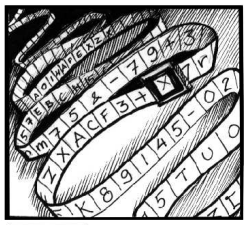


Numeerista matematiikkaa Python-kielellä

Antti Rasila
Tutkija
Matematiikan ja tilastotieteen laitos, Helsingin yliopisto

Johdanto

Edellisessä Solmun numerossa käsiteltiin yksinkertaisten ohjelmien kirjoittamista Python-kielellä. Tässä osassa on tarkoitus edetä varsinaisiin numeerisen matematiikan ongelmiin ja niihin liittyviin matemaattisiin ohjelmointitehtäviin. Käsiteltävät ongelmat keskittyvät differentiaali- ja integraalilaskennan peruskursseilla esiintyviin yhden muuttujan reaaliarvoisiin funktioihin. Aikomuksena on jatkaa kirjoitusta myöhemmin tässä käsittelemättä jäävistä aiheista, kuten numeerisesta lineaarialgebrasta ja visualisoinnista. Aluksi esitellään muutamia tavallisia menetelmiä funktion nollakohdan etsimiseksi.

Välinpuolitusmenetelmä

Välinpuolitusmenetelmän idea on yksinkertainen ja geometrisestikin ilmeinen. Menetelmä perustuu seuraavaan Bolzanon lauseena tunnettuun tulokseen:

Lause. [Myr, s. 91] Olkoon funktio f suljetulla välillä $[a, b]$ jatkuva ja $f(a) < 0$, $f(b) > 0$ (tai $f(a) > 0$, $f(b) < 0$). Tällöin on olemassa ainakin yksi välin piste z, jossa $f(z) = 0$.

Lähtökohtana siis on annettu väli $[a, b]$, $a < b$ ja jatkuva funktio f, jonka nollakohtaa etsitään ja jolla on erimerkkiset arvot välin päätepisteissä. Tutkitaan nyt pistettä $c = (a+b)/2$. Saadaan joko $f(c) < 0$, $f(c) > 0$ tai $f(c) = 0$. Viimeisessä tapauksessa nollakohta[1] on löytynyt ja voidaan lopettaa. Koska f:llä on annetun välin päätepisteissä erimerkkiset arvot joko $f(a)$ tai $f(b)$ on erimerkkinen $f(c)$:n kanssa. Saadaan siis, että joko väli $[a, c]$ tai $[c, b]$ toteuttaa Bolzanon lauseen ehdot funktiolle f; sovelletaan samaa menettelyä tähän. Koska annettu funktio f on jatkuva, päästään toistamalla mielivaltaisen lähelle funktion oikeaa nollakohtaa.

```
# Valinpuolitusmenetelma
from math import *
import sys
# funktio, jota tarkastellaan
def fun(x): return cos(x)-2*x
n=20                  # puolitusten lkm
a,b=-8.0,10.0         # aloituspisteet
# tarkastetaan funktion arvojen etumerkit
```

[1] Koska liukulukua ei pidä verrata suoraan nollaan, yhtälö $f(c) = 0$ on tulkittava epäyhtälöksi `abs(f(c))<eps`, missä `eps` riippuu koneen laskentatarkkuudesta.

```
if (fun(a)<0) & (fun(b)>0): m=1.0
elif (fun(a)>0) & (fun(b)<0): m=-1.0
else: sys.exit(1)    # virhe, poistutaan
print "askel     piste        funktion arvo"
for i in range(n):
        c=(a+b)/2
        if m*fun(c)<=0: a=c
        else: b=c
        print "%4d%14.6f%14.6f"%(i,c,fun(c))
```

Tuloste:

```
askel     piste        funktion arvo
   0      1.000000     -1.459698
   1     -3.500000      6.063543
   2     -1.250000      2.815322
   3     -0.125000      1.242198
   4      0.437500      0.030814
   5      0.718750     -0.684871
   6      0.578125     -0.318761
...
  16      0.450272     -0.000214
  17      0.450203     -0.000047
  18      0.450169      0.000037
  19      0.450186     -0.000005
```

Kiintopisteiteraatio

Tarkastellaan funktiota $F(x) = x - f(x)$. Yhtälön $f(x) = 0$ ratkaiseminen voidaan tulkita F:n avulla yhtälön $F(x) = x$ ratkaisemiseksi. Tämän yhtälön ratkaisuja kutsutaan funktion F kiintopisteiksi.

Aloitetaan jostakin välin $[a, b]$ pisteestä x_0. Määritellään nyt funktion F iterointien jono $x_0, x_1, \ldots$ kaavalla $x_{i+1} = F(x_i)$. Tunnetusta Banachin kiintopistelauseesta (todistettu esim. [MNV, s. 169]) seuraa, että tämä jono suppenee kohden funktion kiintopistettä, jos seuraavat ehdot ovat voimassa:

1. $F([a, b]) \subset [a, b]$,
2. $|F(x) - F(y)| \leq L|x - y|$ jollakin $L < 1$ kaikille $x, y \in [a, b]$.

Käsiteltävissä esimerkeissä iteraation suppenemista voi tutkia valitsemalla $L = \max\{|f'(x)| : x \in [a, b]\}$.

```
# Kiintopisteiteraatio
from math import *
from whrandom import *
# funktio
def fun(x): return cos(x)
# iteroitava funktio
def iterf(x): return x-fun(x)
# aloituspiste
x=20.0*random()
n=10  # iterointien lkm
print "askel  piste    funktion arvo"
for i in range(n):
        x=iterf(x)
        print "%4d%10.6g%14.6g"%(i,x,fun(x))
```

Tuloste:

```
askel  piste    funktion arvo
   0   10.6745     -0.315614
   1   10.9901   -0.00548963
   2   10.9956   -2.7573e-08
   3   10.9956  -4.28612e-16
   4   10.9956  -4.28612e-16
   5   10.9956  -4.28612e-16
   6   10.9956  -4.28612e-16
   7   10.9956  -4.28612e-16
   8   10.9956  -4.28612e-16
   9   10.9956  -4.28612e-16
```

Newtonin menetelmä

Newtonin menetelmä, jota myös kutsutaan Newtonin ja Raphsonin menetelmäksi, on tunnetuimpia menetelmiä yhden reaalimuuttujan yhtälön juuren löytämiseksi. Menetelmässä tarvitaan tietoa sekä funktiosta $f(x)$ että sen derivaatasta $f'(x)$.

Menetelmän ideana on, että funktiota approksimoidaan sen tangentilla pisteessä x_i, ja etsitään piste, jossa tangenttisuora leikkaa x-akselin. Tämän pisteen x-koordinaatti valitaan seuraavaksi tarkastelupisteeksi x_{i+1}. Iteraatio perustuu funktion f Taylorin sarjaan, joka antaa seuraavaan yhtälön pisteen x ympäristössä:

$$f(x + \delta) \approx f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2 + \ldots$$

Riittävän pienillä δ:n arvoilla, tämä johtaa seuraavaan approksimaatiokaavaan yhtälön $f(x+\delta) = 0$ ratkaisulle:

$$\delta \approx -\frac{f(x)}{f'(x)}.$$

Soveltamalla kaavaa toistuvasti saadaan parempia approksimaatioita ratkaisulle. Menetelmän ongelmana on, että jos iteraatio osuu funktion derivaatan nollakohtaan (lokaali maksimi tai minimi), iteraatioaskelta ei voida ottaa. Geometrisesti tämä tarkoitaa, että funktion tangentti tarkasteltavassa pisteessä ei leikkaa x-akselia. Tämän ongelman ratkaisemiseen on olemassa menetelmiä, joita ei kuitenkaan käsitellä tässä.

```
# Newtonin menetelma
from whrandom import *
from math import *
import sys
# funktio
def fun(x): return cos(x)-2.0*x
# derivaatta
def deriv(x): return -sin(x)-2.0
n=8        # askelten lkm
```

```
x=10.0*(random()-0.5) # aloituspiste
print "askel    piste    funktion arvo"
for i in range(n):
        if abs(deriv(x))<1e-8: sys.exit(1)
        x=x-fun(x)/deriv(x)
        print "%4d%12.5g%14.5g"%(i,x,fun(x))
```

Tuloste:

```
askel    piste    funktion arvo
   0     -1.5681        3.1389
   1      1.5708       -3.1416
   2      0.5236      -0.18117
   3     0.45113    -0.0023048
   4     0.45018   -4.0287e-07
   5     0.45018   -1.2212e-14
   6     0.45018   -1.1102e-16
   7     0.45018             0
```

Numeerinen derivaatta

Edellä käsitellyn Newtonin menetelmän heikkous on, että menetelmän soveltamiseen vaaditaan tietoa kulloinkin käsiteltävän funktion derivaatasta. Tämä ei ole ongelma, jos funktio on annettu helposti käsiteltävällä kaavalla. Saattaa kuitenkin olla, että ratkaistavassa ongelmassa esiintyvä funktio on sellainen, että sillä ei ole mitään varsinaista kaavaa, vaan tietoa on ainoastaan funktion saamista arvoista. Tällaisen ongelman ratkaisuun voidaan soveltaa numeerista derivointia. Numeerisen derivoinnin käyttäminen helpottaa myös ohjelmoijan työtä, koska derivaattaa ei tarvitse laskea symbolisesti ja uudelleenkirjoitettavaa koodia on siten vähemmän.

Tarkastelemalla suoraan erotusosamäärää pienillä h:n arvoilla saadaan seuraava karkea esitys numeeriselle derivaatalle pisteessä x_0.

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h},$$

missä h on jokin pieni luku, esim. $h = 10^{-8}$. Parempaan tarkkuuteen päästään esimerkiksi viiden pisteen approksimaatiokaavalla [AS, 25.3.6], jota ei kuitenkaan esitellä tässä[2].

```
# Numeerinen derivointi: numder.py
# yksinkertainen toteutus
from math import *
# funktion f derivaatta pisteessa x
def numdf(f,x):
        dx = 1e-8     # pieni luku
        return (f(x+dx)-f(x))/dx
```

Testiohjelma:

```
# Numeerinen derivointi: testiohjelma
from math import *
from numder import *
# funktio
def f(x): return sin(x)
# symbolisesti laskettu derivaatta
def dfunc(x): return cos(x)
# paaohjelma alkaa
print "piste derivaatta  num.deriv.   virhe"
for j in range(10):
        x=0.1*j
        print "%4.1f%10.6f%12.6f%14.6e"%\
         (x,numdf(f,x),dfunc(x),numdf(f,x)-dfunc(x))
```

Tuloste:

```
piste derivaatta  num.deriv.   virhe
 0.0  1.000000    1.000000  0.000000e+00
 0.1  0.995004    0.995004 -1.061836e-10
 0.2  0.980067    0.980067  5.238188e-11
 0.3  0.955336    0.955336 -1.429860e-09
 0.4  0.921061    0.921061 -5.412757e-09
 0.5  0.877583    0.877583  2.850324e-10
 0.6  0.825336    0.825336 -3.944140e-09
 0.7  0.764842    0.764842  2.297794e-09
 0.8  0.696707    0.696707  5.195307e-09
 0.9  0.621610    0.621610 -4.768086e-09
```

Seuraava ohjelma käyttää numeerista derivointia yhtälön juuren hakuun Newtonin menetelmällä.

```
# Newtonin menetelma numeerisella
# derivoinnilla
from whrandom import *
from math import *
from numder import *
import sys
# funktio
def fun(x): return cos(x)-2.0*x
n=8        # askelten lkm
x=10.0*(random()-0.5) # aloituspiste
print "askel    piste    funktion arvo"
for i in range(n):
        if abs(numdf(fun,x)<1e-8: sys.exit(1)
        x=x-fun(x)/numdf(fun,x)
        print "%4d%12.5g%14.5g"%(i,x,fun(x))
```

Tuloste:

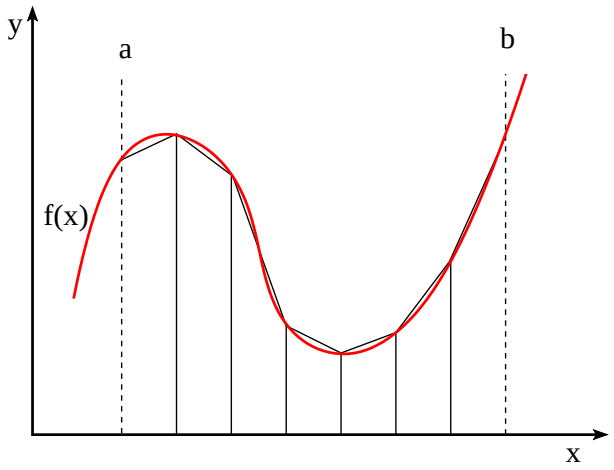
```
askel    piste    funktion arvo
   0      1.4697       -2.8384
   1     0.52193      -0.17699
   2     0.45109    -0.0022036
   3     0.45018   -3.6831e-07
   4     0.45018   -1.0658e-14
   5     0.45018             0
   6     0.45018             0
   7     0.45018             0
```

[2] Viiden pisteen kaavan toteutus on annettu www-sivulta löytyvässä esimerkkiohjelmassa, jonka nimi on `numder2.py`.

Numeerinen integrointi

Derivaatan tapaan myös funktion määrätty integraali jollakin välillä $[a, b]$ voidaan laskea numeerisesti. Tähän on käytettävissä useita menetelmiä, joista tässä esiteltävä on kaikkein yksinkertaisin.

Ajatus on, että väli $[a, b]$ jaetaan n:ään (yleensä, mutta ei välttämättä yhtäpitkään) väliin. Funktion paloittain lineaarinen approksimointi erikseen kullakin näistä väleistä johtaa integraalille esitykseen summana puolisuunnikkaan muotoisten alueiden pinta-aloista.



Kaavana tämä voidaan kirjoittaa seuraavasti:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n-1}(x_{i+1}-x_i)(y_{i+1}+y_i)/2,$$

missä $x_1 < x_2 < \ldots < x_n$ ovat jakopisteet välille $[a, b]$, $x_1 = a$, $x_n = b$ ja $y_i = f(x_i)$. Tämän kaavan, jota kutsutaan puolisuunnikaskaavaksi, antaman arvon voi helposti laskea ohjelmalla.

```
# Numeerinen integrointi
from math import *
# integroitava funktio
def fun(x): return sin(x)
# integrointivali
a,b=0.0,pi
n=1000              # jakopisteiden lkm
h=(b-a)/(n-1.0)     # jakovalin pituus
s=0.0
for k in range(n-1):
        s=s+fun(a+(k+1)*h)+fun(a+k*h)
s=s*h/2.0
tarkka=cos(a)-cos(b)
print "tarkka          numeerinen       virhe"
print "%12.6e%14.6e%14.6e"%(tarkka,s,abs(s-tarkka))
```

Tuloste:

```
tarkka          numeerinen       virhe
2.000000e+00  1.999998e+00  1.648229e-06
```

Linkkejä

- Pythonin kotisivu
  `http://www.python.org/`
- Antti Laaksonen: Johdatus Python-ohjelmointiin
  `http://www.ohjelmointiputka.net/opas.php?tunnus=python`
- Beginner's Guide to Python
  `http://www.python.org/topics/learn/`
- Python Tutorial by Guido van Rossum
  `http://docs.python.org/tut/tut.html`
- Python ja tieteellinen laskenta (erilaisia laajennuksia Python-kieleen)
  `http://www.python.org/topics/scicomp/`
- Python-kielestä ohjelmoinnin kouluopetuksessa
  `http://www.seapig.org/PythonInSchools`
- Toinen artikkeli samasta aiheesta
  `http://www.4dsolutions.net/ocn/overcome.html`
- Sivusto ohjelmoinnin kouluopetuksesta tytöille. (sivustolla keskitytään pääasiassa pedagokiikkaan, mutta käytettävä opetuskieli on Python)
  `http://www.seapig.org/GirlProgrammers`
- Python in the Mathematics Curriculum by Kirby Urner (artikkeli Python-kielen käytöstä matematiikan opetuksessa)
  `http://www.python.org/pycon/dc2004/papers/15/`

Viitteet

[AS] Milton Abramowitz, Irene A. Stegun: *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, New York, Dover, 1965. Tämä kirja on saatavissa myös verkosta:
`http://jove.prohosting.com/~skripty/`

[Myr] Lauri Myrberg: *Differentiaali- ja integraalilaskenta, osa 1*, Helsinki, Kirjayhtymä, 1977.

[MNV] Matti Mäkelä, Olavi Nevanlinna, Juhani Virkkunen: *Numeerinen matematiikka*, Helsinki, Gaudeamus, 1982.