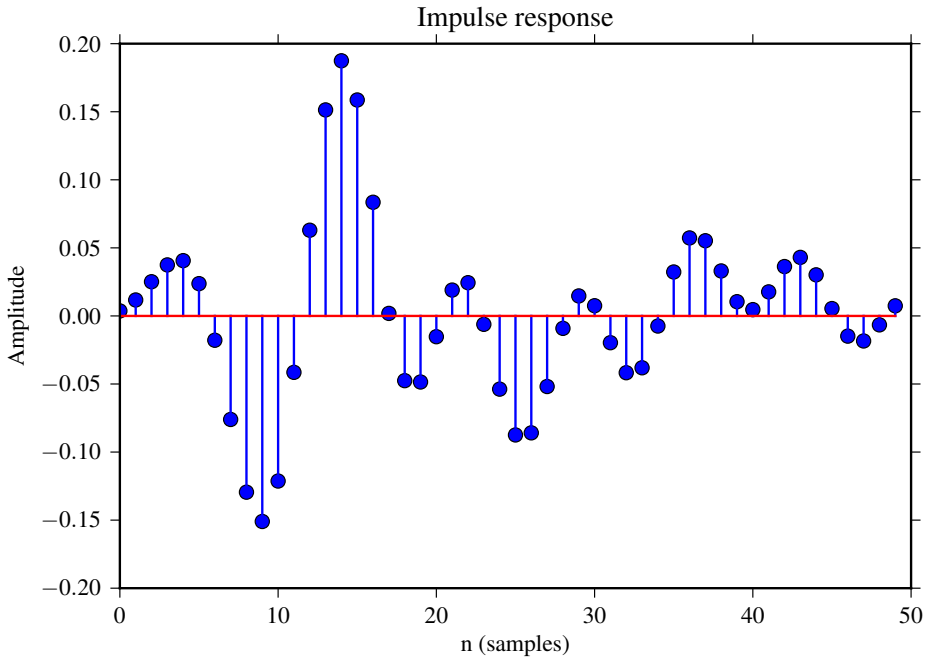
SCIENTIFIC PYTHONIN ALKEET ELI:

SciPy OPAS

Matti Pastell
matti.pastell@helsinki.fi
Maataloustieteiden laitos, Helsingin yliopisto

12. tammikuuta 2010



http://papers.mpastell.com/scipy_opas.pdf

Sisältö

1 Johdanto

Tämän oppaan tarkoitus on opettaa SciPyn eli Scientific Pythonin [1] peruskäyttö ja hyödyllisiä komentoja. Opas sisältää suurinpiirtein samat osiot kuin MATLAB oppaani ja SciPy tarjoaakin hyvän ilmaisen vaihtoehdon MATLABILLE. SciPyn käyttänä syntaksi on myös hyvin lähellä Matlab-koodia. SciPyn etuna on se, että voit käyttää ohjelmissa myös Python[2]-ohjelmointikielen muita ominaisuuksia ja luoda suhteellisen helposti graafisia sovelluksia, joita saat levittää vapaasti edelleen.

Pyrin täydentämään opusta tarpeen mukaan ja viimeisimmän version löytää kotisivuiltani. Suositan aloittelijalle komentojen testaamista samalla, kun luet opasta. Parannusehdotukset ja virheilmoitukset voi lähettää sähköpostitse matti.pastell@helsinki.fi. Opas on kirjoitettu LaTeX- kuvauskielellä.

Usein kysytään eikö olisi helpompaa käyttää Exceliä, SPSS tms. Komentorivipohjaiset ohjelmat saattavat tuntua aluksi vaikeilta käyttää, mutta niiden ominaisuudet ovat ylivoimaisia esim. suurten datamäärien käsittelyssä, omien funktioiden kirjoittamisessa ja simuloinnissa. Jatkuvissa mittauksissa syntyy tarve automatisoida datan käsittely. Useita satoja tai tuhansia tiedostoja analysoivan ohjelman teko jollain ohjelmointikielellä (Python, R, MATLAB yms.) on verraten helppoa ja säästää valtavasti aikaa. Lisäksi hieman nopeammissa mittauksissa syntyy niin paljon dataa, että sen käsittely Excelissä on käytännössä mahdotonta. Esim. 50 Hz mittaustaajuudella vuorokauden mittauksessa kertyy n. 4,2 miljoonaa datapistettä eli n. 63 Excelin täyttä saraketta.

[1] SciPy http://www.scipy.org
[2] Python www.python.org

2 Ohjelman hankinta ja käytön aloitus

SciPy on avoimen lähdekoodin tieteellisen laskennan kirjasto Python ohjelmointikieleen. SciPy käyttää NumPy kirjastoa laskutoimituksiin eli saadaakseen ohjelman käyttöönsä tarvitsee asentaa yhdistelmä SciPy+Numpy+Python+Matplotlib, joista viimeksimainittu on kuvien piirto kirjasto. Windowsille helpoin tapa on asentaa Python(x,y)-paketti www.pythonxy.com, jokas sisältää edellä mainitut ja lisäksi Spyder-kehitysympäristön. Useimmissa Linux jakeluista edellämainitut on helppo asentaa pakettienhallinnan kautta ja Mac asennusta varten löydät ohjeet ohjelmien kotisivuilta.

Käynnistettyäsi Python tulkin (valikoista tai komentoriviltä komennolla: python) sinun tarvitsee kertoa, mitä kirjastoja haluat käyttää. Saat SciPyn funktiot ja Matplotlibin käyttöösi antamalla komennon:

```
from scipy import *
from pylab import *
```

3 Muuttujat ja peruslaskutoimitukset

3.1 Muuttujien määrittely ja peruslaskutoimitukset

Peruslaskutoimitusten teko on helppoa. Muuttujia ei tarvitse erikseen määritellä vaan määrittely tapahtuu automaattisesti, kun muuttujalle annetaan arvo esim. a = 2. Huomaa, että muuttujien kirjankoolla on väliä eli $a \neq A$. Taulukossa 1 on esitetty peruslaskutoimitusten teko ja peruskomentoja data käsittelyyn.

Ohjelmat käsittelevät dataa matriisimuodossa tai vektorimuodossa. Muuttuja, jossa on vain yksi rivi tai sarake on vektori ja muuttuja, jossa on monta saraketta ja riviä on matriisi (array). Vektori tarkoittaa tässä yhteydessä peräkkäistenlukujen järjestettyä joukkoa eikä sillä yleensä ole geometristä merkitystä.

Myös matriisimuotoisilla muuttujilla voidaan suorittaa skalaarilaskutoimituksia eli kertoa esimerkiksi vektorin kaikki elementit vakiolla tai toisen vektorin vastaavilla elementeillä. Vektoreista ja matriiseista voidaan valita osia käyttämällä indeksejä.

Taulukko/matriisimuotoisena suurenkin datamäärän käsittely on joustavaa joustavaksi. Yleensä perusmittausdatan käsittelyssä ei tarvita osaamista matriisilaskennassa, vaan halutaan esimerkiksi, kertoa ja jakaa koko datamäärä esim. kalibrointikertoimellatms. tämä käy komentoriviltä hyvin helposti. Usein laskutoimituksia halutaan suorittaa myös riveittäin eli esim. laskea jokaisen rivin summa, keskiarvo tai keskihajonta.

SciPyssä on muutamia esim. Matlabista ja R:stä poikkeavia ominaisuuksia, jotka on hyvä muistaa:

- Pythonille tarvitsee kertoa että kyseessä on desimaaliluku tai ohjelma pyöristää laskutoimituksien tulokset kokonaisuvuksi eli laskutoimitus 3/2 antaa vastauksen 1, mutta 3.0/2 tai 3./2 vastauksen 1.5.
- Lukujoukkoja luodessa välin viimeinen luku ei sisälly joukkoon esim. komento arange(1.,11) luo vektorin jossa on elementit 1-10.
- Pythonissa indeksit alkavat alkavat nollasta eli muuttujan a ensimmäinen elementti saadaan valittua komennolla a[0]. Matriisien indeksit annetaan samoin kuten matematiikassakin eli järjestyksessä [rivi, sarake]

Taulukko 1: Peruslaskutoimitukset ja muuttujien indeksointi SciPy:ssa

Operaatio ja *huomioita*	**Komento, muuttujat a ja b ovat vektoreita ja muuttuja X ja Y matriiseja**
Muuttujien luominen *Vektori: luvut 1-10*	a = 1. a = arange(1. , 11) X = array([[3, 2], [4, 1], [6, 3]])
Perusoperaatiot elementeittäin * + -/	2. + 5.5; a-3; a*b; X/Y
Potenssit	2**3; a**2; X**10
Neliöjuuri	sqrt(a)
Osajoukon valinta *matriisin indeksit järjestyksessä [rivi, sarake]*	b = a[0:5] Y = X[0:2, 2:5]
Elementtien määrä	len(x); shape(X)
Summa *sarakkeittain* *riveittäin*	sum(x); sum(X) sum(X, 0) sum(X, 1)
Keskiarvo *sarakkeittain* *riveittäin*	mean(x); mean(X) mean(X, 0) mean(X, 1)
Keskihajonta *sarakkeittain* *riveittäin*	std(x); std(X) std(X, 0) std(X, 1)
Kumulatiivinen summa *sarakkeittain* *riveittäin*	cumsum(x); cumsum(X) cumsum(X, 0) cumsum(X, 1)
Vektori ja matriisitulo	dot(a,b); dot(X,Y)
Transponointi	transpose(X)
Ajettavan ohjelman keskeytys	CTRL + C
Satunnaislukujen luonti *normaalijakaumasta*	rand(100) randn(10,10)
Muuttujan poisto	del a
Komennon ohje	help(mean); help(std)

Taulukko 2: Kuvien piirto

Operaatio	**Komento, x ja y ovat vektoreita**
Viivakuvaaja	plot(x,y); plot(x,y, '–r')
Pylväskuvaaja	bar(x,y)
Histogrammi	plot(x,y)
Boxplot	boxplot(x,y)
Viivan selite	legend('viiva')
Akselien nimet	xlabel('x-akseli'); ylabel('y-akseli')
Kuvaajan otsikko	title('kuvaajan otsikko')
Akselien skaalaus	axis([x-pienin, x-suurin, y-pienin, y-suurin])
Useampi samalle 'sivulle'	subplot(221); numerot = rivien määrä, sarakkeiden määrä, monesko kuva
Kuvan talennus	savefig('kuva.pdf'); savefig('kuva.png', dpi = 300)

4 Kuvaajat

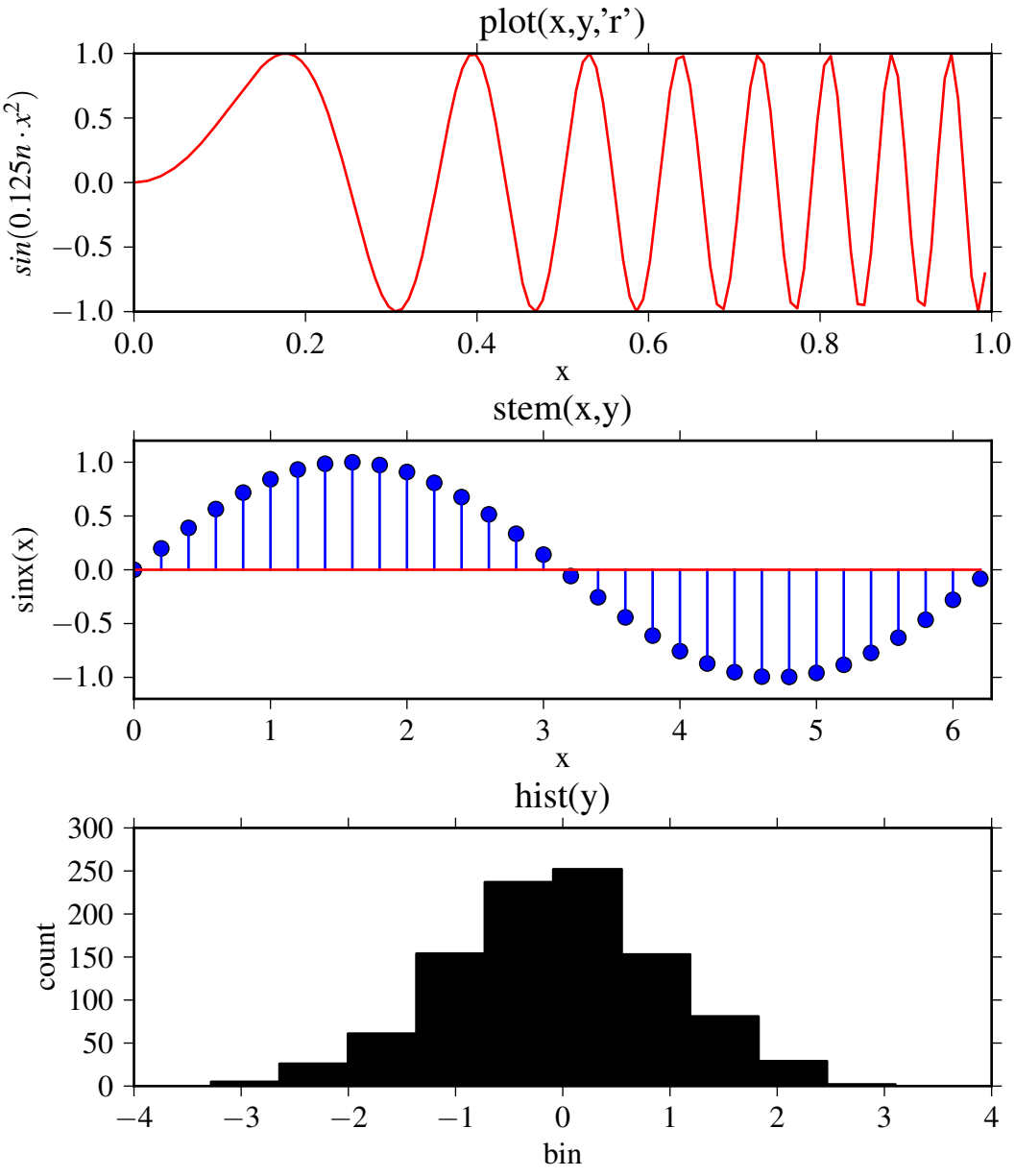
Usein datasta halutaan myös kuvaaja. Scipyssä on mahdollisuus piirtää useita erilaisia kuvaajia Matplotlib - kirjaston avulla, alla muutamia yksinkertaisia esimerkkejä. Joskus kuvaaja saadaan näkyviin plottauskäskyn jälkeen vasta antamalla komento show(), tämä riippuu Matplotlibin sen hetkisistä asetuksista jotka vaihtelevat esim. käyttöliittymien ja käyttöjärjestelmien välillä. Verkosta löytyy myös paljon hyviä ohjeita kuvien laatimiseen, kuten `www.scipy.org/Cookbook/Matplotlib/`. Taulukossa 2 on listattu perusplottauskomentoja ja alla on esimerkki niiden käytöstä. Esimerkin tulos näkyy kuvassa 1.

Kuvan tallentaminen tiedostoon onnistuu käskyllä `savefig('kuva.pdf')`, tiedostomuoto määritellään antamalla kuvalle oikea pääte, tuettuja ovat mm. png, pdf, eps, svg. Kuvan voi tallentaa myös kuvaajan valikosta klikkaamalla levykkeen kuvaa. Vektorimuotoisia kuvia tallentaessa laatu on aina hyvä, mutta esimerkiksi png kuvalle kannattaa valita 300dpi:n resoluutio jos kuvia aikoo käyttää esim. gradussa.

4.1 Esimerkki kuvaajista

Ao. esimerkissä esitellään muutamia kuvanpiirtokomentoja, lopputulos kuvassa 1.

```
from pylab import *

#Tehdään 4 kuvaajaa samalle sivulle, 1 kuvaaja
subplot(221)
#Luodaan data
n = 128.
x = arange(n)/n
y = sin(0.125*pi*n*x**2)
#Piirretään kuva, punainen viiva
plot(x,y,'r')
title("plot(x,y,'r')")
xlabel('x')
#Matplotlib tukee Latexin käyttöä tekstissä
ylabel(r'$sin(0.125n \cdot x^2)$')

#Toinen kuva sivulle
subplot(222)
x = arange(0, 2*pi, 0.2)
y = sin(x)
stem(x,y)
axis([0, 2*pi, -1.2, 1.2])
xlabel('x')
ylabel('sinx(x)')
title('stem(x,y)')

subplot(223)
y = randn(1000)
hist(y)
title('hist(y)')
xlabel('bin')
ylabel('count')

subplot(224)
boxplot(y)
ylabel('y')
title('boxplot(y)')

#Tallennetaan kuva
savefig('kuvat.pdf')
```



Kuva 1: Muutamia peruskuvaajatyyppejä

5 Tiedostojen käsittely ja hakemistot

Tiedostojen ja hakemistojen käsittelyn on Pythonissa os-moduulissa [3] ja glob-moduulissa [4] on kätevä komento tiedoston listaukseen ehtojen mukaan. Alla esimerkkejä käytöstä:

```
#Tuodaan kirjasto
import os
#Listataan hakemiston sisältö
os.listdir('.')
#Siirrytään hakemistoon data
os.chdir('data')
#Tarkistetaan missä hakemistossa ollaan.
os.getcwd()
#Listataan kaikki tekstitiedostot
import glob
glob.glob('*.txt')
```

NumPy-kirjastossa on funktiot loadtxt ja savetxt numeroarvoja sisältävien ASCII - tiedostojen lukemiseen ja tallentamiseen. Sekä funktiot load ja save NumPyn oman binääriformaattia varten ja scipy.io kirjastosta löytyy funktiot loadmat ja savemat Matlabin datatiedostoille. Excel tiedostojen lukemiseen ja kirjoittamiseen sopivat kirjastot ovat nimeltään xlrd ja xlwt. Muutamia esimerkkejä:

```
from pylab import *
#Luetaan numeerinen data, jossa on yksi otsikkorivi
data = np.loadtxt('data.txt', skiprows = 1)
#Tallennetaan data NumPyn formaattiin
data=dlmread('data.txt','\t',1,0);
#Avataan tallennettu tiedosto
load tiedosto.mat
```

[3] http://docs.python.org/library/os.html
[4] http://docs.python.org/library/glob.html

6 Ohjelmointi

6.1 Omat ohjelmat

Komentorivipohjaisten ohjelmien todellin hyöty tulee esille, kun aikaisemmissa kappaleissa esitellyistä komennoista kerätään oma ohjelma. Esimerkiksi kuvien piirto kappaleen esimerkki on oma ohjelmansa, joka luo esimerkkidataa ja tekee niistä kuvan. Ohjelma voi sisältää omia funktioita, toistorakenteita ja ehtolauseita tms.

Python ohjelmat ovat tekstitiedostoja, joissa on peräkkäin suoritettavia komentoja. Tiedostoille annetaan yleensä pääte .py. Ohjelman voi tehdä esim. Pydeen editorilla, mutta myös millä tahansa muulla editorilla.

Pythonissa eri ohjelmointirakenteet erotetaan toisistaan sisennyksellä eikä esim, suluilla joten komentoja edeltävä tyhjä tila on merkityksellinen osa koodia.

Ohjelma saadaan ajettua Python-komentotulkista komennolla execfile('ohjelma.py') ja suoraan käyttöjärjestelmän komentoriviltä python ohjelma.py.

6.2 Funktiot

Funktio on kätevä tapa lisätä ohjelmaan usein toistettavia laskutoimituksia. Esimerkiksi SciPyn keskiarvon laskeva komento mean on funktio, joka laskee keskiarvon. Voi tarkastella minkä tahansa funktion koodia antamalla komennon source # esim. source(mean). Funktiot määritellään ohjelman alussa.

Funktiota kutsutaan komentoriviltä ja sille annetaan haluttu määrä argumentteja (dataa, kertoimia tms.) ja myös kerrotaan mitä muuttujia funktion halutaan palauttavan. Erotuksena skriptiin funktio toimii omassa työtilassaan eli funktiossa määritellyt muuttujat eivät jää ohjelman muistiin.

Tehdään esimerkkifunktio, joka laskee yleisesti käytettäviä tilastollisia tunnuslukuja, huomaa että sisennys on merkityksellinen.

```
#Tämä on esimerkkifunktio kluvut, joka laskee siihen
# syötetyn datan keskiarvon, hajonnan ja keskiarvon
#keskivirheen.  Kutsu komentoriviltämuotoa [
#keskiarvo, keskihajonta,keskivirhe]=kluvut(arvo)
#HUOMAA, että 'keskiarvo, keskihajonta, keskivirhe
#ja x' ovat muuttujan nimiä ja niiden tilalla
#voidaan käyttää haluttuja nimiä. Muuttujassa "x" on
# käsiteltävä data, 'keskiarvo, keskihajonta ja
#keskivirhe' ovat funktion tuloksia.
```

```
def kluvut(x):
    keskiarvo = mean(x)
    keskihajonta = std(x)
    #Lasketaan datassa olevien arvojen määrä, eli n
    n = len(x)
    #Keskiarvon keskivirhe on keskihajonta/sqrt(n)
    keskivirhe = keskihajonta/sqrt(n)
    #Komentoriville palautettavat arvot
    return(keskiarvo, keskihajonta, keskivirhe)
```

Kun olet tehnyt funktion aja se python tulkissa (execfile tai liitä suoraan tulkiin). Tämän jälkeen testataan, että funktio toimii: Luodaan funktiolle annettavat arvot, 1000 satunnaislukua

```
data=rand(1000)
#Kutsutaan äsken tehtyä funktiota kluvut
#komentoriviltä ja
tallennetaan tulokset muuttujiin x, y ja z
x,y,z = kluvut(data)
```

6.3 Toistorakenteet eli silmukat

Yksi lähes joka ohjelmassa tarvittava komponentti on silmukka eli ”loop”. Silmukan sisällä olevat toiminnot toistetaan jokaisella ajokerralla. Silmukkaa tarvitaan esimerkiksi simuloinnissa tai jos halutaan suorittaa lukea monta tiedostoa ohjelmaan ja analysoida ne samalla tavalla. Silmukka ajetaan niin monta kertaa, kunnes annettu pysäytysehto täyttyy. Yleensä toiminta saadaan suoritettua käyttämällä **for** - looppia, joka ajaa komennot tietyn määrän kertoja.

```
from pylab import *
#Esimerkki ajetaan silmukka 10 kertaa, muuttuja
#kerta saa jokaisella ajokerralla uuden arvon.
#Ajamalla ohjelman saat käsityksen mitä tapahtuu
#loopissa..
#Luodaan loopissa käytettävä muuttuja vektori
vektori = repeat(0., 10)
#Aloitetaan silmukka
for (kerta = arange(1,11)):
    kerta
```

```
        #Rakennetaan loopilla vektori, joka saa aina
        # arvon kerta*2
    vektori[kerta] = kerta*2
```

6.4 Ehtolauseet: if, else ja elif

Silmukoiden lisäksi ohjelmissa tarvitaan usein ehtolauseita, joilla kontrolloidaan tiettyjen käskyjen suorittamista. Ehtolauseet toteutetaan komennoilla **if**, **else** ja **elif**. Yksinkertaisia esimerkkejä:

```
#b=3 jos a>4
if a > 4:
   b = 3

#b=3 jos a>3 ja b=6 jos a<2, muussa tapauksessa b=0
if a > 3:
   b = 3
elif a < 2:
   b = 6
else:
   b=0
```

7 Tehtävien automatisointi eli ”Batch-mode”

Usein mittauksissa kertyy suuri määrä samanlaisia tai lähes samanlaisia tiedostoja joiden käsittely halutaan tehdä mahdollisimman automaattisesti. Tämä tapahtuu tekemällä oma peräkkäisistä komennoista koostuva ohjelma, jolla voi vaikkapa laskea kaikkien hakemistossa olevien tiedostojen keskiarvon.

Esimerkkinä käytän lehmän jalkapainomittauksesta saatua dataa, jonka voi ladata internetistä http://files.mpastell.com/hdata.zip. Data on saatu mittaamalla lehmän jokaisen jalan painoa erikseen lypsyn aikana ja tiedosto koostuu neljästä sarakkeesta, joista jokaisessa on yhden jalan paino. Esimerkissä lasketaan jokaisen jalan keskipaino ja keskihajonta lehmän kokonaispaino, ja tallennetaan jokaisesta tiedostosta kuvaaja erilliseen hakemistoon.

```
# -*- coding: utf-8 -*-
#Ensimmäinen rivi mahdollistaa skandimerkit
#kommenteissa
```

```
from pylab import *
from glob import glob
from os import mkdir
#Datassa on otsikkorivi ja itse mittaustieto 7:ssä
#sarakkeessa. Harjoituksessa käytetään sarakkeita
#2-5, joissa on lehmän jokaisen jalan paino lypsyn
#aikana. Luetaan kaikki hakemiston tekstitiedostot
#muuttujaan 'tiedostot'
tiedostot = glob('*.txt')
#Lasketaan tiedostojen määrä ja luodaan hakemisto
#kuville
maara = len(tiedostot)

#Luodaan tyhjät matriisit arvojen tallennusta varten
keskiarvo = zeros((maara, 4))
keskihajonta = zeros((maara, 4))

#Ajetaan silmukka, jossa käsitellään kaikki
#hakemistossa olevat tiedostot
for kerta in arange(maara):
    #Luetaan silmukan ajokerran osoittama tiedosto
    data = np.loadtxt(tiedostot[kerta],
    skiprows = 1)
    #Valitaan kiinnostuksen kohteena oleva sarakkeet
    # 2-5
    data = data[:, 2:5]
    #Lasketaan lehmän kokonaispaino jalkapainojen
    #summana
    data = column_stack((data, sum(data, 1)))
    #Lasketaan tiedoston jokaisen sarakkeen
    #keskiarvo. Lopuksi muuttujassa on kaikkien
    #tiedostojen keskiarvot, jokainen omalla
    #rivillään.
    keskiarvo[kerta, 0:4] = mean(data,0)
    #Lasketaan keskihajonta
    keskihajonta[kerta,0:4]=std(data,0)
    #Piirretään kuvaaja ja lisätään selitteet
    plot(data)
    xlabel('measurement point')
    ylabel('weight (kg)')
    #Tallenetaan kuvaaja
```

```
        savefig('kuvat/kuva' + str(kerta) + '.png')
        #Tyhjennetään kuvaaja seuraavaa varten
        clf()

#Tulostetaan tulokset silmukan jälkeen ruudulle
print 'Keskiarvot: \n', keskiarvo
print 'Keskihajonnat: \n', keskihajonta
```

8 Signaalin käsittely

8.1 IIR - suodattimen suunnittelu

SciPyn signal kirjasto sisältää hyvät funktiot IIR-suodattimien suuuunitteluun, mutta siinä ei ole esim. vaihevasteen ja impulssibasteen laskevia funktioita. Alla olevassa esimerkissä tehdään ensin kaksi omaa funktiota ja suunnitellaan sitten elliptinen kaistanpäästösuodatin. Tulokset ovat kuvassa 2.

```
# -*- coding: utf-8 -*-
import scipy.signal as signal
from pylab import *

#Laskee ja plottaa IIR tai FIR suodattimen taajuus-
# ja vaihevasteen
def mfreqz(b,a=1):
    w,h = signal.freqz(b,a)
    h_dB = 20 * log10 (abs(h))
    subplot(211)
    plot(w/max(w),h_dB,'k')
    ylim(-150, 5)
    ylabel('Magnitude (db)')
    xlabel(r'Normalized Frequency (
    x$\pi$rad/sample)')
    title(r'Frequency response')
    subplot(212)
    h_Phase = unwrap(arctan2(imag(h),real(h)))
    plot(w/max(w),h_Phase,'k')
    ylabel('Phase (radians)')
    xlabel(r'Normalized Frequency (
    x$\pi$rad/sample)')
```

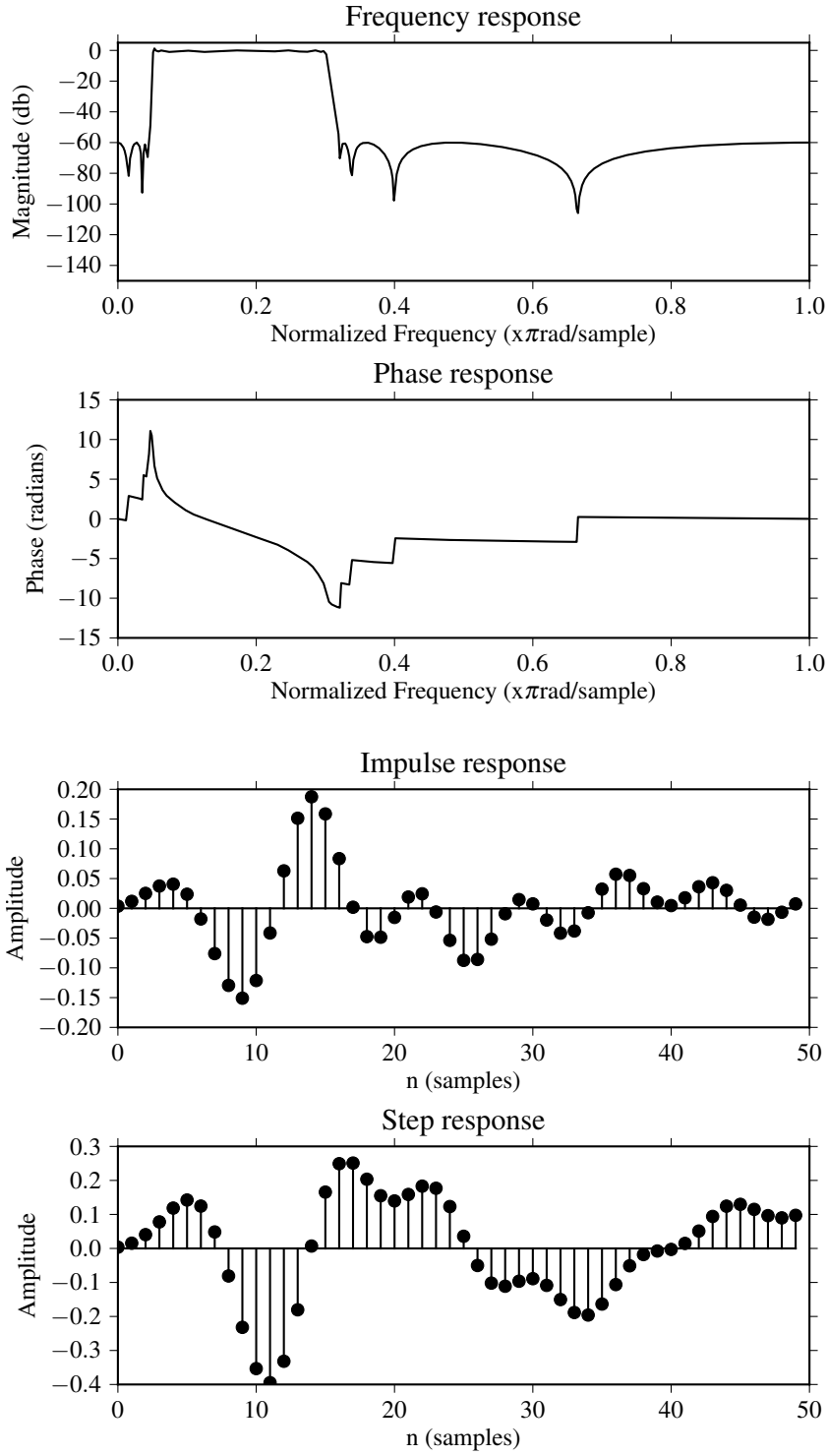
```
    title(r'Phase response')
    subplots_adjust(hspace=0.5)

#Laskee ja plottaa IIR tai FIR suodattimen impulssi-
#  ja askelvasteen
def impz(b,a=1):
        impulse = repeat(0.,50); impulse[0] =1.
        x = arange(0,50)
        response = signal.lfilter(b,a,impulse)
        subplot(211)
        stem(x, response,'k','ko','k')
        ylabel('Amplitude')
        xlabel(r'n (samples)')
        title(r'Impulse response')
        subplot(212)
        step = cumsum(response)
        stem(x, step, 'k','ko','k')
        ylabel('Amplitude')
        xlabel(r'n (samples)')
        title(r'Step response')
        subplots_adjust(hspace=0.5)


#Suunnitellaan elliptinen kaistanpäästösuodatin,
#päästökaista 0.05-0.3 kertaa Nyquistin taajuus,
#estokaistan vaimennus 60 dB, päästökaistan
#vaimennus 1 dB
b,a = signal.iirdesign(wp = [0.05, 0.3], ws= [0.02,
0.32], gstop= 60, gpass=1, ftype='ellip')
mfreqz(b,a)
savefig('figs/freqz.pdf')
figure(2)
impz(b,a)
savefig('figs/impz.pdf')
show()
```

8.2 FFT - taajuuden etsintä signaalista

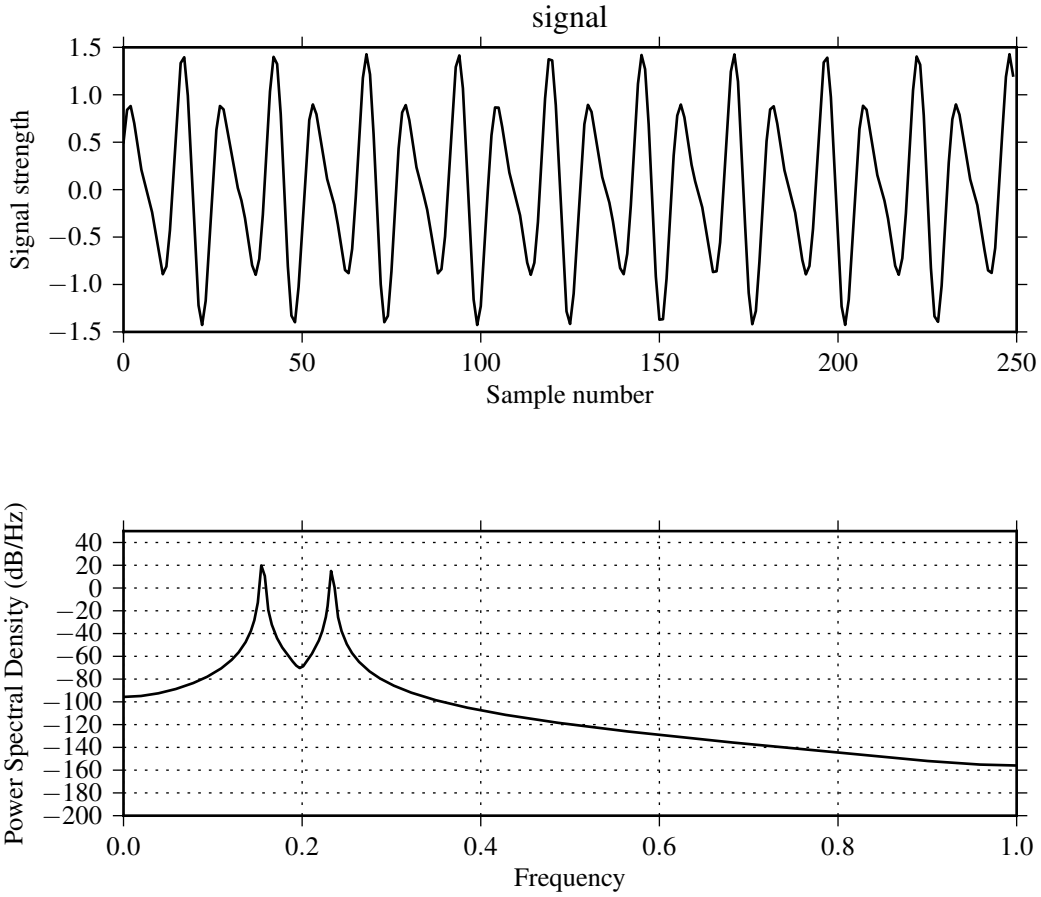
FFT (Fast Fourier Transform) on laskentamenetelmä, jolla voidaan laskea eri signaalissa esiintyvien taajuuksien voimakkuus. Muunnoksen tulosta kutsutaan te-

hospektriksi ja SciPyssä on sen laskemiseen ja piirtämiseen valmis funktio psd. Alla esimerkki FFT:n laskemisesta, tulos Kuvassa 3.

```
#Luodaan signaali
x = sin(linspace(0., 500, 1024))
+ 0.5*cos(linspace(0, 750, 1024))
subplot(211)
plot((x)[0:250])
title('signal')
ylabel('Signal strength')
xlabel('Sample number')
subplot(212)
#FFT muunnos ja plottaus, tehdään 1024 pisteelle
psd(x, NFFT = 1024)
savefig('figs/psd.pdf')
show()
```



Kuva 2: Suunnitellun IIR-suodattimen taajuus-, vaihe-, impulssi- ja askelvaste.



Kuva 3: Generoitu signaali ja sen tehospektri