IN5450/9450
Mandatory Exercise 2

Andreas Austeng and Sven Peter Näsholm

April, 2020

Distribution date	April 2, 2020
Deadline	May 4, 2020
Send report to	svenpn@ifi.uio.no
Version	1.0, April 2, 2020

Part A: High Resolution Beamforming on farfield monochromatic signals

- This mandatory exercise is based on the examples shown in figures 4b and 5 in the paper: H. Krim, M. Viberg, *Two decades of array signal processing research – The parametric approach*, IEEE Signal Processing Magazine, pp. 67–94, July 1996. You may download the document from here: http://dx.doi.org/10.1109/79.526899 (You need to be on the University of Oslo network to get access.)
- Problem 1–7 are mandatory for all students.
- Problems 8 is mandatory for INF9450 students, but voluntary for other students. .
- The subject is on estimation of the spatial spectrum for an $M = 10$ element uniform linear array with half-wavelength spacing. The input consists of two incoherent signals at 0 and -10 degrees in additive spatially white noise. The signal to noise ratio for both sources is 0 dB, and $N = 100$ samples are available of the input. The signal and noise model is described on page 73 of the paper.
- The MATLAB code for the generation of the input is found at: http://folk.uio.no/inf5410/2005V/proj2b.m

1. Estimate the spatial correlation matrix. Estimate and plot the spatial correlation.
2. Estimate the spatial spectrum using the conventional method (Figure 4b). Plot the response both in linear and dB scale. Discuss why the sources are not separated.
3. Estimate the spatial spectrum for the same signal using the minimum variance beamformer (Capon's beamformer) (Figure 5). Plot the response both in linear and dB scale. Discuss the differences from the conventional beamformer.
4. Plot the distribution of the eigenvalues of the correlation matrix and explain it on the basis of the signal and noise model.
5. Estimate the spectrum using the MUSIC algorithm (Figure 5) assuming that the number of signals is known. Plot the response both in linear and dB scale. Discuss the differences from the previous estimates.

6. Estimate the spatial spectrum by the eigenvector method (see the lecture notes for definition). Plot the response both in linear and dB scale. Discuss the differences from the MUSIC beamformer.

7. *Incorrect estimate of the number of sources*
   Estimate the spatial spectrum with the MUSIC method (and eigenvector method) when the number of signals is incorrectly estimated. Let the estimate of the number of signals be $0, 1,$ and 3. Discuss the differences between the estimates for the various cases. (Which spatial spectrum estimator is the eigenvector method equivalent to when 0 signals are assumed to be present?)

8. *Coherent sources*
   Modify the signal generator so that it generates coherent signals instead. Find the properties of the previous beamformers for coherent signals (You may have to change the angles of incidence also). Implement the various forms of averaging of the correlation estimate and see if this gives the methods a better ability to handle coherence.

Part B: High Resolution Beamforming on broadband ultrasound signals

In this exercise you shall implement the Delay-And-Sum (DAS) beamformer for a uniform linear ultrasound transducer array. Available is a simulation of a 36-element transducer array. The transmitter is a single element, unfocused transducer located in position $(x,y,z) = (0,0,0)$. We are imaging several distinct points at 20, 60, 65 and 100 mm. Noise is added to the simulated data. The simulation is done with the MATLAB toolbox `FieldII` (see http://www.field-ii.dk). The MATLAB file that defines the simulation is `CreateArrayData.m`. The output of the simulation is stored in the file `in5040_obligII_arraydata_points.mat`. It contains the following variables:

```
>> P = load('in5040_obligII_arraydata_points.mat')

P =

       image_data: [2416x36 single]
               f0: 2500000
               fs: 15000000
                c: 1500
   element_pitch: 3.0000e-04
            kerf: 7.5000e-05
        nElements: 36
            tAxis: [1x2416 double]
      elementsRx: [3x36 double]
```

That means:
A `P.nElements` array with inter-element spacing `P.element_pitch [m]` and center frequency `P.f0 [Hz]` has been simulated. The element positions are given in `P.elementsRx [m]`. The data is sampled at sampling frequency `P.fs [Hz]`. The speed-of-sound is set to `P.c [m/s]`. The resulting raw data is given in the matrix `P.image_data`, where the first index indicates time and the sample times are given in `P.tAxis [s]`, and the second index indicates the element number. Figure 1 shows the image obtain after processing the data.

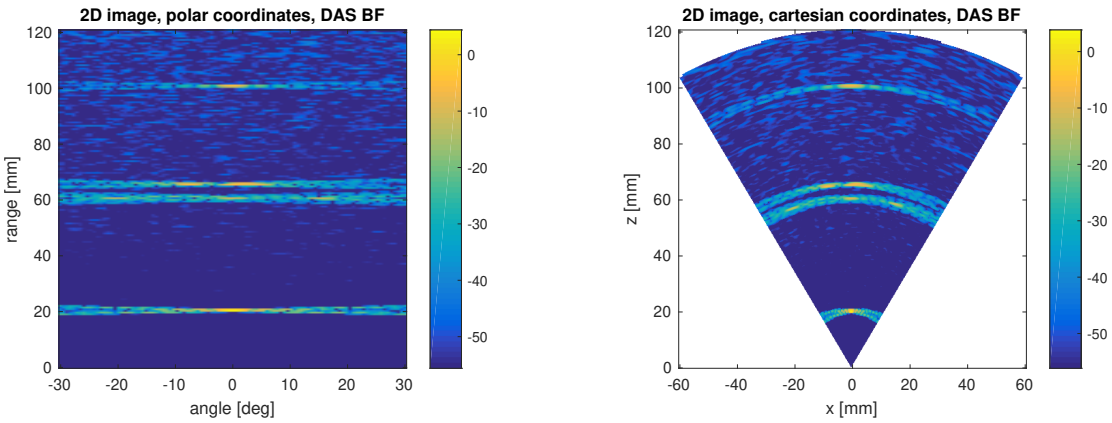

Figure 1: Beamformed images, Delay-And-Sum (DAS) beamformer (BF). The left image is shown in polar coordinates. The right image is scanconverted and shown in cartesian coordinates.

1. The following code loads the data and sets up some possible coordinates to beamform the data into.

```
%%
fname = 'in5040_obligII_arraydata_points';
P = load(fname);

% In the end we want to find the envelope of the data. This can be achieved
% by taking the Hilbert transform of the data. We choose to do that on the
% raw data. For MVDR, complex data is needed to distinguish signals coming
% from \theta and -\theta.

dataH = hilbert(P.image_data);  % hilbert is taken along dimension '1', i.e. time

%%
% Define angles (relative to z-axis):
nAngles = 37;
uMax = sind(30);
P.angles = asin(linspace(-uMax,uMax,nAngles));

%%
% Define points to beamform: A 2D grid defined by the polar coordinates
% P.angles and P.zPoints
%
MaxRange = P.tAxis(end)*P.c / 2;  % max range of data
lambda = P.c/P.f0;
dR = lambda;  % Samplinginterval along range
P.rPoints = (0:dR:MaxRange)';
```

   - `P.angles` defines 37 angles. Why is this a reasonable/unreasonable number to use for making an image as the one in Figure 1?
   - Apply the conventional method from Part A-2 using phases to beamform the data into the given angles and ranges. Observe that the data `dataH` is not stationary. This means that you can only use one (or a few) snapshots (time samples) for estimating the covariance matrix. The covariance matrix must be estimated for each depth you would like to image. Plot the resulting image. Discuss the result.

2. Apply the minimum variance beamformer from Part A-3 to the same data. You will need to estimate a new invertible covariance matrix $\mathbf{R}$ for each depth you would like to image. To make the covariance matrix invertible, you need to apply robustification. A diagonal loading factor of $\varepsilon \cdot \text{trace}(\mathbf{R})/L$, where ε is between 1/100 and 5/100 and L is the sub-array length are typical values used. Forward-backward averaging is also possible to apply. If processing time becomes an issue, you can limit the ranges to calculate the image to 55 – 70 mm. Plot the resulting image. Discuss the result.

3. Write a function that for a given set of angles and reconstruction ranges calculates the delays from each transducer element to each reconstruction point, and then back again to the transducer, and thereafter delays the raw data such that the data for each receive element is focused

onto these points. The function should transform a set of data of the form `time-samples × receive elements` to the form `ranges × angles × receive elements`. This delayed-but-not-summed data cube should then be returned from the function. A possible outline of the function could be:

```
function dataDelayed = delayData(rawData, tAxis, c, elPos, angles, rs)
%
% Function that delays a set of raw data from every point defined by the
% polar coordinates 'angels' and 'rs' to every receiver in 'ElPos'
%
%
%   Input:
%           rawData : [#ranges x #els] (real or complex)
%           tAxis   : [#ranges x 1] range-axis of rawData [sec]
%           c       : speed-of-sound  [m/s]
%           elPos   : [3 x #els]  (x,y,z) for every element in [m]
%           angles  : set of directions to delay data [radians]
%           rs      : set of reconstruction ranges [m]
%
%   Output:
%       delayedData : [#rs x #angles x #els]
%

% get hold of dimentions
nEls = size(elPos,2);
nAngles = length(angles);
nRs = length(rs);

% Allocating memory
dataDelayed = zeros(nRs,nAngles,nEls,'like',rawData(1));
delays = zeros(nRs,nEls);

% calculate delays
for ii=1:nAngles  % loop over anges

    for jj=1:nRs  % loop over ranges, calculate delay('range','elements')
        delays(jj,:) = ( <"distance from center of array to point rs(jj)"> + ...  % transmit ↩
             distance
            <"distance from all elements to point rs(jj)"> ... % receive distance
            )/c; % converting distance to time.
    end

    % loop over elements
    for kk=1:nEls
        % Pick out the correct data for all delays. At the same time
        % perform time-gain-compensation.
        dataDelayed(:,ii,kk) = single(interp1(tAxis,double(c*(tAxis.*rawData(:,kk))),delays(:,kk)));
    end

end
```

- Implement DAS on the delayed data. Plot the resulting image. Discuss the result.
- How and why does this image differ from the previous images?

4. Implement the minimum variance beamformer on the delayed-but-not-summed data cube. Note that you now have to estimate the covariance matrix for each angle and range, but you only need one estimated image value from each covariance matrix. Since the delay part has been applied to the data, the appropriate steering angle for all points will be broadside. You will need to apply robustification when estimating the covariance matrix. Plot the resulting images. Discuss the result.

MATLAB code

At https://www.uio.no/studier/emner/matnat/ifi/IN5450/v18/undervisningsmateriell/obligII/ you will find the following MATLAB functions/data:

CreateArrayData.m The function used to create the array data.

getScanConvertedImage.m A function that scanconverts a polar image into a cartesian one.

in5040_obligII_arraydata_points.mat A mat-file with the needed data used in the array processing assignment.

The presentation

Please provide an electronic presentation containing a short description of the problem, all necessary derivations, results, m-code, and discussions. In addition to the presentation, please also provide your MATLAB source code and a run-time example of you code (in MATLAB, use 'diary').

Your presentation should not exceed 15-20 slides. The time for you to present it should be $\sim$15 minutes.

If you prefer to write some of the presentation or make some illustrations by hand, it is okay. Please then scan the handwritten pages and include these in your presentation.

Deliver your files in time to svenpn@ifi.uio.no as a zip archive named
oblig_1_LASTNAME.zip, where you replace LASTNAME with your last name.