

Antenna synthesis using the Orchard-Elliotts method *

Johan Skatt, e93_jsk@e.kth.se

Stefan Petersen, e92_spe@e.kth.se

Daniel Ringström, e92_dri@e.kth.se

1997-05-20

*Typeset in L^AT_EX

Abstract

The given task was to, with the use of the Orchard-Elliott method, synthesis a shaped beam antenna pattern. Here it will be shown how this can be done using MATLAB and a little C.

There is also some simplifications introduced to the OE-method that speeds up calculations, these simplifications gave as a bonus the ability to use a non-derivable function as the shaping function. Some alternative ideas on the optimization of excitations are also presented.

1 Objective

The objective of this project was to synthesis a shaped-beam antenna pattern according to the specifications in section 2.

There were also specified that the Orchard-Elliott [1] method should be used, which is described in section 3.

2 Specification

The following data were given:

- 41 isotrope elements
- Distance between each element: $d = \lambda/2$
- The level of the sidelobes should be -20 dB for $0^\circ < \theta^1 < 110^\circ$
- Mainlobe should be shaped as \csc^2 for $110^\circ < \theta^2 < 150^\circ$, ripple $< \pm 0.5$ dB
- The level of the sidelobes should be -30 dB for $150^\circ < \theta < 180^\circ$

3 Orchard-Elliotts (OE) method

The idea of OE is that with the technique of placing roots, on or about the unit circle, the desired pattern is to be approximated. In the shaped region (region I) the roots are displaced from the unit circle and in region II, see figure 1, the roots are placed on the unit circle to suppress sidelobes.

Starting with the arrayfactor as:

$$F = \sum_{n=0}^N I_n e^{jnkd \cos \theta} \quad (1)$$

(1) can then, with the substitutions $\psi = kd \cos \theta$ and $w = e^{j\psi}$, be written:

$$\begin{aligned} F &= \sum_{n=0}^N I_n w^n = \{\text{Factor the polynom}\} = \\ &= I_N \prod_{n=1}^N (w - w_n) = \{w = e^{a_n + jb_n}\} = \prod_{n=1}^N (w - e^{a_n + jb_n}) \end{aligned} \quad (2)$$

¹ θ measured from endfire

²The true limits is $20^\circ < \theta < 60^\circ$ because in the specification θ is measured with broadside as reference

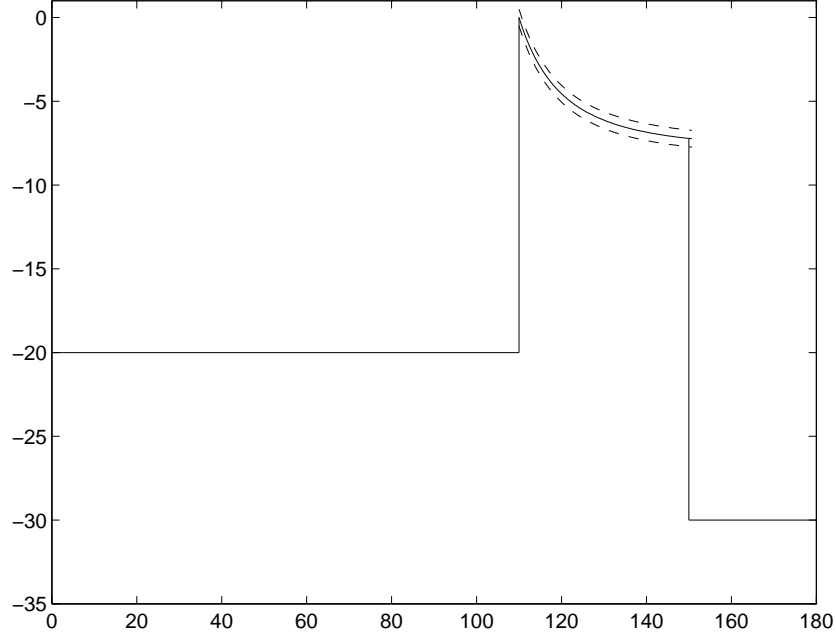


Figure 1: Specified antenna pattern

According to this the arrayfactor may be described using a polynomial with N roots. Letting the root w_N be fixed at -1 , e.g. $a_N = 0$ and $b_N = \pi$. This allows us to write F , after taking the absolute value and logarithmizing it, as:

$$G = \lg |F|^2 = \sum_{n=1}^{N-1} 10 \lg[1 - 2e^{a_n} \cos(\psi - b_n) + e^{2a_n}] + \quad (3)$$

$$+ 10 \lg[2(1 + \cos \psi)] + C_1$$

Were the constant C_1 has been added to allow G to float up and down, so the maximum level may be set to 0 dB.

Now let N_1 represent the number of roots in region I and N_2 the ones placed in region II. This gives us the total amount of roots as $N = N_1 + N_2 + 1$.

The parameters available for placing the roots are a_n och b_n . Let $a_n = 0$ in region II and $a_n \neq 0$ in region I, the sign of a_n does not interfere with the form of the beam because e^{a_n} or e^{-a_n} is related as mirroring in the unit circle. This leaves us with $2N_1$ eligible parameters in region I and N_2 parameters in region II, also C_1 may be choosen freely. The total amount of eligible parameters may therefore be written as $N_3 = 2N_1 + N_2 + 1$. These N_3 parameters are assumed assembled into a vector \mathbf{x} in following order. The first N_1 positions

contains the a_n :s in region I next N_2 positions contains the b_n :s in region II followed by the N_1 b_n :s in region I, the C_1 constant is placed in the last position of the vector.

In region I the use of a shaping function is necessary to specify the desired shape of the beam. In this case \csc^2 . $S(\psi)$ then gives the ideal behavior of G in region I. Unfortunately it is not possible for G to behave exact as S (with the use of a infinitely amount of roots it would be possible). It is easily shown that the best approximation is to let G oscillate around S , compare to the Fourier series [2].

In region II the only specification that has to be made is the maximum allowed value of G , which of course will be set to the desired one. This means that there is no reason to comprimize between the desired shape of the pattern and the one that is available using G .

3.1 The algorithm in practice

Now how should this be accomplished?

Well if done by hand you would be forced to choose a good start guess for a_n and b_n to avoid large and complicated calculations. Usage of a computer makes the start guess less interesting. The computer makes it possible to choose a simpel start guess and then iterate towards desired antenna pattern.

First we have to specify the values for the maximas of G in region II and the maximas and minimas of $G - S$ in region II. They will of course be of same number as the roots, N_3 . Let the specified values be placed into a vector \mathbf{g} . This vector should for convenience have its elements placed from left to right according to figure 1. This means that the values corresponding to the roots that are stationed on the unit circle are placed into the first N_2 positions next, in the following $2N_1 + 1$ positions the maximas and minimas for the beam G minus the shaping function S are placed in alternating order. This procedure makes the last position contain the value of $G - S$ at the peak of the shaped region, which may be used to calculate C_2 .

Now we have a nice specification, so lets choose some neat starting values for a_n and b_n . The starting guess may be choosen as:

$$b_n = \left(\frac{2n}{N+1} - 1 \right) \pi, \quad n = 1, 2, \dots, N-1 \quad (4)$$

$$a_n = 0, \quad n = 1, 2, \dots, N_2 \quad (5)$$

$$a_n = 0.02, \quad n = N_2 + 1, \dots, N-1 \quad (6)$$

Now we have to locate the maximas/minimas of G that are generated by the starting guess. To do this in a efficient way let us use Newtons algorithm. This algorithm converges rapidly, in fact quadratic. The only problem is that it solves for the roots of the function using its derivate. This forces us to calculate both the first and second order derivates of G . In [1] Orchard and Elliott suggestes that in region I the maximas and minimas of $G - S$ should be found, this makes it necessary to calculate the first and second derivate of S . Now notice that if S don't vary to swift, then G will have its maximas/minimas exactly at the same angles, ψ :s, as $G - S$. In this case S is \csc^2 which is a smooth and nice function. Therfore there is no reason for us to calculate the derivate of S , it will do just fine with the first and second derivate of G to locate the maximas/minimas. This saves a **lot** of calculation. Another advantage is that it's possible to use a function S that has a non continues derivate. This is shown in figure 2, were S has been taken as a triangle. The MATLAB-code for the Newton algorithm may be found in appendix A.1.

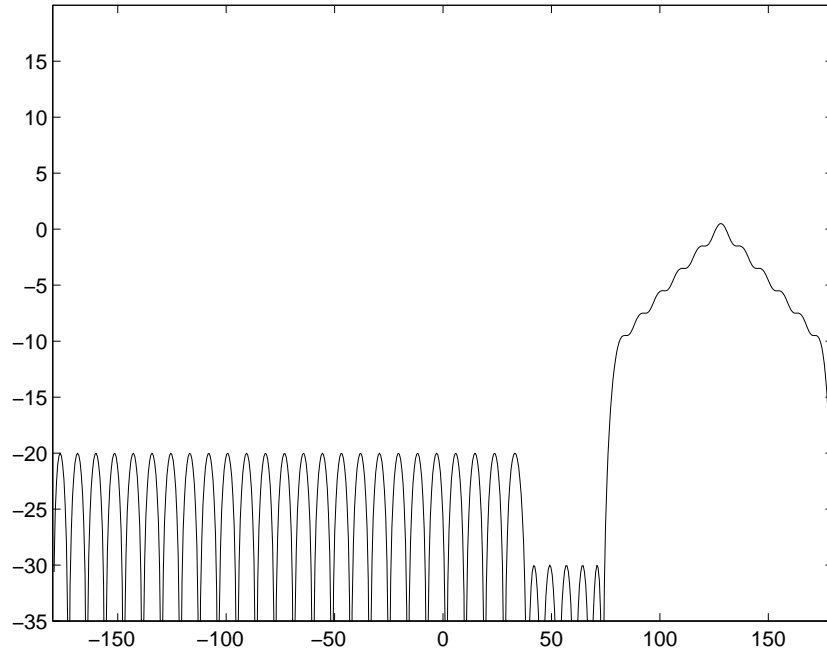


Figure 2: The beamform created using a triangle as the shaping function

The angles , ψ_i , $i = 1, 2, \dots, N - 1$, that are returned from the Newton algorithm would now be used to calculate a better approximation by creating a new vector called $\hat{\mathbf{g}}$. This is done as follows. First the rotation angle ψ_r has

to be found. The ψ_r is needed to compute S over the specified area, region I. The ψ_0 and θ_0 are related as $\psi_0 - \psi_r = kd \cos \theta_0$, where ψ_0 and θ_0 are the angles at the peak of the beam. When S is supposed to be calculated for $110^\circ < \theta^3 < 150^\circ$. This allows us to express S in ψ as:

$$S(\theta) = \csc^2 \theta = \csc^2(\arccos(\psi_0 - \psi_r)) \quad (7)$$

Now when we know both S and G we are able to take the maximum values of G and the maximums/minimas of $G - S$ and assemble them together in the vector $\hat{\mathbf{g}}$.

The simplest way to create a better approximation to \mathbf{x} is to use the linear part of the Taylor's series for $\hat{\mathbf{g}}$, which is given as:

$$\mathbf{g} = \hat{\mathbf{g}} + \mathbf{A}\Delta\mathbf{x} \quad (8)$$

Where the matrix \mathbf{A} is the Jacobian matrix, (see appendix A.3) whose components $a_{i,j}$ are defined by:

$$a_{i,j} = \frac{\partial G(\psi_i, \mathbf{x})}{\partial x_j}, \quad i, j = 1, 2, \dots, N_3 \quad (9)$$

Solving equation (8) for $\Delta\mathbf{x}$ gives us $\Delta\mathbf{x} + \mathbf{x}$, which may be used as a better approximate vector. This completes the iteration. Now by looping this a number of times the difference between \mathbf{g} and $\hat{\mathbf{g}}$ approaches zero. In our case we had to use 5 iterations to get the vector $\hat{\mathbf{g}}$ to look like \mathbf{g} , this must be considered as a very rapid convergence.

3.2 Results

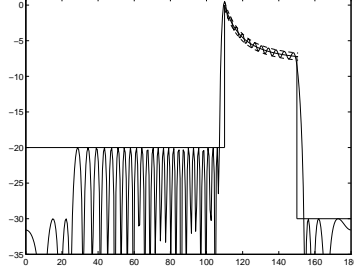
After having completed 5 iterations we had a beamform as in figure 3(a). We now applied the algorithm described in section 4, this gave the result shown in figure 3(b) which if compared to the unoptimized pattern is identical. The only difference between the two patterns are the sign of the a_n 's in region I. This change in sign only affects the root distribution as can be seen in figure 3(c) and 3(d), notice that the angle, b_n , remains the same though the radius e^{a_n} is changed in some cases to e^{-a_n} .

As a comparison the following relationships between I_{max} and I_{min} were calculated.

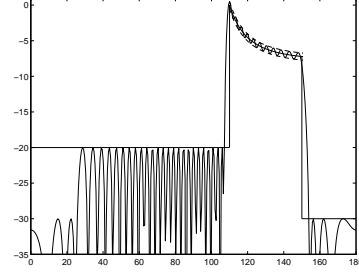
Before optimization:

$$\frac{I_{max}}{I_{min}} = 7.2083 \quad (10)$$

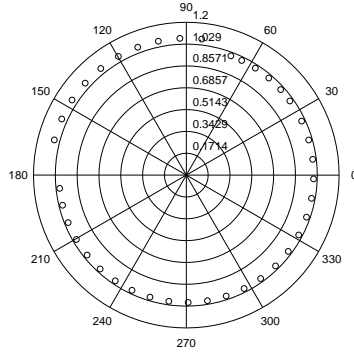
³The true computation of S is done for $20^\circ < \theta < 60^\circ$



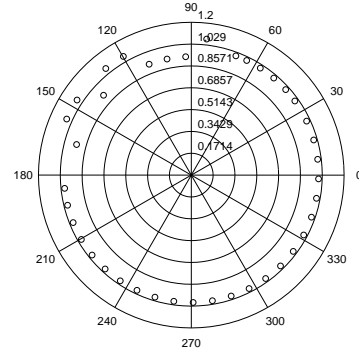
(a) Unoptimized pattern in the θ -plane



(b) Optimized pattern in the θ -plane



(c) Distribution of the roots for the unoptimized pattern



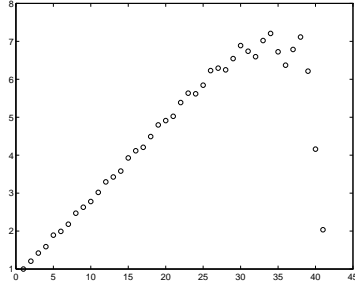
(d) Distribution of the roots for the optimized pattern

Figure 3: Antenna pattern and corresponding root distribution before and after optimizing the excitation I_n

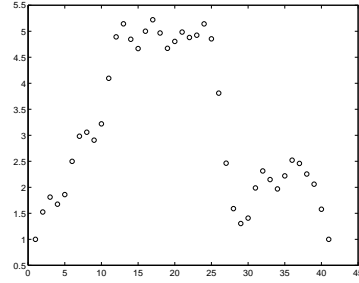
After optimization:

$$\frac{I_{max}}{I_{min}} = 5.2260 \quad (11)$$

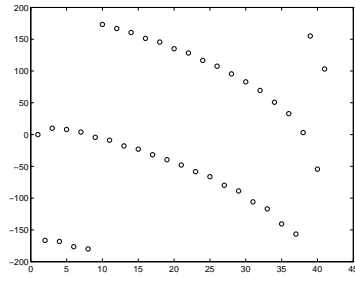
This shows clearly that the optimization well suits its purpose, and if used the right way maybe it makes the design and implementation of the real antenna possible. Also notice, as shown in figure 4, that the absolute value of excitation changes dramatically, when the optimization is done.



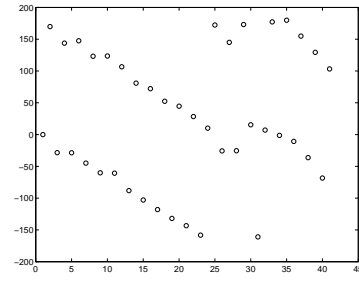
(a) Unoptimized amplitude distribution for the excitation



(b) Optimized amplitude distribution for the excitation



(c) Unoptimized phase distribution for the excitation



(d) Optimized phase distribution for the excitation

Figure 4: Distribution of the amplitude and phase before and after optimizing the excitation I_n

4 Optimization on excitation of the elements

When a_n and b_n is iterated to give an appropriate beamform, you can from these values calculate the excitation, I_n , of each element using formula (2). Since a_n and b_n give the roots of the polynomial, we used MATLAB's built-in `poly`-function.

4.1 Why optimization?

In [1] Orchard and Elliott suggests that mirroring any of the roots that is placed off the unit circle should give exactly the same beamform, but change excitation of the elements quite dramatically. They also suggest that the

best solution is the one that gives the lowest ratio in amplitude, I_{max}/I_{min} in excitation between the elements. As motivation on that they [1] say that if these requirements are fulfilled then the lowest coupling between elements are achieved, and therefore produces the best “calculations-to-real-world” solution. We wanted to give the best solution to the given problem, so we had to find the solution which gave the lowest ratio of amplitude on the elements. We used a “brute-force” algorithm since we had no idea of how the best solution could look like.

4.2 How should it be done?

The idea was to generate a matrix which, multiplied with a vector \mathbf{a} should give all permutations of a_n (see formulas 5 and 6). Then we should use it to calculate the excitations I_n and thus finding max, min and so on. The problem was, that if N_1 is the number of roots placed off the circle, we have 2^{N_1} solutions to the problem. In our example we had 41 elements and $N_1 = 15$ at first. This gives 32768 solutions to test for. The generated matrix was 15×32768 , which MATLAB couldn’t deal with at all. A better optimized pattern gave $N_1 = 10$ which leaved us with 1024 solutions to test, which was possible to calculate. The execution time was lowered from not achiveable to 20-30 seconds on a Digital ALPHA workstation.

The matrix we wanted to construct looked like

$$\begin{pmatrix} -1 & 1 & -1 & 1 & -1 & 1 & \dots \\ -1 & -1 & 1 & 1 & -1 & -1 & \dots \\ -1 & -1 & -1 & -1 & 1 & 1 & \dots \\ -1 & -1 & -1 & -1 & -1 & -1 & \dots \\ -1 & -1 & -1 & -1 & -1 & -1 & \dots \\ -1 & -1 & -1 & -1 & -1 & -1 & \dots \\ -1 & -1 & -1 & -1 & -1 & -1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \quad (12)$$

However we didn’t find a smart way to generate it in MATLAB. Instead we made our self a small C-program [4], see appendix B.1.

4.3 How well did it work?

Excitation of the elements in both the unoptimized and the optimized case can be studied in figure 4. Since we haven’t had the possibility to test the antenna in practice we can’t make any conclusion on how well it performs in real life. Simulations with NEC could be an alternative way to at least do a “near real life”-simulation for testing purposes.

From figure 4 we noticed that the excitation amplitude makes quite big leaps between each element in the optimized case, compared to the unoptimized case. This seems quite odd since the lowest coupling between elements is achieved when the difference between excitation is low for neighbouring elements. [1] calls the criterion we have used a “rough criterion”, which seems true. The idea is good, but maybe another criterion should be used?

This gave us an idea, why not try something like, “lowest mean of absolute value of amplitude between neighbouring elements” as criterion. Which may be written as:

$$\text{Min} \left(\frac{1}{N_1 - 1} \sum_{n=1}^{N_1-1} |a_{n+1} - a_n| \right) \quad (13)$$

The idea was easy to implement so why not give it a shot. The only thing that had to be changed was the selection algorithm, see appendix A.4. The result was nicer in many ways, as can be seen in figure 5. If compared to figure 4(b) there is easy to see that the two optimizing methods gives a quite different distribution of the excitation. Note that the relation between I_{max} and I_{min} in the alternative case is quite high, but the steps in amplitude between the neighbouring elements is smaller. Phase distribution is not presented since it always looks almost the same, independent of what root distribution you choose.

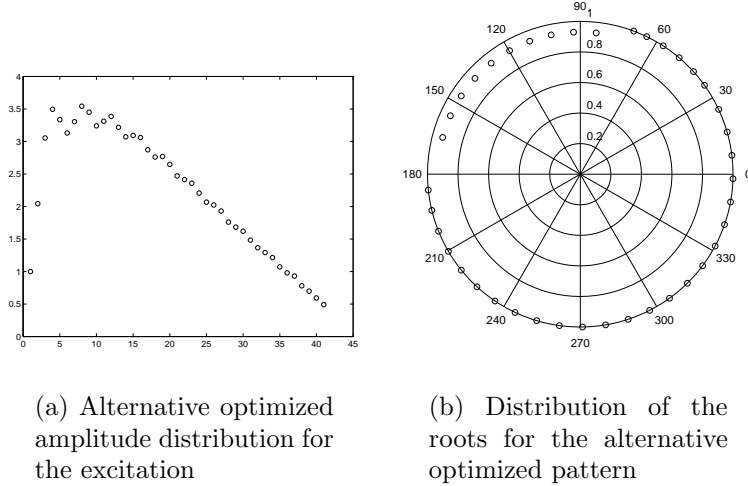


Figure 5: Distribution of the amplitude and roots after the alternative optimization

Clearly, there is a need for a better verification of what an optimized excitation is. Since [1] was published in 1985 someone maybe already have come up with a better definition of what a optimum excitation is. The results presented here is just what we found interesting and maybe the are useful in some way.

References

- [1] H.J. Orchard & R.S. Elliott *Optimising the synthesis of a shaped beam antenna patterns* Proc IEE, no 1 February 1985, page 63-68
- [2] Lennart Råde & Bertil Westergren *Beta Mathematics Handbook* Second Edition, Studentlitteratur, Lund, 1990
- [3] Lennart Edsberg *Användarhandledning för MATLAB v4.0* NADA KTH, Stockholm, 1993
- [4] Brian W. Kernighan & Dennis M. Ritchie *The C programming language* Second Edition, Prentice Hall, New Jersey, 1988

A MATLAB-code

A.1 main.m

```
%%% This is the main program for calculating
%%% antenna patterns using the Orchard Elliott method.
%%%
%%% Written by Johan Skatt, Royal Institute of Technology 1997
%%%
%%% Usage: You'll have to change the parameters in the
%%%         specification part to them that you've got.
%%%         You'll also have to change the parameter
%%%         given to the external C-program "binmat.c"
%%%         It should be equal to N1.
%%%
%%%         Eventually you would like to change some
%%%         other things later on, hopefully this wouldn't
%%%         cause any problems.

clear,clf

%%% Here's where you give the specifications !

% Makes theta to zero at broadside
thetar=pi/2;

% The limits for the beam
theta0=110*pi/180;
theta1=150*pi/180;

% N1 are the number of zeros in region I (the beam)
N1=10;
% N2 are the number of zeros in region II (the sidelobes)
N2=30;

n=1:N1+N2;

%%% Create wanted g

gg=[];
```

```

% -20 dB in first half of region II, using 25 zeros
gg=[gg -20*ones(1,25)];

% -30 dB in second half of region II, using 5 zeros
gg=[gg -30*ones(1,5)];

% 0.5dB ripple in region I, using 10 zeros
for k=1:21
    if rem(k,2)==0 gg=[gg -0.5];
    else gg=[gg 0.5]; end
end

%%% End of specification !

%%% Some handy constants

fi=-pi:0.01:pi;
K=180/pi;

%%% Some initializing

% Generate some nice starting values for a, b, C1 and C2
b=(2*n/42-1)*pi;
a=0.02*ones(1,N1);
C1=0;
C2=0;
% Assemble a, b and C1 into the iteration vector x
x=[a b C1];

%%% Here is the main loop !

for n=1:5
    % Find psi at maximas/minimas
    psii=Newton(x,N1,N2);
    psi2=psii(:,1:N2);
    psi1=psii(:,N2+1:2*N1+N2+1);

    % Find the rotate angle psir
    psi0=psii(length(psii));
    psir=psi0-pi*cos(theta0);

```

```

% Find the values for G-S at psi1, S is cosecans^2
C2=(csc(acos((psi0-psir)/pi)-thetar)).^2;
S=(csc(acos((psi1-psir)/pi)-thetar)).^2-C2;
g1=G(x,psi1,N1,N2)-S;

% Find the values for G at psi2
g2=G(x,psi2,N1,N2);

% Assemble into the gtop vector
gtop=[g2 g1];

% Calculate Jacobian
A=Jacobi(x,psii,N1,N2);

% Calculate deltax
deltax=A\((gg'-gtop)');
deltax=deltax';
x=x+deltax;
end

%%% End of main loop !

%%% Plot the wanted beam in the psi-plane
plot(K*fi,G(x,fi,N1,N2));
title('Beamform in the psi-plane');
pause

%%% Plot the wanted beam in the theta-plane, unoptimized
theta=0:0.01:pi;

% Plot specification
spec;
axis([0 180 -35 1]);
hold on;
pause

% Plot calculated beam
plot(K*theta,G(x,pi*cos(theta)+psir,N1,N2));
axis([0 180 -35 1]), hold off;
%title('Unoptimized Beamform in the theta-plane');
print -deps thetaunopt.eps

```



```

pause

%%% Plot the zeros in the w-plane
a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
a=[a2 a1];
b=[b2 b1];
polar(b,exp(a),'ro');
%title('Unoptimized zeros in the w-plane');
print -deps wunopt.eps
pause

%%% Calculate the relationship between the largest excitation
%%% Imax and the smallest Imin,
%%% to compare with the optimum version later on

nonoptimalI=max(abs(calcI(x,N1,N2)))/min(abs(calcI(x,N1,N2)))

% Plot the absolute value of the excitation
plot(abs(calcI(x,N1,N2)),'o');
print -deps absunoptex.eps
pause

% Plot the phase of the excitation
plot(K*angle(calcI(x,N1,N2)),'o');
print -deps phaseunoptex.eps
pause

%%% Calculate the optimum excitation, by means of minimizing
%%% the relationship Imax/Imin.
disp('Trying to optimize the result');
% Generate the binary matrix with an external program (N1=10)
disp('Generating the matrix');
!binmat 10 binary.m

[ourI]=bestI(x,N1,N2);

[optimalI,optimalIndex]=min(ourI(:,1));
optimalI

```

```

max(ourI(:,1));
x(1:N1) = x(1:N1).*ourI(optimalIndex, 2:N1+1);

% Plot the absolute value of the optimum excitation
plot(abs(calcI(x,N1,N2)), 'o');
print -deps absoptex.eps
pause

% Plot the phase of the excitation
plot(K*angle(calcI(x,N1,N2)), 'o');
print -deps phaseoptex.eps
pause

%%% Plot the wanted beam in the theta-plane, optimized

% Plot specification
spec;
hold on;

% Plot calculated beam
Q=G(x,pi*cos(theta)+psir,N1,N2);
% Allow a bit of floating up or down
Gmax=max(Q);
plot(K*theta,Q-Gmax+0.5);
axis([0 180 -35 1]), hold off;
%title('Optimized Beamform in the theta-plane');
print -deps thetaopt.eps
pause

%%% Plot the optimized zeros in the w-plane
a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
a=[a2 a1];
b=[b2 b1];
polar(b,exp(a), 'go');
%title('Optimized zeros in the w-plane');
print -deps wopt.eps

```

A.2 Newton.m

```
function [result]=Newton(x,N1,N2);
% Calculates the maximas and minimas, using Newton method.
% Returns a vector containing the values of psi at
% the extreme points.
% Written by Johan Skatt, Royal Institute of Technology 1997
% Usage: Newton(x,N1,N2)

a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
C1=x(:,2*N1+N2+1);
a=[a2 a1];
b=[b2 b1];

% Bild up the start psi-vector
psi=[(-pi+b(1))/2];
for n=1:N2-1
    psi=[psi (b(n)+b(n+1))/2];
end
for n=N2:N1+N2-1
    psi=[psi (b(n)+b(n+1))/2 b(n+1)];
end
psi=[psi 3];

M=20/log(10);
% The Newton algorithm for finding the zeros.
for n=1:length(psi)
    if psi(n)==0, psi(n)=0.0001; end
    x=psi(n); dx=x; iter=0;
    while abs(dx/x)>1e-4 & iter<20
        D=1-2*exp(a).*cos(x-b)+exp(2*a);
        Gprim=M*sum(exp(a).*sin(x-b)./D)-M*sin(x)./(2+2*cos(x));
        Q=M/(2+2*cos(x));
        Gbiss=M*sum((exp(a).*((1+exp(2*a)).*cos(x-b)-2*exp(a)))./D.^2)-Q;
        dx=-Gprim/Gbiss; x=x+dx; iter=iter+1;
    end
    if iter==20,
        disp('You are in trouble here, it does not converge');
```

```

    end
    result=[result x];
end

```

A.3 Jacobi.m

```

function [result]=Jacobi(x, psi,N1,N2);
% Calculates the Jacobian matrix according to
% Orchard et al formula 6.
% Returns the Jacobian matrix
% Written by Johan Skatt, Royal Institute of Technology 1997
% Usage: Jacobi(x,psi,N1,N2)

a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
C1=x(:,2*N1+N2+1);

M=20/log(10);
a=[a1 a2 a1];
b=[b1 b2 b1];
D=[];
for n=1:2*N1+N2+1
    D=[D; 1-2*exp(a).*cos(psi(n)-b)+exp(2*a)];
end

a=[a2 a1];
b=[b2 b1];
diffGb=[];
diffGa=[];
for n=1:2*N1+N2+1
    diffGa=[diffGa; M*exp(a1).*(exp(a1)-cos(psi(n)-b1))];
    diffGb=[diffGb; -M*exp(a).*sin(psi(n)-b)];
end
part1=[diffGa diffGb]./D;

result=[part1 ones(2*N1+N2+1,1)];

```

A.4 bestI.m

```
function [compare]=bestI(x,N1,N2)
%function [bestexc, resultx]=bestI(x,N1,N2)
% Returns the "best" excitation of elements and corresponding x
% Written by Stefan Petersen
% Royal Institute of Technology 1997

% The alternative method presented in the paper is commented out
% in the loop.

% Read the binary matrix
disp('Started getting the binary matrix. Please wait!');
binary;
disp('OK!');

xmod=x;
compare=[];
[size_binmat, dummy]=size(binmat(:,1));
disp('Optimizing');
for exc=1:size_binmat
    xmod(1:N1) = binmat(exc,:).*x(1:N1);
    I=calcI(xmod,N1,N2);
    % diffI = diff(I);
    % compare(exc,:) = [mean(abs(I)) binmat(exc,:)];
    compare(exc,:) = [max(abs(I)) / min(abs(I)) binmat(exc,:)];
end
disp('Optimizing ready');
```

A.5 calcI.m

```
function [result]=calcI(x,N1,N2)
% Returns excitation I of all elements in complex form.
% Written by Johan Skatt & Stefan Petersen
% Royal Institute of Technology 1997
% Usage: calcI(x,N1,N2)

a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
```

```

b1=x(:,N1+N2+1:2*N1+N2);
C1=x(:,2*N1+N2+1);
a=[a2 a1];
b=[b2 b1];

result=poly(exp(a+i*b));

```

A.6 G.m

```

function [result]=G(x, psi, N1, N2);
% Calculates the powerpattern function G according to
% Orchard et al formula 5. Returns a vector containing
% the value of G at the given psi angle(s).
% Written by Johan Skatt & Stefan Petersen
% Royal Institute of Technology 1997
% Usage: G(x,psi,N1,N2)

a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:2*N1);
b1=x(:,2*N1+1:2*N1+N2);
C1=x(:,2*N1+N2+1);

a=[a2 a1];
b=[b2 b1];

result=[];
for n=1:length(psi)
    part1=10.*log10(1-2*exp(a).*cos(psi(n)-b)+exp(2*a));
    part2=10*log10(2*(1+cos(psi(n))))+C1;
    result=[result sum(part1)+part2];
end

```

A.7 spec.m

```

% Plots the wanted beamform in the theta-plane
% Written by Johan Skatt, Royal Institute of Technology 1997

temp=110*pi/180:0.01:151*pi/180;
plot([0 110],[-20 -20],'c'), hold on;
plot([110 110],[-20 0],'c'), hold on;

```

```

S=(csc(temp-pi/2)).^2;
S=S-max(S);
plot(180*temp/pi,S,'c'), hold on;
plot([150 150],[min(S) -30],'c'), hold on;
plot([150 180],[-30 -30],'c'), hold on;

% Ripple +/- 0.5 dB
plot(180*temp/pi,S+0.5,'r--'), hold on;
plot(180*temp/pi,S-0.5,'r--'), hold on;

```

A.8 absmain.m

```

%%% This is an example of how an antenna pattern could be
%%% created using a triangle as the S function
%%%
%%% Written by Johan Skatt, Royal Institute of Technology 1997

clear,clf

%%% Here's where you give the specifications !

% Makes theta to zero at broadside
thetar=pi/2;

% The limits for the beam
theta0=110*pi/180;
theta1=150*pi/180;

% N1 are the number of zeros in region I (the beam)
N1=10;
% N2 are the number of zeros in region II (the sidelobes)
N2=30;

n=1:N1+N2;

%%% Create wanted g

gg=[];

% -20 dB in first half of region II, using 25 zeros
gg=[gg -20*ones(1,25)];

```

```

% -30 dB in second half of region II, using 5 zeros
gg=[gg -30*ones(1,5)];

% 0.5dB ripple in region I, using 10 zeros
for k=1:21
    if rem(k,2)==0 gg=[gg -0.5];
    else gg=[gg 0.5]; end
end

%%% End of specification !

%%% Some handy constants

fi=-pi:0.01:pi;
K=180/pi;

%%% Some initializing

% Generate some nice starting values for a, b, C1 and C2
b=(2*n/42-1)*pi;
a=0.02*ones(1,N1);
C1=0;
C2=0;
% Assemble a, b and C1 into the iteration vector x
x=[a b C1];

%%% Here is the main loop !

for n=1:5
    % Find psi at maximas/minimas
    psii=Newton(x,N1,N2);
    psi2=psii(:,1:N2);
    psi1=psii(:,N2+1:2*N1+N2+1);

    % Find the rotate angle psir
    psi0=psii(length(psii));
    psir=psi0-pi*cos(theta0);

    % Find the values for G-S at psi1, S is the triangle function

```



```

S=[0 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1 0]-10;
g1=G(x,psi1,N1,N2)-S;

% Find the values for G at psi2
g2=G(x,psi2,N1,N2);

% Assemble into the gtop vector
gtop=[g2 g1];

% Calculate Jacobian
A=Jacobi(x,psii,N1,N2);

% Calculate deltax
deltax=A\(gg'-gtop');
deltax=deltax';
x=x+deltax;
end

%%% End of main loop !

%%% Plot the wanted beam in the psi-plane
plot(K*fi,G(x,fi,N1,N2)), axis([-180 180 -35 20]);
%title('Beamform in the psi-plane');
print -deps abs.eps
pause

%%% Plot the zeros in the w-plane
a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
a=[a2 a1];
b=[b2 b1];
polar(b,exp(a),'ro');
title('Unoptimized zeros in the w-plane');
pause

%%% Calculate the optimum excitation, by means of minimazing
%%% the relationship Imax/Imin.
disp('Trying to optimize the result');

```

```

% Generate the binary matrix with an external program (N1=10)
disp('Generating the matrix');
!binmat 10 binary.m

[ourI]=bestI(x,N1,N2);

[optimalI,optimalIndex]=min(ourI(:,1));
x(1:N1) = x(1:N1).*ourI(optimalIndex, 2:N1+1);

%%% Plot the optimized zeros in the w-plane
a1=x(:,1:N1);
a2=zeros(1,N2);
b2=x(:,N1+1:N1+N2);
b1=x(:,N1+N2+1:2*N1+N2);
a=[a2 a1];
b=[b2 b1];
polar(b,exp(a),'go');
title('Optimized zeros in the w-plane');

```

B C-code

B.1 binmat.c

```

/* Program to generate a binary matrix in
 * textformat for simple inclusion
 * in Matlab.N cols and 2^N rows.
 *
 * Stefan Petersen, Royal Institute of Technology, 1997
 */

#include <stdlib.h>
#include <stdio.h>

void usage()
{
    printf("(C) Stefan Petersen, ");
    printf("Royal Institute of Technology, 1997\n");
    printf("\nUsage:\n");
    printf("binmat <N> <filename>\n");
}

```

```

printf("where N is number of zeroes off the circle and \n");
printf("filename is the name of file to ");
printf("save the matrix in.\n");

return;
}

void main(int argc, char *argv[])
{
    FILE *fs;
    char *filename;
    char matrixname[] = "binmat";
    int n,N;
    long temp,m,M;

    if (argc != 3) {
        printf("Wrong number of arguments!\n");
        usage();
        exit(1);
    }

    if ((N = atoi(argv[1])) == 0){
        printf("Illegal form of N given\n");
        usage();
        exit(1);
    }
    M = (1<<N);

    filename = argv[2];

    if((fs = fopen(filename, "w+")) == NULL){
        fprintf(stderr, "Couldn't create %s\n", filename);
        exit(1);
    }

    fprintf(fs, "%s=", matrixname);

    for(m = 0; m < M; m++){

```

```
temp = m;
for(n = 0; n < N; n++){
    fprintf(fs, "%d ", (temp & 0x01L)?1:-1);
    temp>>=1;
}
fprintf(fs, ";\n");
}

fprintf(fs, "];\n");
fclose(fs);
return;
}
```