Tieteellinen laskenta harjoitustyöraportti

Itsenäisten komponenttien analyysi

Harjoitustyö toteutettu Matlab veriolla R2017b (9.3.0.713579)  
Testattu myös Matlab versiolla R2015b

Mikael Janhonen 510217

Miika Tulkki 514106

# Harjoitustyön aihe

Harjoitustyön aiheena oli itsenäisten komponenttien analyysifunktioiden tutkiminen ja testaus.

Itsenäisten komponenttien analyysi (ICA) pyrkii etsimään syötesignaalijoukosta X tietyn määrän itsenäisiä signaaleja, minimoimalla tulossignaalien normaalijakautuneisuutta. Pääkomponenttianalyysi (PCA) pyrkii etsimään syötesignaalijoukosta X tietyn määrän signaaleja niin, että informaatio säilyy mahdollisimman suurena.

Harjoitustyössä käytössä on 4 signaalia, joita merkitään . Pidetään signaaleja matriisissa S, jossa rivit ovat eri signaaleja. Signaalit ja ovat informaatiota, joista ollaan kiinnostuneita ja ja ovat kohinaa. Näistä on muodostettu 4 uutta signaalia, joihin on summattu signaalit S eri painoarvoilla kerrottuna. Merkitään näitä signaaleja , ja pidetään niitä matriisissa X. Siis

Signaalit matriisissa X sisältävät siis signaaleja ja , mutta myös kohinasignaaleja ja .

Tarkoitus on etsiä alkuperäiset signaalit ja signaalijoukosta X. Tähän käytetään Matlabiin saatavilla olevaa pakettia [1]. Paketissa ovat funktiot fastICA() ja kICA(), jotka molemmat suorittavat itsenäisten komponenttien analyysin annetulle matriisille. Paketissa on myös funktio PCA(), joka suorittaa pääkomponenttianalyysin matriisille. Paketista löytyy myös apufunktioita, kuten normalizeAudio(), joka skaalaa matriisien arvot välille [-1, 1].

Funktiot palauttavat joukon signaaleja, jotka tallennetaan matriiseihin Y1, Y2 ja Y3. Saatuja signaaleja verrataan alkuperäisiin signaaleihin ja . Vertailussa käytetään euklidista etäisyyttä (L2 normi), joka lasketaan kaavalla

missä on alkuperäisen signaalin :s näyte ja on analyysistä saadun signaalin :s näyte. Mitä pienempi signaalien välinen etäisyys on, sitä paremmin alkuperäiset funktiot on saatu eroteltua signaalijoukosta X.

# Koodin toiminta

Testaus suoritetaan ajamalla ica.m tiedosto. Koodi testaa yhdellä ajokerralla signaalien erottelun jokaisen funktion (fastICA(), kICA() ja PCA()) avulla. Lähdekooditiedoston alussa voidaan määrittää, kuinka monta signaalia analysoidaan ja kuinka monta signaalia pyritään etsimään. Voidaan määrittää myös analysoitavien näytteiden määrä, ICA erottelun tyyppi ja minkä menetelmän tulokset halutaan piirtää.

Muuttujat koodin alussa:

% setup variables

L = 50000; % samples per vector

RowsToAnalyze = 4; % rows of the matricies to input to the analysing functions

RowsToFind = 3; % rows to output from the analysing functions (number of signals we are looking for)

ICAType = 'kurtosis'; % Type of ica for function fastICA. 'kurtosis' or 'negentropy'

PlotFunction = 'fastICA'; % The function results to plot {'fastICA', 'kICA', 'PCA'}

ICAType määrittelee minkälaista riippumattomuutta (nongaussianity) algoritmi pyrkii lisäämään. Se voi olla joko ’kurtosis’ eli huipukkuus, jolloin algoritmi pyrkii lisäämään käyrien terävyyttä, tai ’negentropy’, jolloin algoritmi minimoi käyrien entropiaa.

Koodissa erilaisia tehtäviä on jaettu omiin funktioihin, jotka löytyvät omista tiedostoistaan. Tehtävät on lyhyesti esitelty alla.

|  |  |
| --- | --- |
| plotMatrix() | Helpottaa signaalijoukon piirtämistä piirtämällä signaaleja sisältävän matriisin tietylle riville monta kuvaajaa sisältävään näkymään. |
| matchMatrices() | Lajittelee annetun signaaleja sisältävän matriisin toisen signaaleja sisältävän matriisin signaalien mukaan niin, että samalla rivillä olevat signaalit ovat mahdollisimman lähellä toisiaan. |
| findClosest() | Etsii signaaleja sisältävästä matriisista signaalin, joka on lähinnä annettua signaalia. |
| calculateDifference() | Laskee kahden signaalin välisen euklidisen etäisyyden. |

Tarkemmat rajapinnat ja määrittelyt löytyvät lähdekoodista jokaisen funktion yläpuolelta.

Analyysifunktiot eivät ota kantaa signaalien järjestykseen, eikä niiden skaalaukseen. Tämä vuoksi kaikki signaalit skaalataan aina välille [-1, 1]. Tulossignaalit matriisissa Y saattavat myös olla eri järjestyksessä kuin alkuperäiset signaalit matriisissa S. Tämän vuoksi tulossignaalit täytyy lajitella vastaamaan samaa järjestystä alkuperäisten signaalien kanssa. matchMatrices() on juuri tätä varten. matchMatrices() käyttää findClosest() funktiota etsimään jokaiselle tulossignaalille lähimmän alkuperäisen signaalin. Analyysissä signaalit on myös saatettu skaalata negatiivisella luvulla, jolloin tulossignaalit ovat käänteisiä alkuperäisiin signaaleihin nähden. Tämän vuoksi oikeaa signaalia etsittäessä huomioidaan myös käänteiset signaalit.

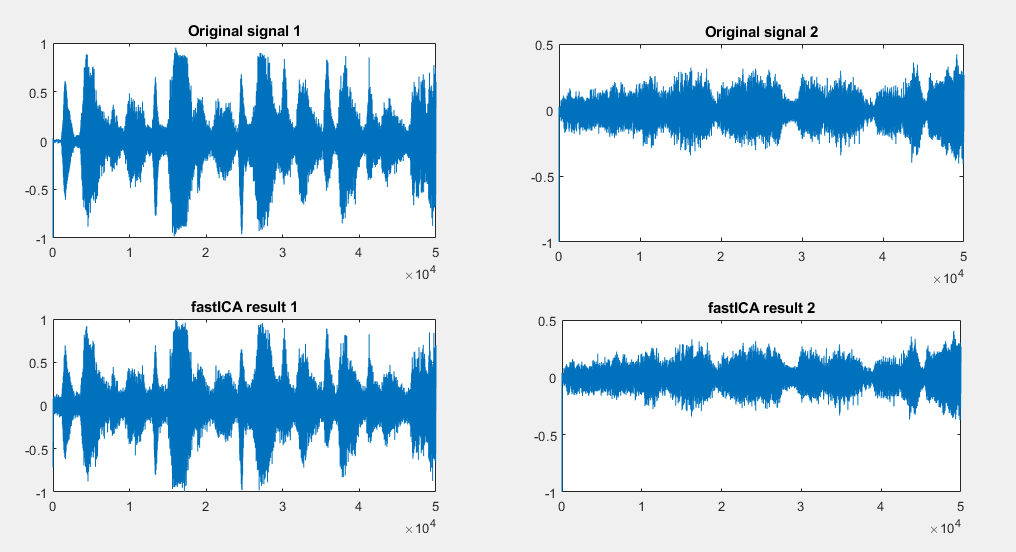
# Tulokset

Suorittamalla koodia huomataan selvästi, että funktioista fastICA() antaa parhaimmat tulokset. kICA() ja PCA() antamat tulossignaalit eroavat huomattavasti enemmän alkuperäisistä signaaleista kuin fastICA():n antamat.

Taulukko 1 - Tulossignaalien etäisyydet alkuperäisistä signaaleista. Näkyvillä olevat arvot ovat saatu, kun on  
etsitty 2 signaalia 4 signaalin joukosta ICA tyypillä 'negentropy’.

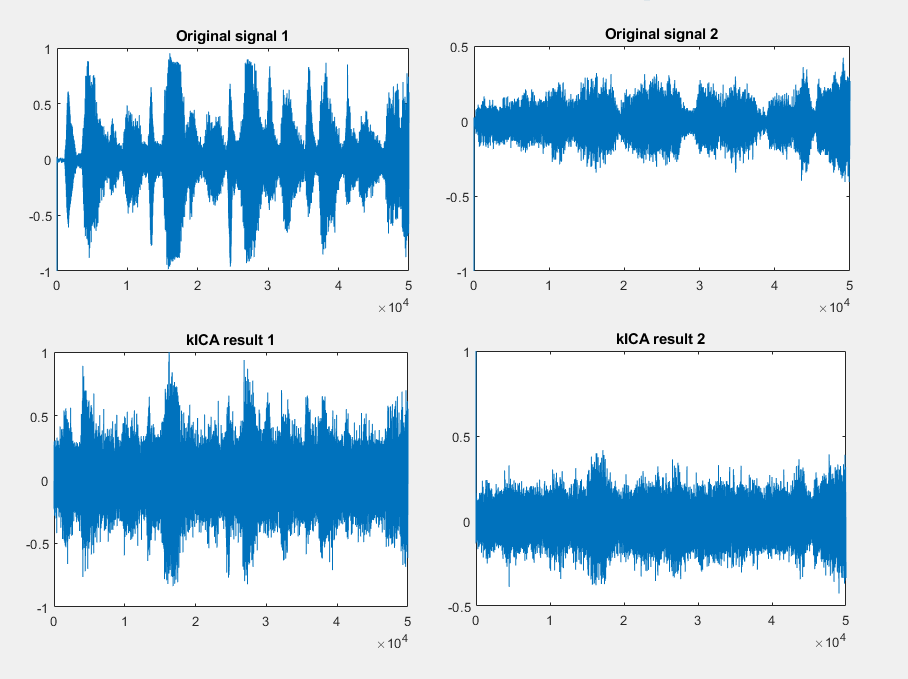
|  |  |
| --- | --- |
| analyysifunktioilla saatu tulossignaali | etäisyys alkuperäiseen |
| signal #1 from fastICA() | 9.177979 |
| signal #2 from fastICA() | 2.068266 |
| signal #1 from kICA() | 33.098347 |
| signal #2 from kICA() | 36.563862 |
| signal #1 from PCA() | 47.567326 |
| signal #2 from PCA() | 21.481082 |

Signaaleita voidaan kuunnella Matlabin komennolla sound(). Kun tuloksia kuunnellaan ja verrataan alkuperäisiin signaaleihin S ja syötesignaaleihin X, huomataan että signaaleita ei olla saatu vastaamaan täysin alkuperäisiä signaaleja, vaan niissä esiintyy vieläkin kohinaa, varsinkin ensimmäisessä signaalissa. Signaalit on kuitenkin saatu paljon selkeämmin eroteltua kuin syötteinä käytetyt signaalit, ja tulossignaaleissa ei esiinny kuultavasti toista informaatiosignaalia.



Kuva 1 - fastICA funktion tulokset näyttävät olevan kohtalaisen lähellä alkuperäisiä signaaleja. Tarkasti katsoen niissä esiintyy kuitenkin jonkin verran kohinaa.

Taulukosta 1 nähdään, että kICA() antaa huomattavasti huonompia tuloksia kuin fastICA(). Paketin dokumentaatiossa käy ilmi, että fastICA() käyttää ”nopeaa” ICA algoritmia ja kICA() käyttää ”huipukkuuden maksimointi” -algoritmiä. Siitä miksi nopea algoritmi toimii tässä tapauksessa huomattavasti paremmin meillä ei ole tietoa, mutta se johtuu todennäköisesti alkuperäisten signaalien rakenteesta.

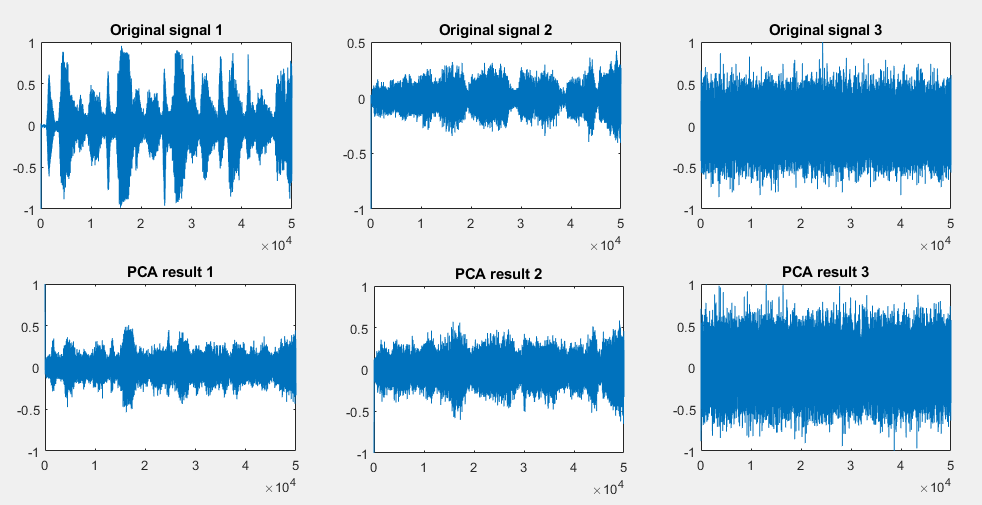


Kuva 2 – kICA funktiolla saadut tulossignaalit muistuttavat jollain tasolla alkuperäisiä signaaleja, mutta ovat kuitenkin paljon kauempana niistä, kuin fastICA menetelmällä saadut signaalit.

PCA() funktio antaa myös huonompia tuloksia kuin fastICA(). Jos PCA:n signaalit piirretään, voidaan huomata toisen löydetyistä signaaleista muistuttavan kohinasignaalia. Jos 2 signaalin sijasta etsitään 3 signaalia, saadaan tulokseksi kolme signaalia, joista kaksi näyttää sisältävän informaatiota (Kuva 3). Etäisyyksiksi näille kolmelle signaalille saadaan seuraavat arvot:

The difference in signal #1 from PCA: 67.230939  
The difference in signal #2 from PCA: 16.939757  
The difference in signal #3 from PCA: 34.125487

Signaalin 2 tapauksessa etäisyys alkuperäiseen signaaliin parani, mutta arvot ovat silti kaukana fastICA():lla saaduista arvoista. PCA ei suoranaisesti etsi itsenäisiä komponentteja, ja tämän vuoksi vain kahta signaalia haettaessa toinen tulossignaaleista on kohinaa, eikä itsenäinen signaali.



Kuva 3 – PCA funktiolla löydetyt signaalit muistuttavat alkuperäisiä signaaleja, mutta ovat kuitenkin kaukana niistä, varsinkin signaalin 1 tapauksessa.

Tarkastellaan seuraavaksi ainoastaan parhaita tuloksia antaneen fastICA() funktion tuloksia.

Taulukko 2 – fastICA() menetelmällä saatuja etäisyyksiä eri syötesignaalien ja etsittävien signaalien määrillä , sekä eri ICA tyypillä.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ICA tyyppi | Syötesignaalien määrä | Etsittävien signaalien määrä | Etäisyys #1 | Etäisyys #2 | Etäisyys #3 |
| negentropy | 3 | 1 | 25.814297 |  |  |
| negentropy | 3 | 2 | 25.518965 | 5.177481 |  |
| negentropy | 4 | 1 | 9.092178 |  |  |
| negentropy | 4 | 2 | 9,177979 | 2,068266 |  |
| negentropy | 4 | 3 | 36.449770 | 47.183422 | 51.414513 |
| kurtosis | 3 | 1 | 26.647303 |  |  |
| kurtosis | 3 | 2 | 26.460913 | 5.005537 |  |
| kurtosis | 4 | 1 | 9.166155 |  |  |
| kurtosis | 4 | 2 | 10.720318 | 3.188139 |  |
| kurtosis | 4 | 3 | 36.503781 | 49.873532 | 52.316547 |

Taulukosta 2 käy ilmi, minkälaisia arvoja saadaan eri syötteillä. Huomataan että syötesignaalien määrä vaikuttaa tuloksiin todella paljon. Kun syötesignaalien määrää nostetaan kolmesta neljään, tulokset paranevat huomattavasti. Huomataan myös, että etsittävien signaalien määrän nosto yhdestä kahteen ei juuri vaikuta ensimmäisen signaalin etäisyyteen. Kuitenkin jos etsittävien signaalien määrä nostetaan kolmeen etäisyydet nousevat huimasti. Tämä selittyy sillä, että signaalijoukossa S on vain kaksi itsenäistä signaalia, sekä kaksi kohinasignaalia, jotka ovat normaalijakautuneita. Algoritmi etsii siis epänormaalijaukautuneisuutta sieltä missä sitä ei ole.

Huomataan myös, että käytetty ICA tyyppi vaikuttaa etäisyyksiin, mutta kuitenkin aika vähäisesti. Käytetyille syötesignaleilla keskimäärin paremmat tulokset saadaan käyttämällä ’negentropy’ algoritmia. Eri algoritmien onnistuminen riippuu kuitenkin paljolti syötesignaalien rakenteesta.

# Lähdekoodi

### ica.m

% setup variables

L = 50000; % samples per vector

RowsToAnalyze = 4; % rows of the matricies to input to the analysing functions

RowsToFind = 2; % rows to output from the analysing functions (number of signals we are looking for)

ICAType = 'kurtosis'; % Type of ica for function fastICA. 'kurtosis' or 'negentropy'

PlotFunction = 'fastICA'; % The function results to plot {'fastICA', 'kICA', 'PCA'}

% constants

f = 44100; % sampling frequency

Ts = 1/f; % sample time

fprintf('Searching for %i signals from the group of %i signals\n', RowsToFind, RowsToAnalyze);

% Load audio files into matrices

s1 = audioread('samples/wave\_1.wav')';

s2 = audioread('samples/wave\_2.wav')';

s3 = audioread('samples/wave\_3.wav')';

s4 = audioread('samples/wave\_4.wav')';

S = [s1(1:L);s2(1:L);s3(1:L);s4(1:L)];

S = normalizeAudio(S);

x1 = audioread('samples/mixed\_1.wav')';

x2 = audioread('samples/mixed\_2.wav')';

x3 = audioread('samples/mixed\_3.wav')';

x4 = audioread('samples/mixed\_4.wav')';

X = [x1(1:L);x2(1:L);x3(1:L);x4(1:L)];

X = normalizeAudio(X);

% Plot the original signals

plotMatrix(S, 3, RowsToAnalyze, 1, 'Original signal');

% Plot the mixed signals

plotMatrix(X, 3, RowsToAnalyze, 2, 'Mixed signal');

% Do the different types of analysis

Y1 = fastICA(X(1:RowsToAnalyze, :), RowsToFind, ICAType, 0);

Y2 = kICA(X(1:RowsToAnalyze, :), RowsToFind);

Y3 = PCA(X(1:RowsToAnalyze, :), RowsToFind);

% Normalize results to range [-1, 1]

Y1 = normalizeAudio(Y1);

Y2 = normalizeAudio(Y2);

Y3 = normalizeAudio(Y3);

% The analysis mixes up the order of the signals so we need to match them

% ourselves.

% The matrices outputted from the analysis functions are matched by finding

% the original signal in S that is closest to each of the outputted

% signals.

Y1 = matchMatrices(S, Y1, RowsToFind);

Y2 = matchMatrices(S, Y2, RowsToFind);

Y3 = matchMatrices(S, Y3, RowsToFind);

if strcmp(PlotFunction,'fastICA')

plotMatrix(Y1, 3, RowsToAnalyze, 3, 'fastICA result');

elseif strcmp(PlotFunction, 'kICA')

plotMatrix(Y2, 3, RowsToAnalyze, 3, 'kICA result');

elseif strcmp(PlotFunction, 'PCA')

plotMatrix(Y3, 3, RowsToAnalyze, 3, 'PCA result');

end

% Print out the results

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y1(i,:));

fprintf('The difference in signal #%i from fastICA: %f\n', i, d);

end

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y2(i,:));

fprintf('The difference in signal #%i from kICA: %f\n', i, d);

end

for i = 1:RowsToFind

d = calculateDifference(S(i,:), Y3(i,:));

fprintf('The difference in signal #%i from PCA: %f\n', i, d);

end

### plotMatrix.m

% Plots the individual rows of the given matrix using subplot()

%

% Parameters:

% mat - the matrix

% rowCount - the amount of rows in the subplot

% colCount - the amount of rows to draw from the matrix, each to

% different column of the subplot

% row - the row of subplot to draw the rows of the matrix

%

function [] = plotMatrix(mat, rowCount, colCount, row, titl)

[r, c] = size(mat);

e = min([colCount, r]);

for i = 1:e

subplot(rowCount,colCount,(row-1) \* colCount + i);

plot(mat(i,:));

title(strcat(titl, {' '}, num2str(i)));

end

end

### matchMatrices.m

% Matches the rows in the second matrix with the rows of the first one

% by finding the ones that are closest to each other in terms of euclidean

% distance.

% If matrices row counts dont match, add all zero rows to mat2

%

% Parameters:

% mat1 - first matrix, the one that will be sorted

% mat2 - second matrix

% rows - the amount of rows to sort, starting with 1

%

% Returns:

% mat - sorted version of mat2

%

function [mat] = matchMatrices(mat1, mat2, rows)

mat = mat2;

[r, c] = size(mat);

[r2, c2] = size(mat1);

while r2 > r

r = r + 1;

mat(r, :) = zeros(1, c);

end

for i = rows:-1:1 % start from the end row => priorisize the first rows

[index, inverse, row] = findClosest(mat1, mat(i, :)); % find the index of the closest row

temp = mat(i, :); % swap the rows

mat(i, :) = mat(index, :);

mat(index, :) = temp;

if inverse == 1 % if inverse then inverse the row

mat(i, :) = mat(i, :) \* -1;

end

end

end

### findClosest.m

% Finds the row of the given matrix that is closest to the given vector

% Also checks inversed versions of each rows (each sample \*= -1)

%

% Parameters:

% mat - The matrix

% vec - The vector

%

% Returns:

% index - The index of the row that is closest to the given vector

% inverse - True if the row is inversed, False if not

% row - the closest row in mat, inversed if closest that way

%

function [index, inverse, row] = findClosest(mat, vec)

[r, c] = size(mat);

if length(vec) ~= c

error('Vector length and matrix column count do not match.');

else

min = calculateDifference(mat(1,:), vec);

inverse = 0;

index = 1;

row = mat(1,:);

for i = 2:r

dif = calculateDifference(mat(i,:), vec);

if(dif < min)

min = dif;

index = i;

row = mat(i, :);

end

end

for i = 1:r

dif = calculateDifference(mat(i,:), vec \* -1);

if(dif < min)

min = dif;

index = i;

inverse = 1;

row = mat(i, :) \* -1;

end

end

end

end

### calculateDifference.m

% Calculates the difference between two vectors.

% The diffenrece is the euclidean distance between the vectors.

% It is calculated with the formula Sqrt((a1 - b1)^2 + (a2 - b2)^2 + .... + (an - bn)^2)

%

% Parameters:

% vec1 - first vector

% vec2 - second vector

%

% Returns:

% diff - The difference between the vectors

%

function [diff] = calculateDifference(vec1, vec2)

if length(vec1) ~= length(vec2)

error('Vectors must have the same length.');

else

diff = 0;

for i = 1:length(vec1)

diff = diff + (vec1(i) - vec2(i))^2;

end

diff = sqrt(diff);

end

end

# Paketit

[1] https://se.mathworks.com/matlabcentral/fileexchane/38300-pca-and-ica-package