Kaikkien paikallisten ääriarvojen haku

30.4.2017 *Heikki Apiola*

Tiedosto: opi_minimointiLIVEn.mlx, opi_minimointiLIVEn.m

Funktio: lok_min.m (help lok_min)

Lisää esimerkkejä: ex_lok_min.m

Abstrakti

Lähdetään liikkeelle diskreetissä pistejoukossa määritellystä funktiosta. Selvitetään juurta jaksain periaatteet, joilla päästään käsiksi ääriarvojen laskentaan, tarvitsematta mitään pohjatietoja asiasta. Kehitellään Matlab/ Octave-työkalut ohjelmien perusominaisuuksia ja ajattelutapoja hyödyntäen, joilla periaatteet saadaan muutetuksi laskenta-algoritmiksi. Kun sileää (jatkuvasti dervoituvaa) funktiota f "digitoidaan", eli lasketaan arvoja rittävän tiheästi diskretoiden, saadaan samalla ohjelmalla lasketuksi halutulla tarkkuudella likiarvoja f:n ääriarvoille.

Verrataan Matlab:n valmiin funktion `fminsearch` laskemiin tuloksiin, ja koetaan jokunen yllätyskin.

Lopuksi valaistaan mahdollisuutta rinnakkaislaskentaan.

Matlab/Octave-perusteita: https://math.aalto.fi/~apiola/matlab/opas/lyhyt/

```
clear; close all;format compact     % Aseteteaan alkutila.
```

Katsotaan ensin perustapaukset:

Otetaan näytteitä sin-funktiosta:

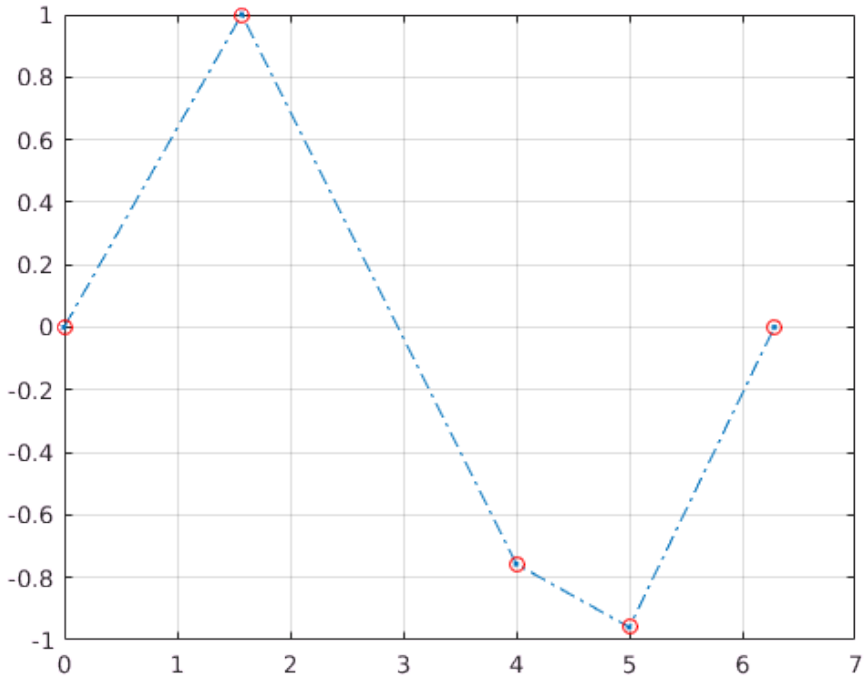
```
x=[0 pi/2 4 5 2*pi]
```

```
x =
         0    1.5708    4.0000    5.0000    6.2832
```

```
y=sin(x)
```

```
y =
         0    1.0000   -0.7568   -0.9589   -0.0000
```

```
plot(x,y,'.-.',x,y,'or');grid on;shg
```



Miten haetaan min ja max diskreetille funktiolle ?

Funktion kuvaajan muodostavat punaiset 'rinkulapisteet':

```
[x;y]
```

```
ans =
         0    1.5708    4.0000    5.0000    6.2832
         0    1.0000   -0.7568   -0.9589   -0.0000
```

Havainnot:

- Välillä (x_2, x_3) funktio kasvaa, eli $y_3 - y_2 > 0$.
- Välillä (x_2, x_3) funktio pienenee, eli $y_3 - y_2 < 0$.

SIIS: x_2 on maksimikohta.

- Välillä (x_2, x_3) funktio pienenee, samoin välillä (x_3, x_4). SIIS: x_3 ei ole min- eikä maxkohta.
- Välillä (x_3, x_4) funktio pienenee ja välillä (x_4, x_5) kasvaa. SIIS: x_4 on minimkohta.

Muistele, derivoituvan funktion tapauksen min-max-sääntöjä !

Lasketaan Matlab:lla (Octavella)

Avaimen onneen tarjoaa funktio **diff**, joka laskee argumenttivektorin peräkkäisten komponenttien erotusvektorin.

Esim:

```
x=[-1 1 1 3 0]
```

```
x =
    -1     1     1     3     0
```

```
diff(x)
```

```
ans =
     2     0     2    -3
```

Palataan tehtäväämme:

```
x=[0 pi/2 4 5 2*pi]
```

```
x =
         0    1.5708    4.0000    5.0000    6.2832
```

```
y=sin(x)
```

```
y =
         0    1.0000   -0.7568   -0.9589   -0.0000
```

Muodostetaan y-vektorin erotukset:

```
diff(y)
```

```
ans =
    1.0000   -1.7568   -0.2021    0.9589
```

Erotukset mittaavat x-vektorin sisäpisteissä x_2, x_3, x_4 tapahtuvia muutoksia.

```
[x;y]
```

```
ans =
         0    1.5708    4.0000    5.0000    6.2832
         0    1.0000   -0.7568   -0.9589   -0.0000
```

Luemme diff(y)-vektorista sen, mikä näkyy y-vektorista (ja kuvasta):

- Pisteessä x_2 erotus muuttuu $+ \rightarrow -$ siis max-piste.
- Pisteessä x_3 $- \rightarrow -$, siis ei ääriarvoa.
- Pisteessä x_4 $- \rightarrow +$, siis min-piste.

Muutoksen suunnan sisäpisteissä x_1, x_2, x_3 näemme suoraan katsomalla erotusvektorin erotuksia, siis toisen derivaatan diskreettiä vastinetta.

Olemme kiinostuneita erotusten merkeistä, joten kannattaa soveltaa vielä *sign*-funktiota

```
sdy=sign(diff(y))
```

```
sdy =
     1    -1    -1     1
```

```
d2y=diff(sdy)  % Erotusvektorin erotukset (Diskreetti 2. derivaatta)
```

```
d2y =
    -2     0     2
```

Tästäpä näkyy suoraan, että x_2 on maksimipiste, x_3 ei ole ääriarvo, x_4 on minimipiste. Jotta voisimme kehittää koodin, joka tekee valinnan puolestamme, tarvitsemme vielä yhden tehokkaan Matlab-työkalun, **loogisen valinnan**.

Kts. https://math.aalto.fi/~apiola/matlab/opas/lyhyt/perusteet.html#loogind

```
maxind=d2y < 0
```

```
maxind = 1×3 logical array

   1   0   0
```

```
minind=d2y > 0
```

```
minind = 1×3 logical array

   0   0   1
```

Sisäpisteiden muodostamista vektoreista

```
xin=x(2:end-1)
```

```
xin =
    1.5708    4.0000    5.0000
```

```
yin=y(2:end-1)
```

```
yin =
    1.0000   -0.7568   -0.9589
```

valitaan ehtojen maxind ja minind avulla ne, joiden kohdalla on 1 (true)

```
xmax=xin(maxind)
```

```
xmax = 1.5708
```

```
ymax=yin(maxind)
```

```
ymax = 1
```

```
xmin=xin(minind)
```

```
xmin = 5
```

```
ymin=yin(minind)
```

```
ymin = -0.9589
```

Kaksi viime vaihetta voidaan yhdistää helposti luettavaksi "idiomiksi" tyyliin:

```
xmin=xin(d2y>0)  % Valitaan ne, joissa toinen differenssi > 0
```

```
xmin = 5
```

HUOM! Edellä suoritetut operaatiot ovat kaikki vektorioperaatioita. Niin ollen ne toimivat ilman muutoksia miten pitkille x- ja y-vektoreille tahansa ja antavat **kaikki** (x,y)-vektoriparin määrittelemän diskreetin funktion max- ja min-pisteet.

Tehtävä: Määritä kaikki min-ja max-pisteet datalle:

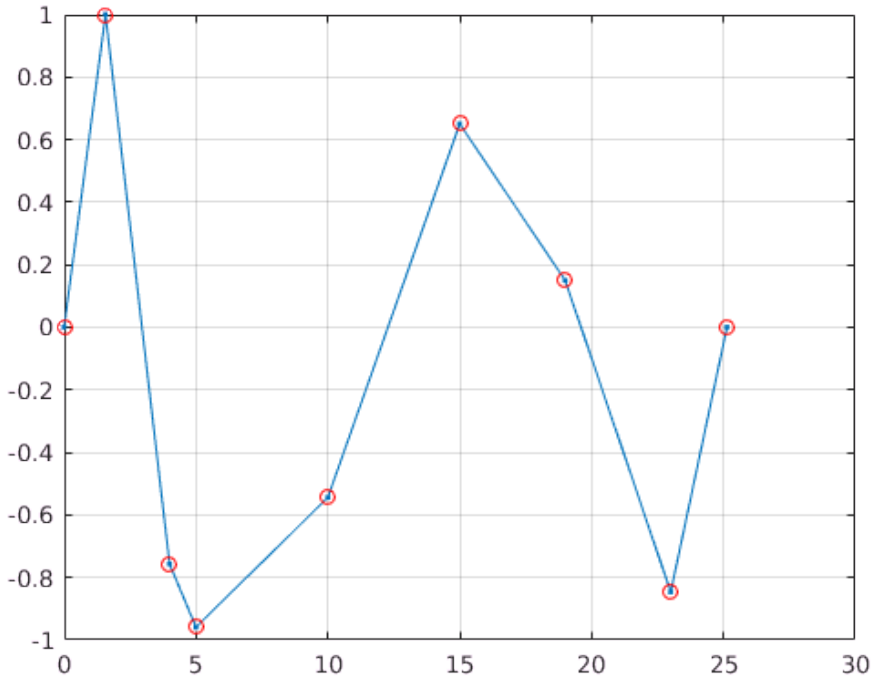
```
x=[0 pi/2 4 5 10 15 19 23 8*pi]
```

```
x =
         0    1.5708    4.0000    5.0000   10.0000   15.0000   19.0000   23.0000 ···
```

```
y=sin(x)
```

```
y =
         0    1.0000   -0.7568   -0.9589   -0.5440    0.6503    0.1499   -0.8462 ···
```

```
plot(x,y,'.-',x,y,'or');grid on;shg
```



Ratkaisu:

Tee yllä olevan mallin mukaan (kopioimalla koodirivit). Suorita sitten piirto tähän tapaan:

|clf; plot(x,y,'.-.',xmin,ymin,'*r',xmax,ymax,'ob');grid on;shg; legend('Datapisteet |

`yhdistettyinä','minimit(red))','maksimit(blue)')`

Kirjoitetaan yleispätevä funktio

Pitkän päälle koodirivien kopiointi käy turhauttavaksi. Siksipä kannattaa tässä vaiheessa kirjoittaa homman hoitava funktio, jolle annetaan argumentteina x ja y-vektorit. Funktion tulee palauttaa vektorit `xmin, ymin, xmax, ymax`

Matlab-funktion, tässä nimeltään `lok_minmax`, perussyntaksi on:

```
%{
function [xin,ymin,xmax,ymax]=lok_minmax(x,y)
 % Alkukommentteja...
 % ... alkukommentteja (help lok_minmax tulostaa alkukommentit)
 matlab-komentoja,
 periaatteessa ne yllä kopioinnin kohteena olleet
 ...
 xmin= ...;
 ymin= ...;
 xmax=...;
 ymax= ...;
%}
```

Funktion lok_minmax koodi:

Huom! >> help lok_minmax tulostaa alkukommenttirivit.

```
type lok_minmax
```

```
function [xmin,ymin,xmax,ymax] = lok_minmax(x,y)
% Etsitaan diskreetilla x-akselilla annettujen y-arvojen paikalliset
% minimit ja maksimit
% Esim 1)
% x=0:0.1:7; y=sin(5*x); [xmin,ymin,xmax,ymax]=lok_minmax(x,y)
% plot(x,y,xmin,ymin,'x');ylim([-1.1,1.1]);grid on;shg
% Esim 2)
% x=linspace(0,7);y=sin(5*x); [xmin,ymin,xmax,ymax]=lok_min(x,y)
%
% Voidaan siis käyttää myös sileän funktion ääriarvojen numeeriseen
% laskentaan.

dysign=sign(diff(y)); % Differenssivektorin (y(2)-y(1), y(3)-y(2), ...) sign
d2y=diff(dysign);     % 2. differenssi (+-2 tai 0)
xin=x(2:end-1);       % x-sisäpisteet
yin=y(2:end-1);       % vastaavat y-pisteet
xmin=xin(d2y>0);      % Poimitaan sisäpistevektorista xin ne, joissa
                      % 2. differenssi > 0.;
ymin=yin(d2y>0);      % vastaavat y-pisteet

xmax=xin(d2y<0);      % Poimitaan sisäpistevektorista xin ne, joissa
                      % 2. differenssi < 0.;
ymax=yin(d2y<0);      % vastaavat y-pisteet

end
```

Esimerkki 2: suurempi määrä min/max-pisteitä, sekä sileitä, että piikkejä.

Kehitellään sopiva nopeammin heilahteleva funktio, jolla myös "piikikkyttä"

$$f(x) = 0.1\,x + (\cos x)\,|\sin(5\,x)|$$

Määritellään Matlab-funktioksi. Alla olevat linkit auttanevat selittämään niitä seuraavaa Matlab-koodiriviä.

Kts: Funktiomääritys Matlab:ssa (Lyhyt opas)

Miksi piste? (.*) Kts:

https://math.aalto.fi/~apiola/matlab/opas/lyhyt/perusteet.html#taulukkooperaatio Taulukko-operaatiot, pisteittäin, Lyhyt opas

Ja myös syksyn 2016 Aalto Scip-kurssi:

https://math.aalto.fi/opetus/MatOhjelmistot/2016syksySCI/Lectures/runBasic_Arrays.pdf Aalto-Scip-Kurssimatskua: "Arrays" Haku: (CTR-F pointwise)

```
close all
f=@(x)0.1*x+cos(x).*abs(sin(5*x))
```

```
f = function_handle with value:

    @(x)0.1*x+cos(x).*abs(sin(5*x))
```

```
vali=[0 10];
xx=linspace(0,10,300);
plot(xx,f(xx));title('Piikkejä ja sileitä ääriarvoja');grid on;ylim([-0.7 1.7]);shg
```

Nautiskellaan funktiomme lok_minmax avulla.

Väli [0,10] jaetaan 300:aan osaväliin, joten osavälin pituus:

```
h=10/300
```

```
h = 0.0333
```

Tämä on siis taattu tarkkuus, jolla seuraavassa laskettavat ääriarvot saadaan.

```
x=linspace(0,10,300);
y=f(x);
[xmin,ymin,xmax,ymax]=lok_minmax(x,y);
hold on
plot(xmin,ymin,'*r',xmax,ymax,'ob');ylim([-0.7 1.7]);
```


```
absvirhemax = 10/300
```

```
absvirhemax = 0.0333
```

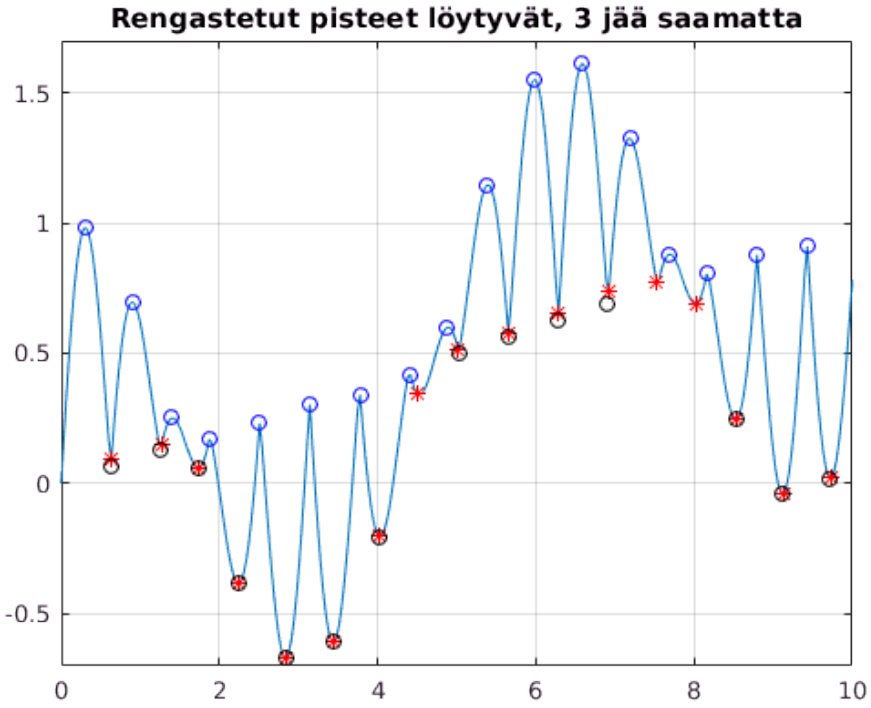
```
% ans = 0.0333
```

Kokeillaan valmista Matlab-funktiota fminsearch

Funktio `fminsearch` hakee yhtä minimiä, kun annetaan rittävän tarkka alkuarvaus.

Annetaanpa alkuarvaukseksi kukin yllä laskettu likiarvo vuorollaan:

```
N=length(xmin);
tarkennetutminimit=zeros(1,N);
for k=1:N
tarkennetutminimit(k)=fminsearch(f,xmin(k));
end
xminT=tarkennetutminimit;
yminT=f(xminT);
%figure(1)
plot(xminT,yminT,'ok',xmax,ymax,'ob');ylim([-0.7 1.7]);
title('Rengastetut pisteet löytyvät, 3 jää saamatta')
```




```
[xmin([7 8 9 13 14 15]);tarkennetutminimit([7 8 9 13 14 15])]
```

```
ans =
    4.0134    4.5151    5.0167    7.5251    8.0268    8.5284
    4.0296    3.4393    5.0265    9.1190    8.5254    8.5254
```

```
%{
ans =
    4.0134    4.5151    5.0167    7.5251    8.0268    8.5284
    4.0296    3.4393    5.0265    9.1190    8.5254    8.5254
%}
```

Ylärivillä on siis omalla ohjelmallamme saadut pisteet, joita käytetään alkuarvoina **fminsearch**-funktiolle. Kuten kuvasta näkyy, 3 minimiä jää saavuttamatta tässä "tarkennnuksessa". Kun katsotaan komponentteja 7 ja 8, huomataan, että alkuarvauksen kasvattaminen johtaa tuloksen pienenemiseen. Käsittelemme tätä alkuarvon valintaproblematiikkaa tarkemmin seuraavassa blogikirjoituksessa funktion nollakohdan määritystehtävässä Newtonin menetelmän yhteydessä.

```
[xmin;tarkennetutminimit]
```

```
ans =
    0.6355    1.2709    1.7391    2.2408    2.8428    3.4448    4.0134    4.5151 ···
    0.6283    1.2566    1.7324    2.2422    2.8359    3.4393    4.0296    3.4393
```

Koko tulos ei näy tässä, siksi alla komentoikkunasssa näkyvä täydellisenä:

```
%{
ans =
```

```
  Columns 1 through 10

    0.6355    1.2709    1.7391    2.2408    2.8428    3.4448    4.0134    4.5151    5.0167
    0.6283    1.2566    1.7324    2.2422    2.8359    3.4393    4.0296    3.4393    5.0265

  Columns 11 through 17

    6.2876    6.9231    7.5251    8.0268    8.5284    9.1304    9.7324
    6.2832    6.9115    9.1190    8.5254    8.5254    9.1191    9.7225
%}
diff(ans,1)
```

```
ans =
   -0.0071   -0.0143   -0.0067    0.0014   -0.0069   -0.0055    0.0163   -1.0757 ···
```

```
%{
ans =

  Columns 1 through 10

   -0.0071   -0.0143   -0.0067    0.0014   -0.0069   -0.0055    0.0163   -1.0757    0.0098

  Columns 11 through 17

   -0.0045   -0.0116    1.5939    0.4986   -0.0031   -0.0114   -0.0099
%}
```

Virheraja näyttää pitävän paikkansa, paitsi komponenttien 8, 9, 13, 14, 15 kohdalla. MUTTA: Niistä lähtien fminsearch suppeneekin kohti "väärää" minimiä. Eli meidän menetelmämme on luotettavampi, joskin epätarkempi. Parannusta saadaan suorittamalla tarkennettu käsittely niille alkuarvoille, jotka fminsearch ajaa virherajan ulkopuolelle.

Jos tarkkuusvaatimus ei ole kovin suuri, voidaan tyytyä oman ohjelmamme käyttöön, jossa saadaan tarkkuutta lisää joko jakamalla koko väli tiheämmin (jokaista dekadia kohti dekadi lisää, aika tehotonta, tosin Matlab:ssa viuhuu lujaa). Parempi tapa: Jaetaan kukin osaväli (xmin(k)-h,xmin(k)+h) esim. 100:aan osaan.

Harjoitustehtävä:

Tee vastaava selvitys maksimipisteille. Huomaa, että `fminsearch` sopii funktion *f*maksimien etsintään tyyliin

```
xmax= -fminsearch(@(x) -f(x),x0),
```

koska $\max f(x) = -min(-f(x))$

Rinnakkaislaskentaa, parallell toobox

Viimeisen vuosikymmenen aikana on algoritmien suunnittelussa yleistynyt enenevässä määrin rinnakkaisuuden hyödyntäminen. Tämä tarkoittaa sitä, että ohjelma pyritään jakamaan toisistaan riippumattomiin osiin, jotka voidaan suorittaa samanaikaisesti. Fyysisesti sen tekee mahdolliseksi tietokonearkkitehtuuri, jossa ison muistin ja yhden tehokkaan prosessorin sijasta on jaettu muisti ja monta prosessoria. Tällaisia suuria "klustereita" on mm. Aalto-yliopiston tritonus-klusteri ja CSC-superlaskentakeskuksen Taito, ym. CSC: Matlab-rinnakkaislaskentaa

Toisaalta rinnakkaislaskentaa voi harjoitella kotikoneellakin, jos siinä on vähintään 2-ydinprosessori.

Edellä esitetty algoritmi soveltuu erinomaisesti rinnakkaislaskentaan, kunkin alkuarvon laskemisen jälkeinen tarkentaminen jakautuu luonnollisesti riippumattomiin osiin.

Matlabissa on rinnakkaislaskentaan erikoistunut "**parallell toolbox**", jolla voidaan suorittaa tämänkaltainen rinnakkaistus hyvin helposti. Harjoittelu kotikoneella on hyvin vaivatonta, monessa tapauksessa koodia ei tarvitse muuttaa lainkaan siirryttäessä isompaan klusteriin. Palaan aiheeseen seuraavissa blogikirjoituksissa. Viittaan tässä kurssiin, jonka pidin syksyllä 2016 yhdessä Juha Kuortin kanssa:

https://math.aalto.fi/opetus/MatOhjelmistot/2016_2_syksySCI/

Optimointia ja nollakohtien hakua usemman muuttujan funktioille

Tehokkuuden tarve lisääntyy huomattavasti, kun muuttujia tulee lisää, jolloin rinnakkaislaskennan edut moninkertaistuvat. Näistä aiheista jatketaan, kunhan ehditään.