynomi on

$$y = 15.7580 - 2.8467t + 1.1644t^2 + 0.0870t^3.$$

Haluttujen kuvien tulostaminen

K: *Piirrän Matlab-komentotiedoston avulla joukon kuvia — miten saan tulostettua haluamani kuvat tiedostoon, kun etukäteen en tiedä, mitkä kuvat ovat kiinnostavia ja mitkä eivät.*

V: Matlabin komennoilla `pause` ja `keyboard` voi pysäyttää komentotiedoston toiminnan. Toiminta jatkuu `pause`-komennon jälkeen painettaessa jotakin näppäintä Matlabin komentoikkunassa. Komennon `keyboard` jälkeen Matlab suorittaa komentorivillä annettuja komentoja, kunnes annetaan komento `return`, jolloin palataan suorittamaan komentotiedostoa. Lisätietoja saa komennoilla `help pause` ja `help keyboard`.

Grafiikkaikkunassa oleva kuva tulostetaan tiedostoon komennolla `print`. Tulostamisen voi liittää esimerkiksi näppäimen painamiseen hiiren osoittimen ollessa grafiikkaikkunan sisällä. Tämä onnistuu antamalla komentotiedoston alussa (ennen grafiikkaikkunan avaamista) käsky

```
set(0, 'DefaultFigureKeypressFcn', ...
['print -deps kuva.ps; ' ...
 'disp(''Printing...'')']);
```

Tällöin Matlab tulostaa kuvan tiedostoon `kuva.ps`. Komentotiedoston ajon jälkeen on syytä antaa komento

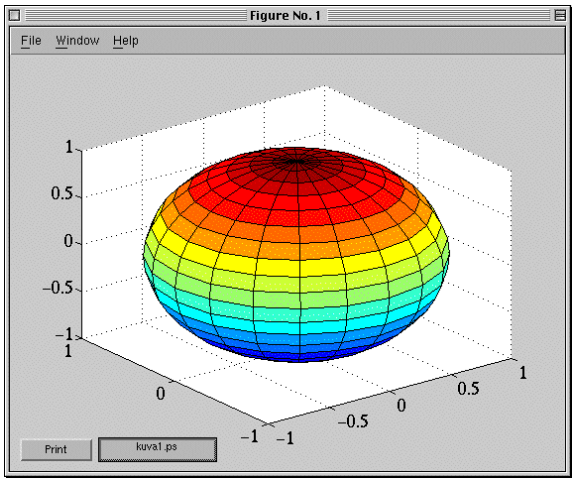
```
set(0, 'DefaultFigureKeypressFcn', '');
```

jolloin näppäimen painaminen ei tulosta uutta kuvaa samaan tiedostoon. Jos haluaa tulostaa useampia kuvia, voi Matlabista siirtyä komentotulkkiin näppäinyhdistelmällä `^Z` ja nimetä tiedoston `kuva.ps` uudelleen.

Toinen mahdollisuus kuvan piirtämiseen on luoda Matlabin avulla painonappi, jota painettaessa nykyisessä grafiikkaikkunassa oleva kuva tulostetaan. Tulostustiedoston nimen voi kirjoittaa esimerkiksi erilliseen tekstikenttään. Seuraavassa esimerkki tarvittavista komennoista:

```
prtcmd = uicontrol(gcf, 'Style', ...
'push', 'Position', [10 10 75 25], ...
'String', 'Print', 'CallBack', ...
['print(''-deps'', ' ...
 'get(prtname, ''String''))']);
prtname = uicontrol(gcf, 'Style', ...
'edit', 'String', 'kuva.ps', ...
'Position', [90 10 120 25]);
```

Seuraavassa on esimerkki painonapin ja tekstikentän toiminnasta:



Painonappia Print painettaessa grafiikkaikkunan kuva siis tulostuu annetun nimiseen tiedostoon (painonappi ja tekstikenttä eivät näy tulostetussa kuvassa).

Lisätietoja saa komennolla `help` ja Matlabin käsikirjasta *Building a Graphical User Interface*.

Kuvan talletus m-tiedostoon

K: *Onko mahdollista tallettaa kuvan tekemiseen käytetyt komennot m-tiedostoon myöhemmin suoritettaviksi?*

V: Matlabissa on mahdollista antaa komennolle `print` valitsin `-dmfile`, jolloin Matlab pyrkii tallettamaan kuvan tekemiseen käytetyt käskyt annetun nimiseen m-tiedostoon:

```
x = -pi:(pi/20):pi;
p1 = plot(x,sin(x),'.'); hold on
set(p1,'MarkerSize',20)
p2 = plot(x,cos(x),'-'); hold off
print -dmfile kuva
```

Komentojen ansiosta Matlab luo tiedostot `kuva.m` ja `kuva.mat`, jotka sisältävät kuvan piirtämiseen tarvittavan komennot ja datan. Kuvan saa piirrettyä uudestaan antamalla Matlabissa komento `kuva`.

Lisätietoja saa komennolla `help print`.

Matlab ja Adobe Illustrator

K: *Miten saan siirrettyä Matlabin tuottaman kuvan Macintoshin Adobe Illustrator -ohjelmaan?*

V: Matlabin komennolla `print` voi tulostaa kuvan EPS-muodossa tiedostoon:

```
print -deps kuva
```

Tiedoston `kuva.eps` voi siirtää ASCII-muodossa Macintoshiin, jossa sitä voi editoida Illustrator-ohjelmalla. Lisätietoja Matlabin tulostusmahdollisuuksista saa komennolla `help print`.

Lisätietoja

Matlabin käyttöesimerkkejä löytyy kirjallisuusluettelossa mainituista teoksista, esimerkiksi CSC:n oppaasta *Matemaattiset ohjelmistot* [Haa98a]. Tietolähteenä voi käyttää myös Usenet News -uutispalstaa `comp.soft-sys.matlab`. Myös CSC:n asiantuntijoilta voi kysellä Matlabiin liittyviä vinkkejä.

Kirjallisuutta

[Haa98a] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[Haa98b] Juha Haataja, Matlab 5.x — kysymyksiä ja vastauksia, *@CSC*, 2/1998.

[Mat96a] The MathWorks, Inc. *Matlab 5 New Features*, 1996.

[Mat96b] The MathWorks, Inc. *Using Matlab, Version 5*, 1996.

[Mat97] The MathWorks, Inc. *Getting Started with Matlab, Version 5.1*, 1997.

[Sav98] Ville Savolainen, Tehokas ohjelmointi Matlabilla, *@CSC*, 1/1998.

Matlab 5.x — uusia kysymyksiä ja vastauksia

Ville Savolainen
Ville.Savolainen@csc.fi

Jatkan tässä artikkelissa vinkkipalstaa Matlab 5.x:n käyttöstä. Aikaisemmat artikkelisarjan osat on julkaistu @CSC:n numeroissa 2/1998 ja 3/1998 [Haa98b, HS98].

Matlabin ja C:n yhteiskäyttö

K: *Onko mahdollista kutsua Matlabista omaa C-kielistä ohjelmaa siten, että palautusarvo saadaan Matlabiin?*

V: @CSC:n numerossa 1/1998 [Sav98] käsiteltiin Fortran-ohjelmien kutsua Matlabista. Idea on sama C-kielessä: funktiokutsun parametrit välitetään MEX-tiedostoksi käännettävään C-ohjelmaan Matlabista osoittimien avulla päällystakkiohjelmasta `mexFunction`. C-kielessä tämä hoituu koneriippumattomilla standardirakenteilla. Matlabin tietotyypit ja funktiot saadaan käyttöön komennolla `#include "mex.h"`. Tiedoston hakupolku tunnetaan automaattisesti, kun käännös suoritetaan Matlabista.

Käsittelemme C-kielisenä samaa virtasilmukan magneettikentän laskevaa MEX-esimerkkiä kuin artikkelissa [Sav98]. Funktion kutsumuoto Matlabista on sama kuin Fortranilla toteutettuna:

```
>> a=0.50; I=2.5e5;
>> r=0.01:0.01:0.25; z=0.01:0.01:0.50;
>> [Br, Bz] = loop(a, I, r, z);
```

Päällystakkiohjelma toteutetaan seuraavasti:

```
void mexFunction(int nlhs, mxArray *plhs[],
  int nrhs, const mxArray *prhs[])
{
  unsigned int m, n;
  double a, I;
  double *r, *z, *Br, *Bz;

  /* Vaakavektorien r ja z koko */
  m = mxGetN(prhs[2]);
  n = mxGetN(prhs[3]);
  /* Luo m x n -matriisit Br ja Bz */
  plhs[0] = mxCreateDoubleMatrix(m, n,
    mxREAL);
  plhs[1] = mxCreateDoubleMatrix(m, n,
    mxREAL);
  /* Osoittimet kaikkiin argumentteihin */
  a = mxGetScalar(prhs[0]);
  I = mxGetScalar(prhs[1]);
  r = mxGetPr(prhs[2]);
  z = mxGetPr(prhs[3]);
  Br = mxGetPr(plhs[0]);
  Bz = mxGetPr(plhs[1]);
  /* Kutsu laskennallista aliohjelmaa */
  loop(Br, Bz, a, I, r, z, m, n);
}
```

Magneettikentän laskevat C-kieliset aliohjelmat ovat tiedostossa `loop.c`, joka löytyy WWW-osoitteesta

```
http://www.csc.fi/programming/
  examples/matlab
```

Tämä ohjelma käännetään Matlabista komennolla

```
>> mex loop.c
```

Grafiikan olioluokat

Matlabin korkeammat tason grafiikkakomennot kuten `plot` toimivat siten, että ne kutsuvat Matlabin *grafiikkaolioita* käsitteleviä alemman tason funktioita sopivilla parametreilla. Grafiikkaoliot on jaettu 11 luokkaan, joita ovat mm. `figure`, `line` ja `axes`. Käyttäjä voi puuttua kuvaajien ulkoasuun joko jo tulostettujen kuvien *kahvojen* avulla tai käyttämällä itse suoraan alemman tason grafiikkakomentoja.

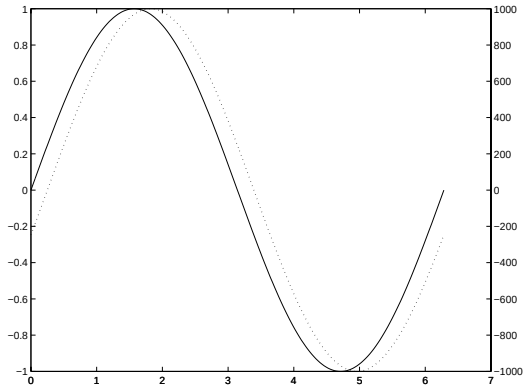
Matlabin grafiikan olioluokat ja kahvat ovat poikineet kolme kysymystä. Vastauksissa esiteltyjen vaihtoehtojen lisäksi näitä voidaan käsitellä kuvaikkunan valikosta *Properties* avattavalla graafisella editorilla tai funktioilla `gcf`, `gca` ja `gco`.Lisätietoja saa manuaalin [Mat97] luvusta *Handle Graphics*.

Kaksi *y*-akselia samaan kuvaan

K: *Haluaisin samaan 2D-kuvaan kaksi käyrää, joilla on eri skaala y-akselin suunnassa. Kuinka toisen y-akselin saa kuvan oikeaan reunaan?*

V: Seuraavassa on eräs ratkaisu kuvaajineen:

```
>> t = 0:pi/100:2*pi;
>> y1 = sin(t-.25);
>> y2 = 1000*sin(t);
>> axleft = axes;
>> plot(t,y1)
>> ylim1 = get(axleft,'YLim');
>> axright = axes;
>> plot(t,y2)
>> ylim2 = get(axright,'YLim');
>> plot(t,y1*diff(ylim2)/diff(ylim1), ...
   ':',t,y2,'-')
>> set(axright,'YAxisLocation','right')
```



Käyrät piirrettiin ensin erikseen, jotta molemmat kuvaajat skaalataan automaattisesti. Skaalaukset luetaan Matlabin kahvasta oliolle `axes` avainsanalla `'YLim'`. Sen jälkeen molemmat käyrät tulostetaan samaan kuvaan tallennettuilla skaalauksilla. Lopuksi toisen kuvaajan skaala siirrettään kuvan oikeaan laitaan muuttamalla avainsanan `'YAxisLocation'` arvoa. Esimerkissä yhtenäinen käyrä liittyy vasempaan ja katkoviiva oikeaan *y*-akseliin.

Sama voidaan tehdä myös komennolla `plotyy`:

```
>> t = 0:pi/100:2*pi;
>> y1 = sin(t-.25);
>> y2 = 1000*sin(t);
>> plotyy(t,y1,t,y2)
```

Tämä komento voi olla jopa liian älykäs, jos haluat itse ohjata tarkemmin tulostusta. Valitettavasti komennon käyttöä ei ole dokumentoitu Matlabin manuaaleissa. Listauksen lähdekoodista saa käskyllä

```
>> type plotyy
```

Histogrammin väri

K: *Kuinka histogrammin värin voi muuttaa oletusarvona olevasta sinisestä?*

V: Komennolle `hist` ei ole mahdollista määritellä väriä, joten muutetaan hiukan sen lähdekoodia. Komento `hist` kutsuu komentoa `bar`, jolle puolestaan voidaan antaa piirtoväri. Muuttamalla tiedoston `hist.m` kopiossa rivi `bar(x,nn,'hist');` esimerkiksi riviksi

```
bar(x,nn,'hist','y');
```

saadaan keltainen histogrammi.

Hilaviivatyylit

K: *Miten saadaan kuvien hilaviivat (gridline) pakkummiksi?*

V: Hilaviivamäärittelyt kuuluvat grafiikan olioluokkaan `axes`. Alla olevassa esimerkissä kierretään komennon `plot` käyttö kutsumalla suoraan funktioita `figure`, `axes` ja `line`. Avainsanojen `'XGrid'`, `'YGrid'` ja `'GridlLineStyle'` määrittelyt piirtävät hilaviivat kuvaan yhtenäisellä viivalla:

```
>> pv = ['Ma'; 'Ti'; 'Ke'; 'To'; ...
   'Pe'; 'La'; 'Su'];
>> T = [5.6 7.4 4.3 2.1 -0.4 4.9 9.5];
>> f = figure;
>> a = axes('YLim', [-5 10], 'Xtick', ...
   1:7, 'XTickLabel', pv, 'XGrid', 'on', ...
   'YGrid' , 'on', 'GridLineStyle', '-');
>> h = line(1:7, T);
```

Mahdolliset avainsanat eri olioluokissa näkee komennolla `get(a)` luokalle `axes` ja komennolla `get(h)` luokalle `line`. Vastaavasti mahdolliset arvot eri avainsanoille luokassa `axes` näkee komennolla `set(a)`. Tuloksena saatava lista näyttää avainsanalle `'GridLineStyle'` vaihtoehdot:

```
[ - | -- | : | -. | none ]
```

Lisätietoja

Matlabin käyttöesimerkkejä löytyy kirjallisuusluettelossa mainituista teoksista, esimerkiksi CSC:n oppaasta *Matemaattiset ohjelmistot* [Haa98a]. Tietolähteenä voi käyttää myös Usenet News -uutispalstaa `comp.soft-sys.matlab`. Myös CSC:n asiantuntijoilta voi kysellä Matlabiin liittyviä vinkkejä.

Kirjallisuutta

[Haa98a] Juha Haataja, toim. *Matemaattiset ohjelmistot*. CSC, 1998. Web-osoite `http://www.csc.fi/oppaat/mat.ohj/`.

[Haa98b] Juha Haataja, Matlab 5.x — kysymyksiä ja vastauksia, *@CSC*, 2/1998.

[HS98] Juha Haataja ja Ville Savolainen, Matlab 5.x — lisää kysymyksiä ja vastauksia, *@CSC*, 3/1998.

[Mat96] The MathWorks, Inc. *Using Matlab, Version 5*, 1996.

[Mat97] The MathWorks, Inc. *Getting Started with Matlab, Version 5.1*, 1997.

[Sav98] Ville Savolainen, Tehokas ohjelmointi Matlabilla, *@CSC*, 1/1998.