Aalto-yliopisto
Teknillinen korkeakoulu

MAPLE

Lempeä johdatus

Harri Hakula

7. lokakuuta, 2010

Sisällys

1 Matemaattisista ohjelmistoista

Insinööri haluaa aina parhaat työkalut. Tekniikan ylioppilas opiskelee matematiikkaa saadakseen eväät oman alansa erityisongelmien ratkaisuun. Työkalujen kehitys – taulukot, laskutikku, laskin, tietokone – on johtanut paradoksaaliseen tilanteeseen, missä Teemu Teekkari tarvitseekin syvällistä ymmärrystä laskinlaitteiden ja matemaattisten ongelmien riippuvuuksista pelkän laskurutiinin sijaan.
Matemaattiset ohjelmistot voidaan karkeasti jakaa kahteen luokkaan, symbolisiin ja numeerisiin. Jaon perusta on ohjelmiston kyky manipuloida lausekkeita, kuten esimerkiksi integroida lauseke suljetussa muodossa, symboliset ohjelmat pystyvät, numeeriset eivät. Matematiikan peruskurssissa BTT1 opetuksessa käytetään symbolisena ohjelmistona Maplea ja numeerisena MATLABia.
Ohjelmien opettelu ja käyttö on suositeltavaa. Jokainen opiskelija asettaa itse omat tavoitteensa, mutta on mahdollista, että osa kotitehtävistä on tehtävä koneella.

1.1 Symboliset ohjelmistot

TKK:lla käytetään opetuksessa ainakin kolmea symbolista ohjelmistoa:

- Maple
- MATHEMATICA
- MathCAD

Näistä vaihtoehdoista Maple ja MATHEMATICA soveltuvat laajojen kokonaisuuksien hallintaan, kun taas MathCAD toimii "alykkäänä paperina" ja on tarkoitettu perinteisen "kynä ja paperi"-laskun korvaajaksi.
Kurssin ohjelmistovalinta perustuu sekä oppikirjojen kirjoittajien että kurssin luennoitsijan mieltymyksiin.

1.2 Numeeriset ohjelmistot

MATLAB on yleisin käytössä oleva numeerinen ohjelmisto. Useilla tekniikan aloilla se on *de facto* standardi. Useissa erikoisalojen oppikirjoissa algoritmiesimerkit esitetään MATLABin esitysmuodossa.

Ohjelmistot ovat hyvin laajoja, täydellisiä ohjelmointiympäristöjä. Vilkkuva kursori ruudulla aiheuttaa helposti ahdistusta. Mutta, on hyvä muistaa että kaikkea ei voi oppia heti ja että ohjelmistojen todellinen arvo ei ole niinkään siinä, paljonko ne auttavat matematiikan opiskelussa, vaan siinä paljonko aikaa säästyy oman alan ongelmanratkaisussa opiskelun edetessä pidemmälle.

2 Maple ATK-keskuksen ympäristössä

Oletetaan nyt, että lupa-asiat ovat kunnossa ja että koneiden peruskäyttö on tuttua. Lisäksi oletetaan, että käyttö tapahtuu jonkinlaisessa UNIX-ympäristössä, minkä ei pitäisi olla ongelma Windows-käyttäjille, sillä erot ovat hyvin pienet.
Maple on tarjolla kahtena eri versiona, jotka eroavat käyttötavan mukaan.

2.1 Työarkki

Käytön opettelussa mukavampi vaihtoehto on ns. työarkkiversio. Se käynnistyy komennolla

```
xmaple
```

Maple varaa ruudulta oman tilansa, jonka sisällä voi olla yhtäaikaa useita työarkkeja.

2.2 Komentotiedosto

Usein helpompi tapa on yksinkertaisesti kirjoittaa komennot tavallisella editorilla erilliseen komentotiedostoon ja suorittaa komennot komentoriviltä.

```
maple
```

Tämä työskentelytapa soveltuu parhaiten silloin, kun tietää mitä on tekemässä.

3 Maple

Tässä luvussa käsitellään perusteita, joiden avulla ainakin tehokas laskinkäyttö sekä kurssikirjojen esimerkkien läpikäynti onnistuu kyynelittä. Maplen kehote on "suurempi kuin"-merkki (`>`). Allaolevissa esimerkeissä on annettu vain käyttäjän antamat syötteet. Syöte suoritetaan vain, jos se on kirjoitettu oikein ja päättyy rivinvaihtoon.

3.1 Manuaali

Maplen sisäinen manuaali on kattava. Työarkkiversiossa on manuaalille oma valikko, mistä voi valita esimerkiksi erilaisia hakuvaihtoehtoja. Valitettavasti manuaalia on helpoin käyttää vasta, kun tietää mitä etsii. Jos komennon eteen liitetään kysymysmerkki, `?plot`, Maple avaa kyseisen komennon manuaalin. Kaksi kysymysmerkkiä, `??`, vie manuaalin alkuun.

Manuaali

```
?plot
??
```

3.2 Ympäristö

Maplea voi käyttää tavallisena funktiolaskimena.

Laskin

```
1 + 1;
sin(2.3);
sqrt(4);
sin(1);
evalf(sin(1));
```

Huomaa, että puolipiste (`;`) kertoo Maplelle, että lauseke on suoritettava eli evaluoitava. Maple pyrkii aina säilyttämään symbolisen esityksen ja niinpä lauseketta `sin(1)` ei evaluoida numeerisesti ellei sitä erikseen pyydetä `evalf`-funktiolla.
Kaksoispiste (`:`) eroaa puolipisteestä siinä, että lausekkeen arvoa ei kaiuteta eli näytetä ruudulla. (Tämä on todella hyvä muistaa jatkossa!) Maple kuitenkin muistaa suoritetun lausekkeen arvon, vaikka sitä ei ruudulla näkyisikään. Edellisen (ajassa) suoritetun lausekkeen arvoon voi viitata prosentilla (`%`). Kaksi prosenttia vie kahden suorituksen päähän jne.

Kaiutus

```
2 + 2:
%;
```

Komentoja voi tietenkin kirjoittaa yhdelle riville useita, mutta pitkän päälle paras tapa on kirjoittaa komento per rivi luettavuuden takia.

```
2 + 2: % * 4;
```

Maple-istunnon aikana suoritetut laskutoimitukset tapahtuvat Maplen ylläpitämässä ympäristössä. Ympäristön asetukset ovat voimassa, kunnes ohjelman suoritus katkaistaan tai laskentaympäristö uudelleen käynnistetään komennolla `restart`.

Uudelleen käynnistys

```
restart:
```

Maplen ympäristössä voi olla paitsi käyttäjän laskutoimitusten arvoja, myös intunnon aikana luotuja muuttujia ja funktioita. Funktioita voi myös koota kirjastoihin ja käyttää niitä uudelleen seuraavissa istunnoissa.

3.3 Muuttujat

Lausekkeiden arvoja voi sijoittaa nimettyihin muuttujiin. Sijoitusoperaattori on `:=`.

Muuttujan arvon asettaminen

```
a := 1;
s := sin(Pi);
```

Muuttujien nimien valinnassa on hyvä miettiä käyttötarkoitusta. Jos aikoo joskus palata työarkin pariin, on hyvä antaa muuttujille nimet, joiden merkitys on riittävän selkeä myös jonkin ajan kuluttua.
Tottumista vaatii Maplen ns. täysevaluaatio eli sijoitusketjun läpikäyminen mahdollisimman pitkälle. Esimerkiksi

```
a := 2;
b := a;
a := 4;
b;
```

mutta

```
d := c;
c := 4;
d;
```

Maple etsii ensimmäisessä tapauksessa `b`:lle `a`:n arvon 2, mutta toisessa esimerkissä `d`:n arvo onkin symbolin `c` arvo, aina.
Muuttujan arvon voi poistaa ympäristöstä seuraavasti:

Muuttujan poistaminen

```
a := 'a';
```

Muuttujiin voi sijoittaa myös lausekkeita. Seuraava esimerkki laskee polynomin derivaatan `diff`-komennolla. Huomaa, että jos `x`:llä on jo jokin arvo, täysevaluaation mukaisesti lauseke voikin saada jonkin muun arvon kuin on tarkoitus. Kokeile!

```
f := x^4 + a*x + d;
diff(f, x);
```

Muutuja voi myös viitata ns. tietorakenteeseen. Maplen neljä perustietotyyppiä ovat jono, lista, joukko ja hajautustaulu. Lista on yleisimmin vastaan tuleva.

```
lista := [1,3,3,5,7];
joukko := {1,3,3,5,7};
```

Tietorakenteiden syvempi ymmärtäminen on tarpeen vain, jos kirjoittaa ei-triviaaleja ohjelmia.

3.4 Funktiot

Maplessa on mahdollista määritellä myös funktioita. Määritellään funktio, joka derivoi argumenttinsa `x`:n suhteen ja käytetään sitä saman tien kahdesti – ensin suoraan lausekkeeseen ja sitten muuttujaan talletettuun lausekkeeseen.

Funktio

```
derivate := (arg) -> diff(arg, x);
derivate(x^3 + 3*x);
f := x^3 + 3*x;
derivate(f);
```

Jos argumentteja on useita, argumentit on määriteltävä sulkujen avulla, yhden argumentin tapauksessa sulut voi jättää pois.

Lausekkeet ja funktiot ovat erilaisia. Määritellään sama polynomi kahdesti ja evaluoidaan se pisteessä 1.

```
poly := (x) -> (x + 3)^2;
poly(1);
p := (x + 3)^2;
subs(x=1, p);
```

Riemastuttavaa on se, että lausekkeen voi tarvittaessa muuttaa funktioksi `unapply`-komennolla. Jatketaan edellistä esimerkkiä.

Lausekkesta funktioksi

```
poly2 := unapply(p, x);
poly2(1);
```

3.5 Ohjelmointi

Maple sisältää erittäin rikkaan ohjelmointiympäristön. Tässä emme suinkaan mene hienouksiin, vaan tyydymme esittämään, miten funktiot voi kirjoittaa aliohjelmina. Aliohjelma eroaa funktiosta lähinnä siinä, että aliohjelman sisällä voidaan määritellä sisäisiä muuttujia, joiden arvot ovat käytettävissä aliohjelman suorituksen ajan.

Aliohjelma

```
derivateProc := proc(fun)
       return diff(fun, x);
end;
```

Lisätään derivaattaan aliohjelman argumentin neliö. Määritellään aliohjelman sisäinen eli lokaali muuttuja `sq`, johon välitulos tallennetaan.

```
derivateProc2 := proc(fun)
       local sq;
       sq := fun^2;
       return diff(fun, x) + sq;
end;
```

Aliohjelman voi tallettaa tiedostoon ja lukea myöhemmin uuteen istuntoon ilman uudelleenkirjoitusta.

Aliohjelman talletus ja luku

```
save(derivateProc2, `aliohjelma.mpl`);
read(`aliohjelma.mpl`);
```

Peruskäytössä funktioilla selviää pitkälle.

3.6 Kirjastot

Käytettävissä olevat valmiit rutiinit on jaettu kahteen osaan: ytimessä olevat, aina saatavilla olevat komennot ja tarvittaessa ladattavat rutiinit. Maplea voi laajentaa ns. kirjastoilla, joita on jo perusversiossa mukana lukuisia. Erityisesti monet grafiikan ja lineaarialgebran kirjastot tulevat varmasti tutuiksi.
Kirjastot ladataan komennolla `with`. Mukavan avaruuskäyrän voi piirtää vaikkapa seuraavasti.

Kirjaston lataus

```
with(plots):
spacecurve([cos(t),sin(t),t], t=0..4*Pi);
```

Jos koko kirjastoa ei halua ladata, voi rutiiniin viitata suoraan:

```
plots[spacecurve]([cos(t),sin(t),t], t=0..4*Pi);
```

3.7 Piirtäminen

Maplen piirto-ominaisuudet ovat hyvin kehittyneet. Peruspiirtorutiinien lisäksi `plots`-kirjasto tarjoaa useita erikoistarkoituksiin sopivia komentoja. Useat komennoista jakavat samat ohjauskomennot, joten varmasti hyödyllisiä sivuja ovat `?plot[options]` ja `?plot3d[options]`.
Keskeisimmät kuvaajatyypit ovat

- käyrä
- parametrisoitu käyrä
- pinta
- tasa-arvokäyrä

Katsotaan lopuksi esimerkit eri tyypeistä. Aluksi on aina piirrettävä sini-funktio. Monta kuvaajaa voi yhdistää laittamalla funktiot hakasulkeisiin.

Käyrä

```
plot(sin(x), x=-Pi..Pi);
plot([sin(x), cos(x)], x=-Pi..Pi);
```

Parametrisoidusta käyrästä käy hyvin ympyrä. Huomaa, että kuvaaja skaalataan automaattisesti, ja ympyrä näyttää ellipsiltä. Optio `scaling=constrained` jättää mitoitukset rauhaan.

Parametrisoitu käyrä

```
plot([cos(t), sin(t), t=0..2*Pi]);
plot([cos(t), sin(t), t=0..2*Pi], scaling=constrained);
```

Pinnan piirto onnistuu myös. On hyvä muistaa, että pintojen ja tasa-arvokäyrien kuvaajien laatu riippuu usein valitusta hilakoosta, `grid=[.,.]`.

Pinta

```
plot3d(sin(x)*sin(y), x=-Pi..Pi, y=-Pi..Pi);
plot3d(sin(x)*sin(y), x=-Pi..Pi, y=-Pi..Pi, grid=[30,30]);
```

Tasa-arvokäyrä

```
plots[contourplot](sin(x)*sin(y), x=-Pi..Pi, y=-Pi..Pi);
plots[contourplot](sin(x)*sin(y), x=-Pi..Pi, y=-Pi..Pi, grid=[30,30]);
```