TMA4215 - Numerical Mathematics

Semester Project - Part Three

Candidate numbers
10003, 10056, 10071

November 17, 2012

Problem 1

1a)

We will here present an approximate solution to the cocktail party problem using Independent Component Analysis (ICA). Our starting point is the following system of differential equations,

$$\dot{W} = -\text{grad}(W), \quad \text{grad}(W) := (\frac{\partial \phi}{\partial W} W^T - W \frac{\partial \phi}{\partial W}^T) W. \tag{1}$$

Here $W = W(\tau)$ and the deriviative of W with respect to τ is $\dot{W}$. To ease the calculations let $\frac{\partial \phi}{\partial W} = A$. We will here show that

$$W(0)^T W(0) = I \implies W(\phi)^T W(\phi) = I, \phi \geq 0 \tag{2}$$

holds.

Consider the integral

$$\int_0^\tau \frac{d}{ds}(W(s)^T W(s)) ds = W(\phi)^T W(\phi) - W(0)^T W(0). \tag{3}$$

and let $\dot{W}$ equal $S(W)W$, where $S(W)$ is a skew-symmetric matrix. By eq. (1) $\frac{d}{ds}\left(W(s)^T W(s)\right)$ can be written out as

$$\begin{aligned}
&\frac{d}{ds}(W(s)^T W(s)) = \dot{W^T} W + W^T \dot{W} = W^T(-\text{grad}(W)) + (-\text{grad}(W))^T W \\
&= W^T(AW^T - WA^T)W + W^T(WA^T - AW^T)W \\
&= W^T(AW^T - WA^T)W - W^T(AW^T - WA^T)W \\
&= W^T S(W) W - W^T S(W) W = 0.
\end{aligned} \tag{4}$$

So that eq. (3) can be used to get

$$\begin{aligned}
&\int_0^\tau \frac{d}{ds}(W(s)^T W(s)) ds = W(\phi)^T W(\phi) - W(0)^T W(0) = 0 \\
&\implies W(\phi)^T W(\phi) = W(0)^T W(0).
\end{aligned} \tag{5}$$

Consider now the function $\gamma(\tau)$

$$\gamma(\tau) := \phi(W(\tau)), \tag{6}$$

which we will show to satisfy $\gamma(\tau_1) \leq \gamma(\tau_0)$ if $\tau_1 \geq \tau_0$.

The derivative of γ can be expressed as

$$\dot{\gamma}(\tau) = \mathrm{tr}(\frac{\partial \phi}{\partial W}^T \dot{W}(\tau)). \tag{7}$$

Where 'tr' indicates the trace of the matrix, i.e. the sum of the diagonal elements in the matrix.

While letting $W^T W = I$, it can be shown that $\dot{\gamma(\tau)} \leq 0$. The Cauchy-Schwarz inequality for inner product (eq. (8)), and the trace properties (eqs. (9) to (12)),

$$
\begin{aligned}
| < A, B > |^2 \leq < A, A > * < B, B >, \qquad < A, B >= \mathrm{tr}(A^* B) && (8)\\
\mathrm{tr}(A^T B) \leq \sqrt{\mathrm{tr}(A^T A)}\sqrt{\mathrm{tr}(B^T B)} && (9)\\
\mathrm{tr}(AB) = \mathrm{tr}(BA) && (10)\\
\mathrm{tr}(ABCD) = \mathrm{tr}(ACDB) = \mathrm{tr}(ADCB) && (11)\\
\mathrm{tr}(A + B) = \mathrm{tr}(A) + \mathrm{tr}(B) && (12)
\end{aligned}
$$

- and some hand crafting,

$$
\begin{aligned}
\dot{\gamma}(\tau) = \mathrm{trace}(\frac{\partial \phi}{\partial W}^T \dot{W}(\tau))\\
\mathrm{trace}(-A^T(AW^T - WA^T)W) = \mathrm{trace}(-A^T AW^T W + A^T WA^T W)\\
= \mathrm{trace}(A^T WA^T W - A^T AW^T W) = \mathrm{trace}(A^T WA^T W) - \mathrm{trace}(A^T AW^T W)\\
\leq \sqrt{\mathrm{trace}(A^T A)}\sqrt{\mathrm{trace}(W^T AW^T WA^T W)} - \mathrm{trace}(A^T AW^T W)\\
= \sqrt{\mathrm{trace}(A^T A)}\sqrt{\mathrm{trace}(W^T AIA^T W)} - \mathrm{trace}(A^T AI)\\
= \sqrt{\mathrm{trace}(A^T A)}\sqrt{\mathrm{trace}(A^T AW^T W)} - \mathrm{trace}(A^T A)\\
= \sqrt{\mathrm{trace}(A^T A)}\sqrt{\mathrm{trace}(A^T AI)} - \mathrm{trace}(A^T A)\\
= \sqrt{\mathrm{trace}(A^T A)}\sqrt{\mathrm{trace}(A^T A)} - \mathrm{trace}(A^T A)\\
= \mathrm{trace}(A^T A) - \mathrm{trace}(A^T A) = 0\\
\implies \dot{\gamma}(\tau) \leq 0.
\end{aligned}
$$

provide the desired result. $\dot{\gamma}(\tau) \leq 0$ for all values of τ so that $\gamma(\tau_1) \leq \gamma(\tau_0)$ if $\tau_1 \geq \tau_0$ since γ is decreasing when τ is increasing.

1b)

In the following, Matlab is used to solve eq. (1) with both the forward (eq. (13)) and backward (fig. 4) Euler method.

$$\tilde{W}_{k+1} = W_k - \alpha(G_k - W_k G_k^T W_k), \quad G_k := \frac{\partial \phi}{\partial W}_{W=W_k} \tag{13}$$

$$\tilde{W}_{k+1} = W_k - \alpha(G_{k+1} - W_{k+1} G_{k+1}^T W_{k+1}), \quad G_{k+1} := \frac{\partial \phi}{\partial W}_{W=W_{k+1}} \tag{14}$$

Here the step size is α - and each step in the implicit backward Euler method is solved by applying fixed point iteration.

Comparing the two methods to the differential equation system above, one can see that the methods assume orthogonality in that $W_k^T W_k = I$. Each step doesn't necessarily produce a resulting orthogonal matrix $\tilde{W}_k^T \tilde{W}_k = I$. Therefore, our implementation of the method allows one to perform the projection that

$$W_{k+1} = Q, \quad \text{where} \quad \tilde{W}_{k+1} = QR \tag{15}$$

by QR decomposition.

The principle behind solving the system in eq. (1) is that the sum of independent variables has a distribution closer to the Gaussian distribution, according to the central limiting theorem. The system can be used to approximate the maximum non-aussianity, as the assumption is that when the system is furthest from Guassianity it will sort out the original components. For simplicity our methods require a mean of 0, and the assumption is that the signals are statistically independent with unit variance. The estimated sources will abide by this assumption and return solutions with mean=0 and variance=1. Consider the three signals shown in fig. 1. These signals represents speaker signals that have been transformed to a mean of 0 with unit variance. By having a minimum of three microphones that record superpositions of the signals, one can use the solution of the problem posed in eq. (1) to find a demixing matrix used to estimate the original sources from the signals each microphone record.

The superpositioned signals that each microphone records are plotted in fig. 2. Note that in the plot, the signals have been transformed using a whitening procedure such that $E[\tilde{x}\tilde{x}^T] = I$.

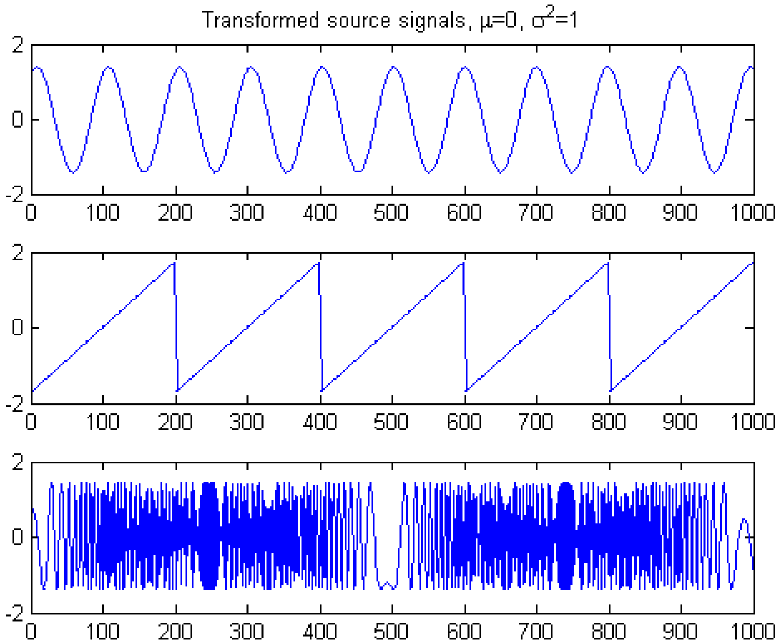

Figure 1: Plot showing the three source signals

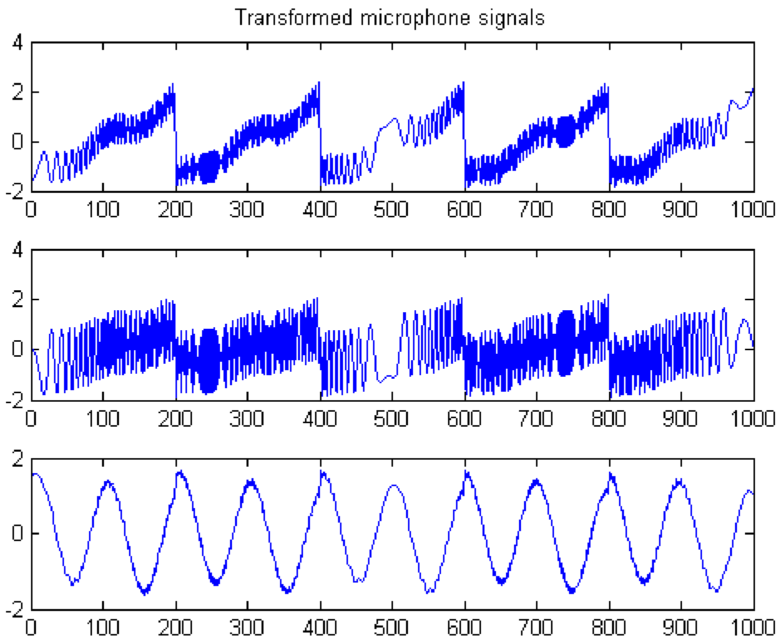

Figure 2: Plot showing the transformed signals after mixing

This way W, found through the Euler methods, will be an orthogonal demixing matrix (given that QR-projection is applied, otherwise it will not be orthogonal) that can be used to find estimates y of the original sources:

$$y(t) := W^T \tilde{x}(t) \tag{16}$$

The estimated source signals after demixing are plotted in the following figures depicting the forward (fig. 3) and backward (fig. 4) Euler method.

The plots look reasonably similar to the original sources. One can note that the signals weren't necessarily returned in the order they were put in, nor were they always returned with the correct sign, meaning solutions may appear with their phase inverted. We noted similar effects for several other mixing-matrices.

When applying the forward Euler method without the QR-projection for orthogonality in each step, the result didn't converge towards any reasonable demixing matrix. This is not surprising, since eq. (13) assumes orthogonality from eq. (1).

The variables in the implementation of the Euler method is the step-size α and whether to apply the forward or backward method. By testing different step-sizes we found that larger step-sizes requires less iterations of the Euler methods, naturally. And as long as the step-size is reasonably small (for the speaker signals α should be less than 0.5), W converged nicely. The backward Euler-method implementation includes a fixed-point iteration loop for each step of the method, which gives it an extra dimension of iterations. We didn't encounter any stability issues with the forward Euler method, therefore the extra iterations of the backward Euler method made it a comparatively slower method for finding the demixing matrix W.

One can use the error measure

$$E_k := |\phi(W_k) - \phi(W_{k-1})| \tag{17}$$

to observe the rate at which $\phi(W_k)$ approaches a minimum. The plots below (figs. 5 to 7) show E_k plotted for step-sizes 0.1, 0.02 and 0.005. As one can see from the plots, the difference from each $\phi(W_k)$ to the next starts out smaller for the smaller step sizes, but the larger step sizes quickly surpass the smaller ones to reach the smaller differences faster.

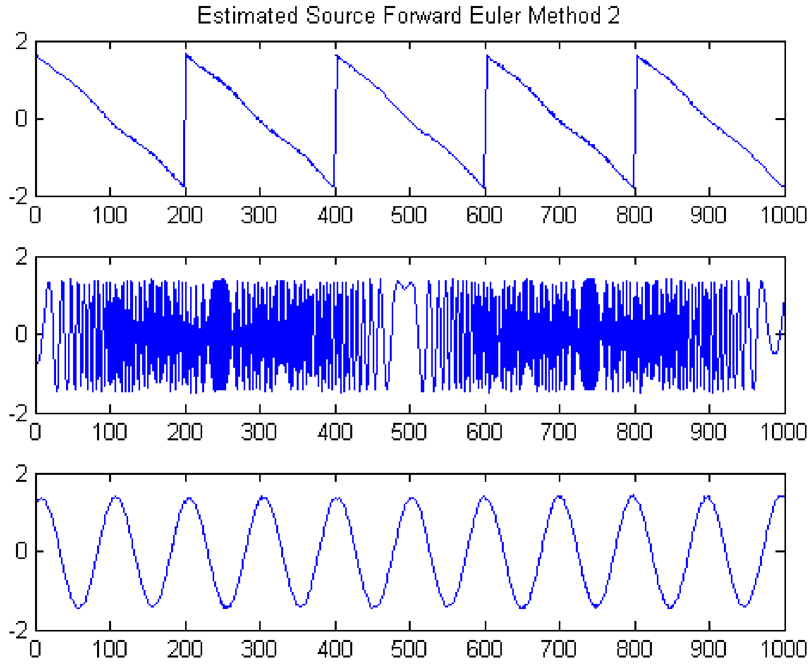

Figure 3: Plot showing the estimates of the original sources by the forward Euler method

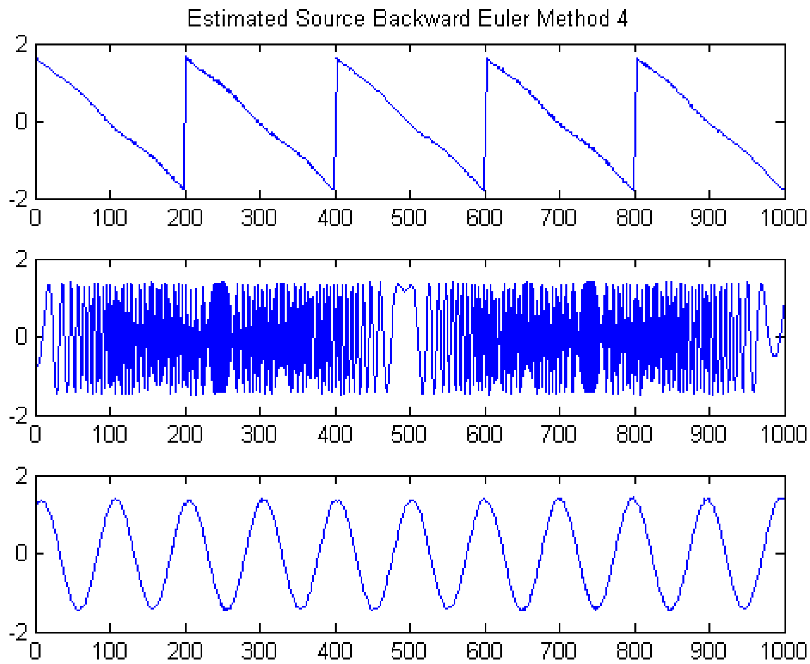

Figure 4: Plot showing the estimates of the original sources by backward Euler method

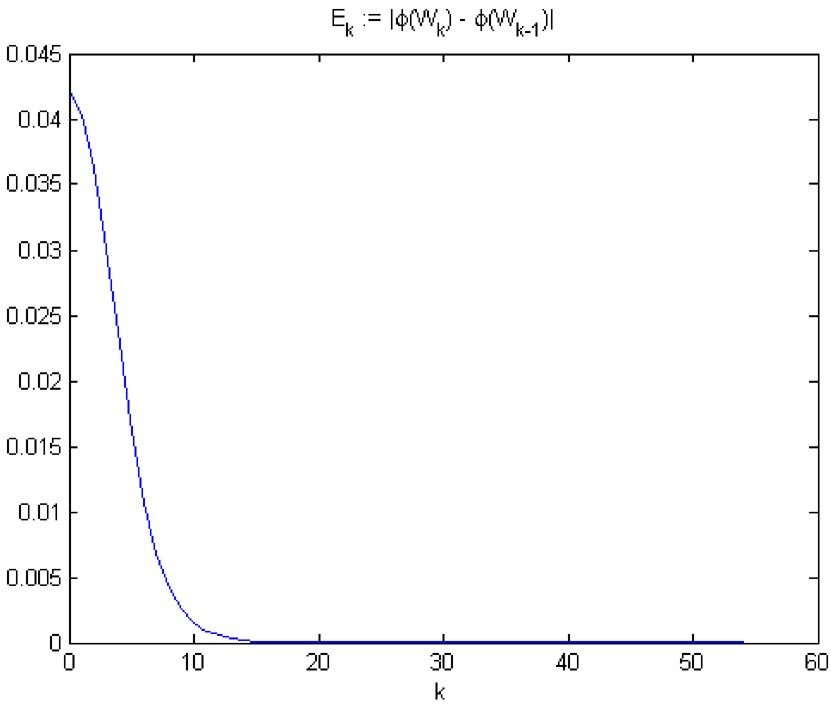

Figure 5: Plot showing the error for $\alpha = 0.1$

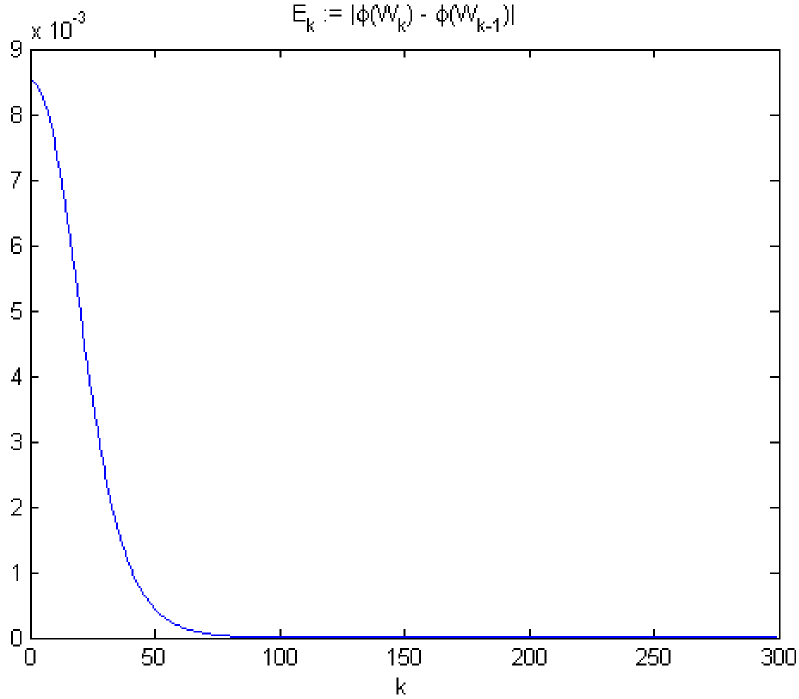

Figure 6: Plot showing the error for $\alpha = 0.02$

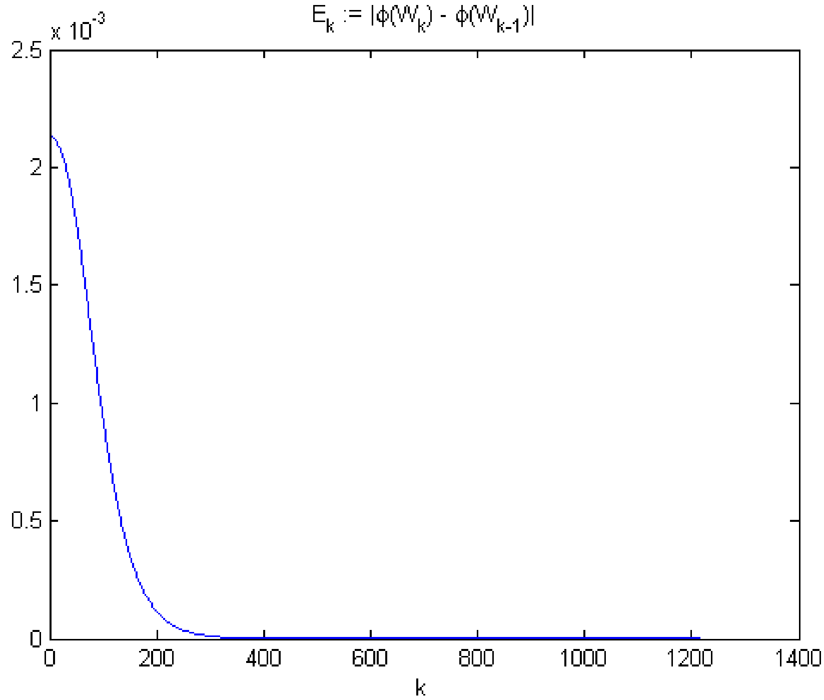

Figure 7: Plot showing the error for $\alpha = 0.005$

1c)

The Matlab script imggen.m takes 3 grayscale pictures with positive kurtosis and turns them into Matlab vectors containing information about the degree of black/white in each pixel of the picture. These 3 pictures can then be mixed in the same way that the signals from 1b were mixed, and again demixed by solving the differential equation system with Eulers Method. fig. 8 shows three grayscale pictures on the first line, the superposition mix on the second from a random mixing matrix, and the estimated signals after de-mixing on the last line. Just as the sign was not always correct on the speaker signals, wrong sign results in an inverted image, as one can see on the picture on the bottom, left. The images on the bottom row were produced with the forward Euler method using the Matlab function makeimg.m which takes the transformed signals from imggen.m as the argument.

Figure 8: The pictures

The whole point of the differential equation system was to solve the opti-

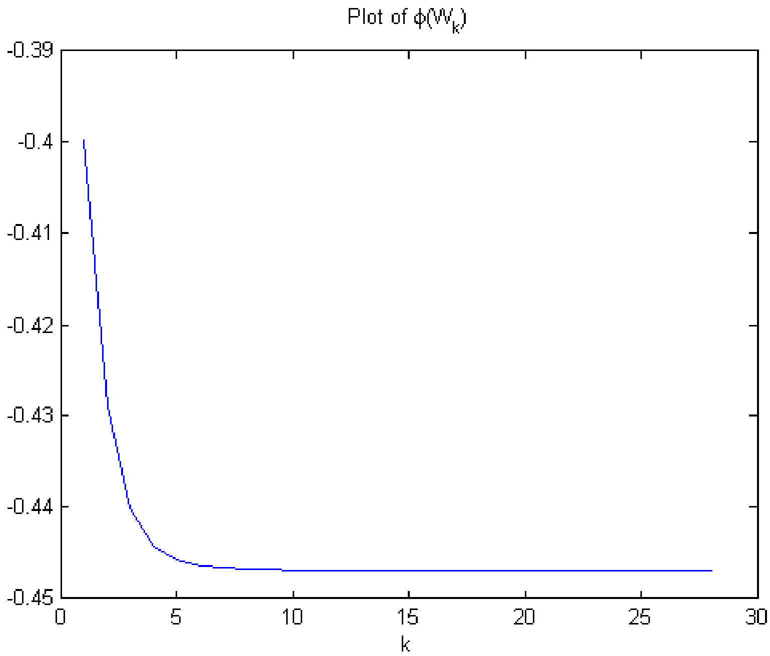

Figure 9: Progression of $\phi(W_k)$

mization problem $\max_{(W \in O)} \phi(W)$. The definition of ϕ being

$$\phi(W) = \pm \frac{1}{4} \sum_{i=1}^{n} kurt((w)_i^T \tilde{x}) \tag{18}$$

Kurtosis is a measure of peakedness for a given distribution. Since we are trying to find solutions further from the Gaussian distribution, kurtosis can be used to decide how close our solution is to the Gaussian distribution, which is what $\phi(W)$ does. Depending on the sign in front of the sum, the problem becomes a maximum or minimum problem. fig. 9 shows the progression of $\phi(W_k)$ for the images, and because of ϕs connection with kurtosis, it illustrates the optimization of the kurtosis.

Problem 2

a)

We want to construct a Lagrange interpolation polynomial p_1 of degree $n = 1$, for a continuous function f defined on the interval [-1,1] using interpolation points $x_0 = -1$ and $x_1 = 1$. The Lagrange interpolation polynomial is given by the equation

$$p_n = \sum_{k=0}^{n} L_k(x) f(x_k). \tag{19}$$

Where L_k is defined as

$$L_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}. \tag{20}$$

For the given interpolation points $L_k(x)$ yields the following

$$\begin{aligned} L_0(x) &= \prod_{i=0, i\neq 0}^{1} \frac{x - x_1}{x_0 - x_1} = \frac{x-1}{-1-1} = \frac{1}{2}(1-x) \\ L_1(x) &= \prod_{i=0, i\neq 1}^{1} \frac{x - x_0}{x_1 - x_0} = \frac{x-(-1)}{1-(-1)} = \frac{1}{2}(x+1). \end{aligned} \tag{21}$$

The interpolation polynomial is thereby given as

$$\begin{aligned} p_1 &= \sum_{k=0}^{1} L_k(x) f(x_k) = L_0 f(x_0) + L_1 f(x_1) \\ &= \frac{1}{2}(1-x) f(x_0) + \frac{1}{2}(x+1) f(x_1). \end{aligned} \tag{22}$$

Assuming that the second derivative of f exists and is continous on [-1,1], we will show that

$$\mid f(x) - p_1(x) \mid \leq \frac{M_2}{2}(1 - x^2) \leq \frac{M_2}{2} \text{ , } x \in [-1, 1], \tag{23}$$

where $M_2 = \max_{x\in[-1,1]} \mid f^{''}(x) \mid$. We can write out the following equation[3]

$$\begin{aligned} &\mid f(x) - p_n(x) \mid \leq \frac{M_{n+1}}{(n+1)!} \mid (x - x_0) \ldots (x - x_n) \mid \\ &\mid f(x) - p_1(x) \mid \leq \frac{M_2}{2} \mid (x+1)(x-1) \mid = \frac{M_2}{2} \mid (x^2 - 1) \mid = \frac{M_2}{2}(1 - x^2). \end{aligned} \tag{24}$$

For the given interval, $[-1, 1]$, we can write $\mid (x^2 - 1) \mid = (1 - x^2)$. And we thereby get

$$| f(x) - p_1(x) | \leq \frac{M_2}{2}(1-x^2) = \frac{M_2}{2} - \frac{M_2}{2}x^2 \leq \frac{M_2}{2}. \tag{25}$$

Lets now make an example of a function f and a point $\tilde{x}$ where the following equality is achieved

$$| f(\tilde{x}) - p_1(\tilde{x}) | = \frac{M_2}{2}. \tag{26}$$

If we write it further out we get the expression

$$| f(\tilde{x}) - p_1(\tilde{x}) | = | f(\tilde{x}) - (1-\tilde{x})f(-1) - (1+\tilde{x})f(1) | = \frac{M_2}{2} \tag{27}$$

We see from equation (23) that $\tilde{x}$ needs to correspond to the value of x which maximizes the left side of the equation, which is when $x = 0$. We thereby choose $\tilde{x} = 0$, and we can then write

$$| f(0) - p_1(0) | = | f(0) - \frac{1}{2}(1-0)f(-1) - \frac{1}{2}(1+0)f(1) | = | f(0) - \frac{f(-1) - f(1)}{2} | \tag{28}$$

If we now let function f be a symmetric function about the y-axis and let $f(-1) = f(1) = 0$, we get

$$| f(0) | = \frac{M_2}{2} \tag{29}$$

We have made som criterias that function f needs to meet:
1) $f(0)$ has to be the global maximum/minimum on the interval $[-1, 1]$ so the inequality (23) holds.
2) f is symmetric about the y-axis.
3) $f(-1) = f(1) = 0 \Longrightarrow p_1(x) = 0$ for simplicity
4) $| f(0) | = \frac{M_2}{2}$

A possible solution to the equality and the given criterias can be given by a second degree polynomial of the form

$$f(x) = Cx^2 - C \Longrightarrow f^{''}(x) = 2C \Longrightarrow M_2 = |2C| \tag{30}$$

Where C is a constant. From the equation we see that all the criterias are met and the given inequality (23) holds for the interval [-1,1].

$$
\begin{aligned}
&| f(x) - p_1(x) | \leq \frac{M_2}{2}(1-x^2) \leq \frac{M_2}{2}\;,\; x \in [-1,1] \\
&| Cx^2 - C - 0 | \leq \frac{|2C|}{2}(1-x^2) \leq |C| \\
&| x^2 - 1 | \leq (1-x^2) \leq 1. \\
&(1-x^2) \leq (1-x^2) \leq 1.
\end{aligned} \tag{31}
$$

References

[1] Nicholas J. Higham *Newton's Method for the Matrix Square Root* MATHEMATICS COMPUTATION VOLUME 46, NUMBER 174, APRIL 1986 Pages 537-549

[2] A Hyvarinen and E. Oja *Independent component analysis: algorithms and applications.*

[3] Theorem 6.2, p. 183, *An Introduction to Numerical Analysis, Endre Suli and David Mayers.*

Matlab code

Script to run the ICA method on signals and images

generate.m

```
clear;
clc;
close all;
%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
%Speaker signals for 1b
xx=gen();
W=makeplots(xx);


%%%%%%%%%%%%%%%%%%%%
%Images for 1c
xx=imggen();
W=makeimg(xx);
```

Speaker Signals

gen.m

```
function [xx  x]= gen()
%Generating the sources (Doubles indicate tilde, f.eks. xx=x.tilde)
    tt=[0:0.1:100];
    vv=sin(1+tt*2/pi);
    w1=[-1:0.01:1-0.01];
    ww=[w1 w1 w1 w1 w1 1];
    uu=cos(1+tt.^2*2/pi);
    SS=[vv;ww;uu];

%Transform sources to mean=0
    myv=mean(vv);
    myu=mean(uu);
    myw=mean(ww);
    v=vv-myv;
    v=v/std(v);
```

```
    w=ww-myw;
    w=w/std(w);
    u=uu-myu;
    u=u/std(u);
    S=[v;w;u];
%Covariance matrices
    M=length(tt);
    ES=(S*S')./M;
%Plot the original sources transformed for reference
    subplot(3,1,1);
    plot(0:1000,S(1,:));
    title('Transformed source signals, \mu=0, \sigma^2=1');
    subplot(3,1,2);
    plot(0:1000,S(2,:));
    subplot(3,1,3);
    plot(0:1000,S(3,:));

% Mixing matrix
    A=[1 2 1 ; 3 4 5; 4 1 2];
    x=A*S;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    EX=(x*x')./length(x);
%Transform x so the covariance matrix E[xx*xx']=I, using the inverse square
%root.
    Q=eigroot(EX);
    xx=Q\x;
    figure;
    subplot(3,1,1);
    plot(0:1000,xx(1,:));
    title('Transformed microphone signals');
    subplot(3,1,2);
    plot(0:1000,xx(2,:));
    subplot(3,1,3);
    plot(0:1000,xx(3,:));
end
```

makeplots.m

```
function W=makeplots(xx)
%Creating I as the starting point for the Euler iterations.
    I=eye(3);
%Creates vectors that decides stepsize and if fuler uses the QR projection
```

```
    steps=[0.1,0.02,0.005];
    projectv=[true,true,true];
%As the derivative of gamma approaches 0, the max(phi)
%optimization problem will approach it's solution so we'll use the
%derivative as a measure of how many times to run the euler methods.
    for k=1:3
%Reset difference for each run
        clear difference;
        gammad=-Inf;
        i=0;
        W=I;
        while (gammad<-1e-12 && i < 10000)
            i=i+1;
            temp=feuler(W,xx,projectv(k),steps(k));
            difference(i)=err(temp,W,xx);
            W=temp;
            gammad=derivegamma(W,xx);
        end
        disp(W);
%Plot the Error agains the number of iterations
%figure;
%plot(0:i-1,difference);
%title('E_k := |\phi(W_k) - \phi(W_{k-1})|')
%xlabel('k')
        fprintf('%s%g, %s%i, %s%i\n','Forward Euler, step=',steps(k),...
            'Projection=',projectv(k),'i=' ,i)
        y=W'*xx;
        figure;
        subplot(3,1,1);
        plot(0:1000,y(1,:));
        title(['Estimated Source Forward Euler Method ',num2str(k)]);
        subplot(3,1,2);
        plot(0:1000,y(2,:));
        subplot(3,1,3);
        plot(0:1000,y(3,:));
    end
%UNCOMMENT FOR BACKWARD EULER METHOD

    for k=1:3
        clear difference;
        gammad=-Inf;
        i=1;
        W=I;
        difference(i)=Inf;
```

```
        while (gammad<-1e-12 && i < 10000)
            i=i+1;
            temp=feuler(W,xx,projectv(k),steps(k));
            difference(i)=err(temp,W,xx);
            W=temp;
            gammad=derivegamma(W,xx);
        end
        disp(W);
        fprintf('%s%g, %s%i\n','Backward Euler, step=',steps(k),'i=' ,i)
        y=W'*xx;

%%%ERROR PLOT
%figure;
%plot(0:i-1,difference);
%title('E_k := |\phi(W_k) - \phi(W_{k-1})|')
%xlabel('k');
        figure;
        subplot(3,1,1);
        plot(0:1000,y(1,:));
        title(['Estimated Source Backward Euler Method ',num2str(k+2)]);
        subplot(3,1,2);
        plot(0:1000,y(2,:));
        subplot(3,1,3);
        plot(0:1000,y(3,:));
    end
end
```

Image generators

imggen.m

```
function xx = imggen()
%Generating the sources
%Turn the images into matlab vectors

    spock = double(imread('spock2.jpg'));
    vader = double(imread('vader.jpg'));
    eric = double(imread('eric.jpg'));

    spock=spock(:)';
    vader=vader(:)';
    eric=eric(:)';

```

```
%Transform sources to mean=0 and unit variance;
    spockm=mean(spock);
    benderm=mean(eric);
    r2d2m=mean(vader);
    spock=spock-spockm;
    spock=spock/std(spock);
    eric=eric-benderm;
    eric=eric/std(eric);
    vader=vader-r2d2m;
    vader=vader/std(vader);
    S=[spock;eric;vader];
    figure;
    colormap gray;
    subplot(3,3,1);
    imagesc(reshape(spock,333,500));

    subplot(3,3,2);
    imagesc(reshape(eric,333,500));
    subplot(3,3,3);
    imagesc(reshape(vader,333,500));


% Mixing matrix
%Signal matrix
    %A=[1 2 1 ; 3 4 5; 4 1 2];
%Semi-random matrix
    A= ceil(7*rand(3));
    x=A*S;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    EX=(x*x')./length(x);
%Transform x so the covariance matrix E[xx*xx']=I, using the inverse square
%root.

    Q=eigroot(EX);
    xx=Q\x;

    subplot(3,3,4);
    imagesc(reshape(xx(1,:),333,500));
    subplot(3,3,5);
    imagesc(reshape(xx(2,:),333,500));
    subplot(3,3,6);
```

```
    imagesc(reshape(xx(3,:),333,500));
end
```

makeimg.m

```
function W=makeimg(xx)
%Creating I as the starting point for the Euler iterations.
    I=eye(3);

    steps=[0.2];
    projectv=[true];
%As the derivative of gamma approaches 0, the max(phi)
%optimization problem will approach it's solution so we'll use the
%derivative as a measure of how many times to run the euler method.
    for k=1
        gammad=-Inf;
        i=0;
        W=I;
        phid=zeros(1,1000);
        while (gammad<-1e-8 && i < 1000)
            i=i+1;
            temp=feuler(W,xx,projectv(k),steps(k));
            phid(i)=phif(temp,xx);
            W=temp;
            gammad=derivegamma(W,xx);
        end
        disp(W);
        fprintf('%s%g, %s%i, %s%i\n','Forward Euler, step=',steps(k),...
            'Projection=',projectv(k),'i=' ,i)
        y=W'*xx;
        subplot(3,3,7);
        imagesc(reshape(y(1,:),333,500));
        subplot(3,3,8);
        imagesc(reshape(y(2,:),333,500));
        subplot(3,3,9);
        imagesc(reshape(y(3,:),333,500));
    end

%Uncomment to run the images through backward euler.

% for k=2
% gammad=-Inf;
% i=0;
```

```
% W=I;
% test=Inf;
% while (gammad< -eps)
%     W=beuler(W,xx,steps(k+1),10);
%     i=i+1;
%     temp=gammad;
%     gammad=derivegamma(W,xx)
%     test=abs(gammad-temp);
% end
% disp(W);
% fprintf('%s%g, %s%i\n','Backward Euler, step=',steps(k+1),'i=' ,i)
% figure;
% y=W'*xx;
% hold on;
% plot(y(1,1:100));
% plot(y(2,1:100),'g');
% plot(y(3,1:100),'r');
% title(['estimated source beuler ',num2str(k)]);
% end
```

Convenience functions

feuler.m

```
%Takes a single step in the forward Euler Method

function forward = feuler(W,xx,project,step)
    forward=W-step*(gradphi(W,xx)-W*((gradphi(W,xx))')*W);
    if project
        [Q,R]=qr(forward);
        forward=Q;
    end
end
```

beuler.m

```
%Calculates a single step in the backwards Euler method, using fixed point
%iteration and the maxnorm as a measure of when appropriate accuracy is
%achieved

function [backwardt, i]=beuler(W,xx,step,iterations)
    backwardt=W;
```

```
    i=0;
    maxnorm=Inf;
    while (i<iterations && maxnorm> 1e-10)
        [Q,R]=qr(backwardt);
        backward=Q;
        temp=backwardt;
        backwardt=W-step*(gradphi(backward,xx)-backward*...
            ((gradphi(backward,xx))')*backward);
        i=i+1;
        maxnorm=max(max(abs(temp-backwardt)));
    end
end
```

eigroot.m

```
%Finds the root of matrix X

function rooted=eigroot(X)
    [V,D] = eig(X);
    rooted = V*(sqrt(D))*V';
end
```

derivegamma.m

```
%Calculates the derivative of gamma PS: I know that it's called
%differentiation and not derivation
function gammad = derivegamma(W,xx)
    gammad=trace((gradphi(W,xx))'*(-gradW(W,xx)));
end
```

gradphi.m

```
%Calculates the gradient of phi

function Ephi = gradphi(W,xx)
    y=W'*xx;
    Ephi=(xx*(y.^3)')/length(xx);
end
```

gradW.m

```
%calculates the gradient of W
function grads = gradW(W,xx)
    grads=(gradphi(W,xx)*(W')-W*(gradphi(W,xx))')*W;
end
```

err.m

```
%Calculates the difference in the non-Gaussianity measure as a measure of
%error.
function difference=err(Wk, Wkneg1,xx)
    difference=abs(phif(Wk,xx)-phif(Wkneg1,xx));
end
```

phif.m

```
%Calculates phi, a suitable measure for non-Gaussianity

function phi = phif(W,xx)
    phi=0;
    n=length(W);
    for i = 1:n
        phi=phi+mean((((W(:,i))')*xx).^4);
    end
    phi=phi-3*n;
    phi=0.25*phi;
end
```