Optimization of response from 2D arrays for medical ultrasound

May 1996

Preface

This thesis is written as a required part of the Candidatus Scientiarum (Master of Science) degree in informatics at the Department of Informatics, University of Oslo, Norway. The work was started in September 1994 and finished in May 1996.

This thesis is a theoretical approach to analyze problems arising in the design of ultrasound transducer arrays. It combines the fields of array signal processing and mathematical optimization. Some background theory is included, but it is assumed that the reader has a fundamental background from digital signal processing. It is also assumed that the reader is familiar with linear algebra and matrix notation.

The background theory is included both to motivate and to introduce the notation required in the proposed methods. It will hopefully give a wider understanding of both the methods and which conditions they apply to. Some theoretical details must necessarily be elaborated, but this presentation is in the interest of simplicity rather than intricate details.

Two optimization methods are proposed to optimize the response from ultrasound array transducers by weighting. One performs only weighting and the other performs simultaneous thinning and weighting. These methods are implemented and computer simulations on several arrays have been carried out.

Some effort has also been made in keeping a consistent notation throughout this thesis. Boldface letters have for instance been reserved for matrices. For example, $\mathbf{A}$ and $\mathbf{b}$ typically denotes a matrix and a column vector, respectively. Physical vectors are denoted with arrows as $\vec{x}$.

Finally, I would like to thank my supervisor, Professor Sverre Holm, for his encouragement and assistance through this work. Also thanks to Dr. Geir Dahl for his contribution of ideas to the optimization methods.

Oslo, May 1996

Bjørnar Elgetun

Contents

Chapter 1

Introduction

1.1 Medical ultrasound

Ultrasound was introduced to medical diagnosis in the 1970's and is now available at many hospitals for obtaining images of internal tissues in the human body. Ultrasound imaging is mainly used in locating the position of the fetus during pregnancy and in cardiology. As ultrasound imaging is non-invasive and non-radiating, it is considered as a harmless anatomical imaging method. Thus, it provides an alternative to X-rays in medicine for certain applications.

The rapid advances in computer technology and the improvements in *transducer*[1] design has drawn the attention to three-dimensional (3D) ultrasound imaging. This is sometimes referred to as volumetric imaging and is at present offered by the first generation 3D imaging systems. An example of a 3D image compared to the conventional 2D image is given in figure 1.1.

The 3D imaging mode's significance in medicine to make more reliable diagnosis is obvious. Further benefits from 3D imaging will certainly evolve. It has already turned out to be a valuable tool for instance in communication between cardiologists and surgeons.

Real-time 3D imaging has probably always been the ultimate goal in diagnostic ultrasound imaging. In contrast, the first generation 3D imaging systems are highly non-real-time. The state-of-the-art systems require time of the order of 15 minutes or more for data acquisition and image reconstruction. This impedes clinical efficiency, applicability and patient throughput.

[1] An ultrasonic transducer is a device that produces and receives ultrasound.

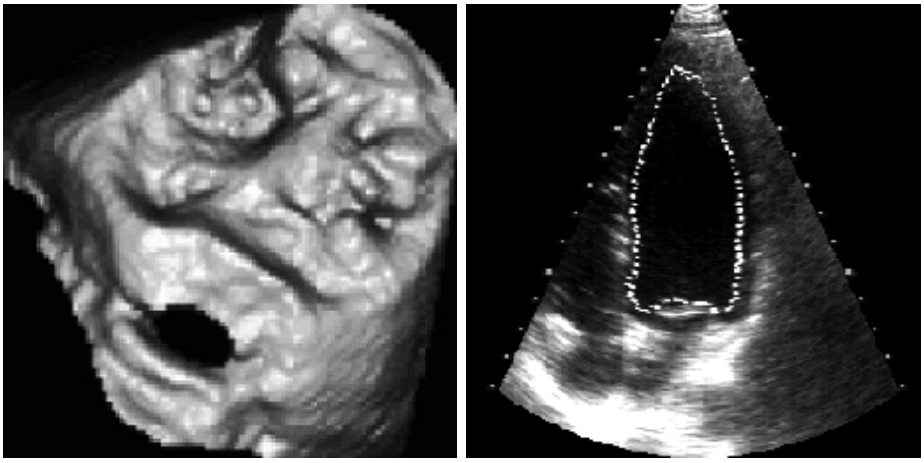

Figure 1.1: To the left, an example of a 3D cardiology image of the mitral valve and the open aorta is shown. It is obtained with a first generation 3D non-real time imaging system. A conventional 2D ultrasound image of the heart is shown to the right.

The first generation 3D imaging systems are based on 1D phased arrays which allow *electronic steering* only in one plane of a 3D segment. To obtain full 3D steering, the array is moved mechanically in different scan-planes. The mechanical steering is a time consuming factor, and it requires a comparatively large equipment setup.

Thus, the ideal transducer for 3D imaging is the *2D phased array* [3]. Each of the transducer elements are delayed or phased individually, which allows electronically steering and focusing in all directions of a 3D segment. This will eliminate the mechanical moving parts present in the first generation 3D imaging systems. Both the equipment and the scanning operation will be simplified with the 2D phased array.

The major problem with the 2D phased array is a large number of channels to handle. As the present linear phased array transducers use $64 - 128$ elements, the 2D array should use from $64 \times 64 = 4096$ to $128 \times 128 = 16384$ elements. This represents a significant problem in cable connection and electronics, and it requires a large amount of computer resources.

1.2 Objective of this thesis

The size and complexity of the 2D phased array still makes 3D real-time imaging futuristic − But could it be that a 2D phased array would operate satisfactory with a *severe* reduction in the number of array elements? This is the underlying question of this thesis.

Some techniques have been suggested to reduce the number of the elements of 2D arrays. Both random thinning [29, 43] and thinning due to a mathematical optimization criterion [24, 25, 30] are considered.

To compensate for the reduction of array elements, the effect of element *weighting*[2] is investigated. By combining theory from array signal processing and mathematical optimization, an optimization problem is established. For the element weighting problem, a *linear program* is set up and solved utilizing the well-known *Simplex method* from mathematical programming. The response from several randomly thinned arrays have been simulated using this implementation.

There are probably more clever ways to reduce the number of array elements than picking them by random. This leads to a mathematical formulation that optimizes both the *thinning* and *weighting* simultaneously. In this thesis the problem is established and solved by a *mixed integer programming* approach using the *branch-and-bound method*. The gain of optimal element placing in contrast to random thinning is examined.

Some array signal processing and mathematical optimization theory is also included. This will hopefully give a wider understanding of the methods used and which conditions they apply to.

[2]Element weighting is termed as shading or tapering as well. This means that each element signal is individually amplified in addition to the delay required for electronically steering.

Chapter 2

Ultrasound imaging

This chapter is intended as a primer for the succeeding chapters. It reveals the main principles in ultrasound imaging. The different technological parts of an ultrasound imaging system will be described briefly, with emphasis on the ultrasound transducer. Ultrasound imaging in respect of the 3D imaging mode is also elaborated in this chapter. Both the new possible applications evolving as well as the technical aspect of 3D imaging is discussed.

2.1 The ultrasound imaging system

Ultrasound is a term used to describe soundlike waves whose frequency is *above* the range of normal human hearing, which extend from about 30 to $20,000$ Hz. In medical ultrasound imaging, such waves are passed through the body and the echoes are registered. By processing the backscattered signals, an ultrasound image is constructed.

The ultrasound wave signals are generated and received by a *transducer* or a *probe*, which is the basic device in an ultrasound imaging system. The transducer is controlled by a *RF-unit*, which is responsible for focusing and steering of the ultrasound waves, often called *beams*. The RF-unit, equipped with both analog and digital electronics, is thus a *beamformer* operating on signals in both space and time.

The *scanline processor* extracts the data acquired by the RF-unit. Image data for each scanline as well as data for color and spectral Doppler is extracted by time domain signal processing. The scanline data is sent to the *image processor*, which performs image construction and filtering. Finally,

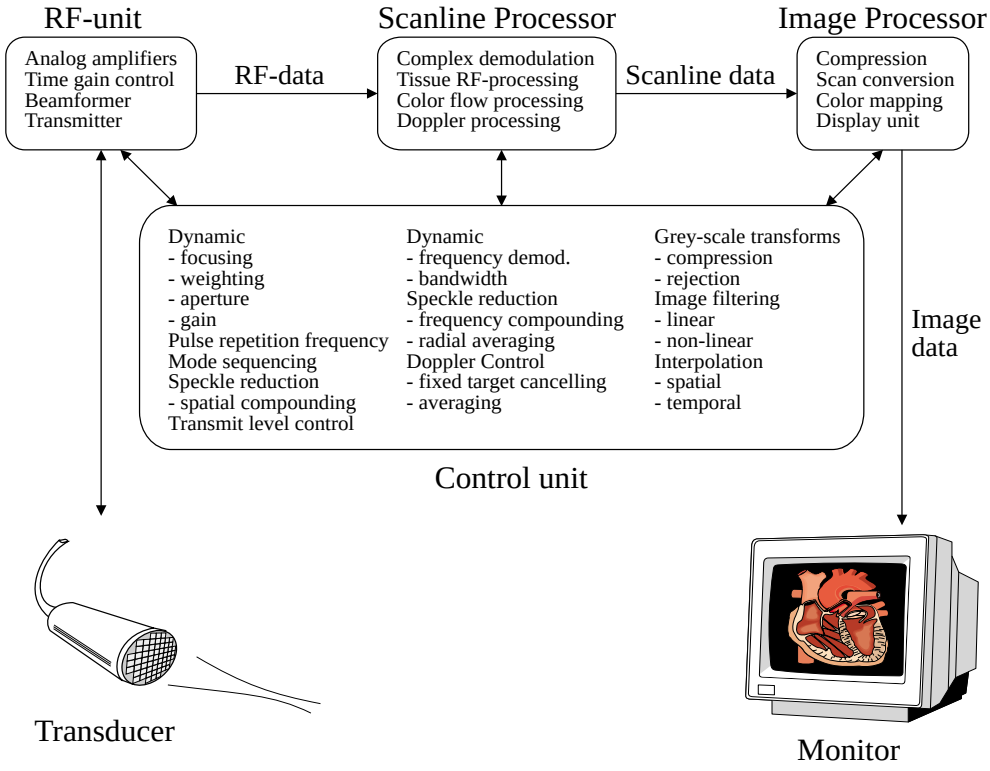

Figure 2.1: A block diagram of an ultrasound imaging system.

the image data is sent to the display, as shown in figure 2.1.

2.1.1 The ultrasound transducer

Today's ultrasound transducers are based on the effect of *piezoelectricity*, which was discovered in 1880 by Pierre and Jacques Curie. This discovery made it possible to transform electricity into pressure waves and vice versa. Utilizing this effect, the ultrasound transducer transforms electricity into high-frequency ultrasound. Medical ultrasound imaging usually apply frequencies ranging from $2 - 10$ MHz [2]. The principle of piezoelectricity is illustrated in figure 2.2.

The modern ultrasound transducers are made of specially cut crystals of materials such as quartz or ceramics such as barium titanate and lead zirconate. An alternating electrical voltage is applied across the opposite faces

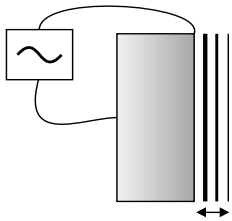

Figure 2.2: A transducer element utilizing the effect of piezoelectricity. Applying an electrical voltage makes the plate vibrating and vice versa.

of a plate made of such a material. This produces an alternating expansion and contraction of the plate at the impressed frequency. This phenomenon is called piezoelectricity.

Similar effects are observed in ceramics. Ceramic objects have the added advantage of being able to be cast in the form of plates, rings, cylinders, and other special shapes that are convenient for engineering applications. In addition, some materials, such as cadmium sulfide, can be deposited in thin films on a solid medium. Such material can then serve as a transducer. Still other ultrasonic transducers are produced in ferromagnetic materials by varying the magnetic-field intensity in the material [41].

2.1.2 Different transducer types

There are numerous transducers available for different applications. There are transducers for external as well as internal use in the human body. Obviously, the transducers designed for insertion into the body have both shape and size constraints. Another example is the size of the transducer intended for cardiology, which is limited by the distance of about 2 cm between the ribs.

The simplest ultrasound transducer is an unfocused piston transducer which consists of one piezoelectric disc. The surface of this simple transducer may be slightly curved to focus the ultrasound energy at a spatial point. This is the same principle utilized in optical lenses. These transducers have a continuous *aperture*[1] consisting of one element.

To achieve better control, *array transducers* are preferred in medical imag-

[1]The aperture is the size of the ultrasound transducer, usually given as the diameter.

ing. These consists of two or more transducer elements which may be controlled individually. The most common transducers in medical ultrasound are the *annular array* and the *phased array transducers* which are shown in figure 2.3.

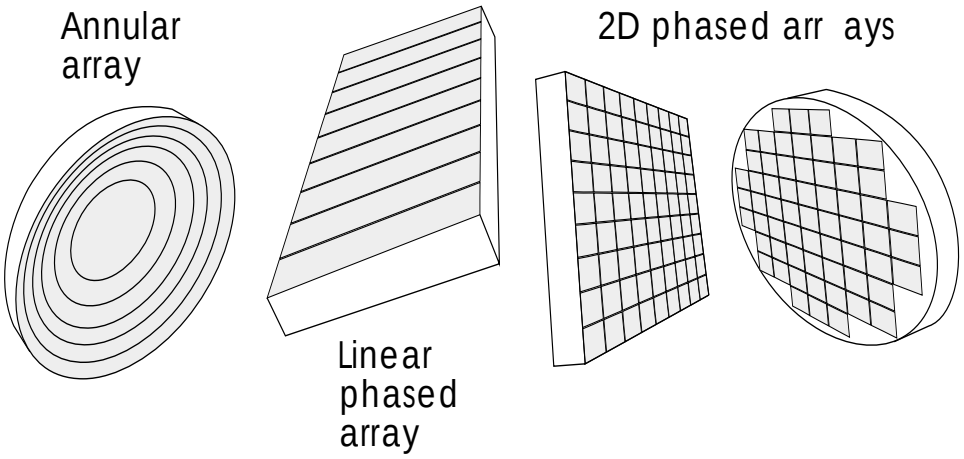

Figure 2.3: Different types of array transducers for medical imaging. The annular array is conceptually different from the phased arrays.

By individually delaying each ring of the annular array, it allows *dynamic focussing* of the ultrasound energy, but the beam must still be steered mechanically. The phased array transducers are thus the most flexible, since they allow both *dynamic focussing* and *electronically steering*. The drawback with the latter is the considerably large number of elements required.

2.2 3D imaging

With the recent introduction of 3D imaging, a new era of ultrasound imaging is established. 3D imaging is still at the research stadium, even though the first generation imaging systems are already available. The 3D imaging systems still suffer from high production costs, complex equipment setup and a long image generation time. But it has already been stated that there is an enormous potential for 3D ultrasound imaging systems.

2.2.1 Applications

The main reason for the potential of 3D imaging is simply the fact that we are interested in imaging 3D objects. With the conventional ultrasound images, an imaging expert such as a cardiologist has to interpret the speckled 2D images. He has to form an idea of the characteristics of the object, which then has to be explained to the medical staff prior to the diagnosis or surgery. If the 3D model instead is obtained directly with the ultrasound imaging system, it will obviously lead to more certain diagnosis.

When the 3D model of the actual object is created, it can be viewed and analyzed from different angles and observation planes. The whole model may also be stored on a disc for later analysis. In combination with data assisted construction, a silicone model of the object may be created. This model could be examined by the surgeon prior to a complicated surgery. Combined with a virtual reality helmet, the surgeon might prepare for the operation having an inside-heart-walk.

2.2.2 Technical considerations

The ideal transducer for 3D imaging is a 2D phased array [3], which allows electronically steering in all directions of a 3D segment. For the futuristic real time 3D imaging, an electronically steered array is preferred to maintain the frame rate and image quality. In contrast, the present first generation 3D imaging systems are based on a comparatively slower mechanic rotation, tilting or translation of a 1D transducer in one of the directions, which clearly restricts the frame rate.

As already stated, a 2D phased array transducer will necessarily need a large number of elements, which leads to severe fabrication difficulties in electrical connection. The total costs is also related to the number of elements. Another drawback with these transducers is the extremely low transducer signal to noise ratio (SNR). Unfortunately, the poor SNR of 2D arrays may not allow the advantages of element weighting [37].

Today's largest 2D phased arrays consists of about 500 elements, which is significantly less than the theoretically requirement of at least $64 \times 64 = 4096$ elements for 2D arrays. With the introduction of optoelectronic transmitters together with fiber-optic cables allowing considerable size reduction, this gap may be reduced. Another possible way to circumvent the problem, is to reduce the large number of array elements while maintaining aperture by

element thinning, which is elaborated later in this thesis.

The problem with the low SNR in 2D arrays may be overcome with the application of multilayer ceramic elements. It is just a matter of time until the 2D phased arrays will be accepted as viable replacements for the linear arrays of today [37].

Chapter 3

Ultrasound wave propagation

Ultrasound waves produced by transducers are physical signals governed by laws for wave propagation, in particular the *wave equation*. During propagation, the ultrasound waves are affected by tissue structures inside the body and physical phenomena like *dispersion*, *attenuation, refraction* and *diffraction*.

In this chapter, the necessary mathematical notation describing propagating waves and wavefields will be given. This involves functions of both space and time. Trying to characterize the wavefield by array transducers, the concept of spatial sampling is introduced. A *spatial sampling theorem* will be established, which resemble the *Nyquist* sampling theorem in digital signal processing.

3.1 Ultrasound waves

Ultrasound waves are caused by oscillations of the molecules in a material. The molecules are oscillating back and forth about their equilibrium points, which produces *longitudal pressure waves*. These waves propagate in a direction parallel to the molecule's oscillating motion.

From *Huygens' principle* [2, 14], the ultrasound wavefront is formed by the interference between spherical waves from each point on the transducer surface. As the partial waves spread spherically in space, the curvature of the circular wave contours decrease. When far enough from the source, the wave contours appear to have insignificant curvature and the wavefront is approximately a *plane wave*.

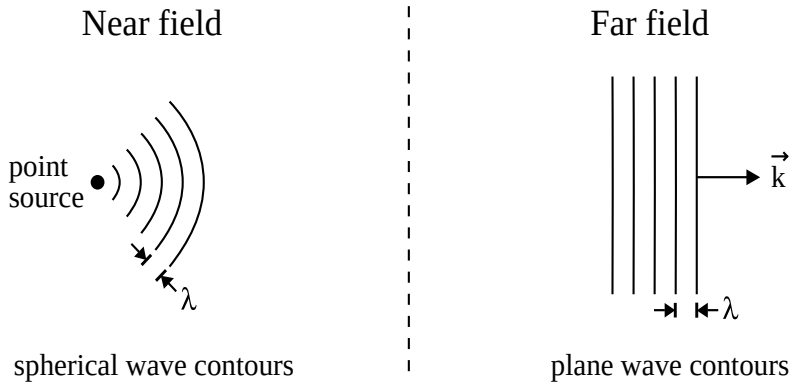

Figure 3.1: An ultrasound wave emitted by a point source. The different wave characteristics in the near and far-field is illustrated.

Consequently, the characteristics of the waves depend on distance from the source. In the *near-field* we have to assume *spherical waves,* while in the *far-field* we may assume *plane waves.* Defining the borderline between the near-field and the far-field is application dependent. In [2] for instance, the near-field of a circular ultrasound transducer extends from the transducer surface to $D^2/2\lambda$, where D is the diameter and λ is the actual wavelength.

3.1.1 Wave parameters

As waves propagate through space, they are characterized both by *temporal* and *spatial* parameters. At a spatial point, ultrasound waves passing produce oscillating pressure variations with a *temporal frequency* denoted f. The normalized *temporal angular frequency* is conveniently given as $\omega = 2\pi \cdot f$. The spatial distance λ between two pressure maxima of the wave is the *wavelength* and is related to the propagation speed c and the temporal frequency through

$$\lambda = \frac{c}{f} = c \cdot \frac{2\pi}{\omega} \tag{3.1}$$

The *wavenumber vector* is a quantity used to describe the spatial content of a propagating wave. As shown in figure 3.1, the *direction* of the wavenumber vector $\vec{k}$ is parallel to the propagation direction of the wave. The magnitude of the wavenumber vector is related to the wavelength as

$$\left|\vec{k}\right| = \frac{2\pi}{\lambda} = \frac{\omega}{c} \tag{3.2}$$

which also may be interpreted as the *spatial frequency* of the propagating wave.

The *slowness vector* is conveniently introduced to simplify notation. It is defined as $\vec{\alpha} = \omega\,\vec{k}$ which points in the same direction as the wavenumber vector. The magnitude of the slowness vector is inversely related to the wave propagation speed c and given as

$$|\vec{\alpha}| = \frac{1}{c} \tag{3.3}$$

3.2 The wave equation

The propagating ultrasound waves are governed by the wave equation, which is *the* equation in array signal processing. It is based on *Maxwell's equations*, a set of partial differential equations that describe the evolution in space and time of the electromagnetic field.

Unlike electromagnetic waves, ultrasound waves can not exist in a vacuum, but still they share the same wave properties. Deriving the wave equation for ultrasound waves is more complicated, since no single unified set of equations governs acoustics. See for instance [9] for a detailed derivation.

Anyway, the acoustic and electromagnetic wave equations take the same form

$$\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2} = \frac{1}{c^2}\frac{\partial^2 s}{\partial t^2} \tag{3.4}$$

where $s\,(x, y, z, t)$ represents the *sound pressure* in space and time in the acoustic case. The only parameter c in the wave equation may be interpreted as the *propagation speed* of the travelling wave. In biological tissue, the ultrasound waves propagate at a speed of $c \approx 1540$ m/s.

3.2.1 Solutions to the wave equation

The wave equation in (3.4) have numerous solutions, including both *spherical* and *plane* propagating waves. The following derivation shows that a propagating plane wave is a solution. Using the *linearity* of the wave equation and the *superposition* of weighted plane waves, the conclusion is that *any signal*

satisfies the wave equation, and the *shape* of the wave is perfectly preserved as it propagates.

Assume first that the sound pressure $s(x, y, z, t)$ of an ultrasound wave may be modeled as a monochromatic[1] propagating plane wave having the complex exponential form

$$s(x, y, z, t) = A \exp\{j\ (\omega t - k_x x - k_y y - k_z z)\} \tag{3.5}$$

where A is a complex constant, ω is the angular frequency with $\omega \geq 0$ and $\vec{k} = (k_x, k_y, k_z)$ is the wavenumber vector. Note that the sound pressure function $s(x, y, z, t)$ may be calculated for all space and time. As the complex exponential is *periodic* both in space and time, it is indeed a propagating wave.

Substituting (3.5) into the wave equation (3.4) and cancelling $s(x, y, z, t)$ gives us an equation constraining wavenumber and frequency

$$\left|\vec{k}\right|^2 = k_x^2 + k_y^2 + k_z^2 = \frac{\omega^2}{c^2} \tag{3.6}$$

As long as this constraint is satisfied, a plane wave with the complex exponential form in (3.5) is *a* solution to the wave equation.

By Introducing vector notation for the spatial location $\vec{x} = (x, y, z)$ and substituting the wavenumber vector $\vec{k}$ with the slowness vector $\vec{\alpha}$ from (3.3) we can rewrite the spatiotemporal plane wave in (3.5) as

$$s(\vec{x}, t) = A \exp\{j\,\omega\,(t - \vec{\alpha} \cdot \vec{x})\}$$

When A and ω are fixed, we may write the sound pressure function $s(\vec{x}, t)$ of the plane wave as a function of one argument as $s(\vec{x}, t) = s(t - \vec{\alpha} \cdot \vec{x})$.

Furthermore, a *weighted linear combination* of such propagating plane waves

$$s(t - \vec{\alpha} \cdot \vec{x}) = \sum_{-\infty}^{\infty} S_n \exp\{j\,\omega\,(t - \vec{\alpha} \cdot \vec{x})\} \tag{3.7}$$

is also a solution to the wave equation. Due to the *linearity property* of the integral [9], an *integral* of weighted propagating plane waves

$$s(t - \vec{\alpha} \cdot \vec{x}) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) \exp\{j\,\omega\,(t - \vec{\alpha} \cdot \vec{x})\}\ d\omega \tag{3.8}$$

[1]The term *monochromatic* means *one color* and origins from optics. Here it refers to a wave with one temporal frequency ω.

is too a solution to the wave equation. From *Fourier theory*, this integral can form *any* arbitrary wave signal whose frequency content is expressed with $S(\omega)$.

The linearity of the wave equation also implies that many plane waves propagating in different directions can exist *simultaneously* in the wavefield. The waves pass through each other unperturbed. A similar derivation applies to spherical waves by expressing the wave equation in spherical coordinates.

3.3 Physical implications

The wave equation in (3.4) demands *homogeneous*, *linear* and *lossless* media. Unfortunately, the human body is not such an *ideal* medium. Ultrasound waves propagating in the human body are affected by physical phenomena like *dispersion*, *attenuation, refraction* and *diffraction* which deviate from the demands of the wave equation.

In *dispersive* media, the wavenumber-frequency relation in (3.6) is *non-linear*, as the propagation speed is a frequency dependent function $c(\omega)$. The wave equation in (3.4) does not yield a dispersive solution, but this may be overcome by *bandlimiting* the signals or *augmenting* the wave equation for dispersion.

Waves propagating in an *attenuating* medium looses energy as they pass through the medium. Attenuation may be modeled by augmenting a *damping* term to the wave equation. Although dispersion and attenuation are two different phenomena, most realistic lossy media also demands dispersion. In ultrasound imaging, this is seen as the higher frequency ultrasound waves are more attenuated.

Refraction occurs when propagating ultrasound waves meet boundaries between structures with different propagation speeds. As a wave encounters such a discontinuity in the medium, it generally splits into a *reflected* wave and a *transmitted* wave[2]. This may be expressed as $\vec{k}_i \cdot \vec{x} = \vec{k}_r \cdot \vec{x} = \vec{k}_t \cdot \vec{x}$ where $\vec{k}_i$, $\vec{k}_r$ and $\vec{k}_t$ are the wavenumber vectors for the incident, reflected and transmitted waves, respectively, and $\vec{x}$ is a point on the boundary. This may also be interpreted as a version of the well known *Snell's law* from optics.

As sound waves bend around corners, ultrasound waves do. This phenomena is called *diffraction* and occurs when the waves meet structures whose

[2]Ultrasound imaging is based on measuring the waves *reflected* from different structures inside the body.

size is comparable to the wavelength. In medical ultrasound this extends from about $0.15 - 0.75$ mm. The straight line propagation assumption in the wave equation is not fulfilled in this case.

3.4 Spatial sampling

Let $f(\vec{x}, t)$ be the value of the wavefield in space and time. To reconstruct the wave signals, the array transducer samples the wavefield both in space and time at each sensor. From digital signal processing, we know that sampling in time may introduce ambiguities as *aliasing*. This is also true for spatial sampling.

According to the *Nyquist* sampling theorem found in most signal processing textbooks, we can reconstruct *any* signal bandlimited to frequencies below $\omega_0 = \frac{2\pi}{T_0}$ as long as the sampling period T is such that

$$T \leq \frac{\pi}{\omega_0} = \frac{T_0}{2} \tag{3.9}$$

where we need at least two samples a period of the signal. If the sampling period T is such that $T > \frac{T_0}{2}$, which violates (3.9), the signal is *undersampled.* High-frequency signals may then appear as low-frequency signals, which is aliasing.

From (3.2) the magnitude of the wavenumber vector may be interpreted as the spatial frequency of a signal. The *Nyquist* sampling theorem also applies here. If the wave signals are bandlimited to wavenumber magnitudes below $\left|\vec{k}_0\right| = \frac{2\pi}{\lambda_0} = \frac{\omega_0}{c}$, then they can be periodically sampled without loss of information as long as the spatial sampling period d is such that

$$d \leq \frac{\pi}{\left|\vec{k}_0\right|} = c \cdot \frac{\pi}{\omega_0} = \frac{\lambda_0}{2} \tag{3.10}$$

The array elements should then have an interelement spacing of $\lambda/2$ or less to avoid spatial undersampling, where λ is the center frequency of the generated ultrasound waves.

Chapter 4

Beamforming

In ultrasound imaging, the array transducers may send and receive *beams* of energy applying both a temporal *delay* and an amplitude *weight* to each array element. The term *beamforming* is used on a wide variety of algorithms that spatially points the array in fixed directions by adjusting each delay and weight. Note that this is an algorithmic concept and beamforming algorithms act like spatial filters.

In digital signal processing, a filter is characterized from it's frequency response. The *array pattern* similarly characterizes a spatial filter in array signal processing. According to this, the array pattern is deduced from the *delay-and-sum* beamforming algorithm in this chapter. The resemblance between the array pattern and the *Fourier transform* of familiar *window functions* in digital signal processing [1, 13] is also elaborated.

The array pattern is thoroughly analyzed, with emphasis on the *angular array pattern*. The ambiguities with the spherical mapping due to the angular array pattern will hopefully clarify through this chapter. The beampattern, which is defined through the array pattern is also introduced. Finally, a discussion on how the array pattern affects image quality is included.

4.1 The delay-and-sum beamformer

Delay-and-sum beamforming, the oldest and simplest array signal processing algorithm remains a powerful approach today. By delaying and summing each of the array element outputs, propagating waves in different directions may be characterized. The algorithm also increases the signal to noise ratio

as derived in [21].

Let the wavefield $f(\vec{x}, t)$ be sampled at the spatial location $\vec{x}_m \in \mathbb{R}^3$ by the m-th sensor. The measured waveform at this sensor is then $y_m(t) = f(\vec{x}_m, t)$. The delay-and-sum beamformer combines each of these M sensor outputs applying a fixed temporal delay $\Delta_m \in \mathbb{R}$ and an amplitude weight $w_m \in \mathbb{R}$. The output signal $z(t)$ from the delay-and-sum beamformer is then defined as

$$z(t) = \sum_{m=1}^{M} w_m y_m \left(t - \Delta_m\right) \tag{4.1}$$

where the delays Δ_m are adjusted to focus the array in different spatial directions, which is also called *phase-steering*. The weights w_m influences the beam and noise characteristics.

4.2 The array pattern

To characterize the delay-and-sum beamformer's directivity, the *array pattern* is examined. The array pattern is simply the delay-and-sum beamformer's response to a monochromatic plane wave[1] impinging on the array from different directions. As a superposition of plane waves expresses an arbitrary wavefield, the *plane wave response determines the beamformer's output for the general case.*

Assume that the wavefield $f(\vec{x}, t)$ consists of a monochromatic wave with temporal frequency ω^0 propagating with direction and spatial frequency given by the slowness vector $\vec{\alpha}^0 \in \mathbb{R}^3$. The wavefield is then

$$f(\vec{x}, t) = \exp\left\{j\omega^0 \left(t - \vec{\alpha}^0 \cdot \vec{x}\right)\right\}$$

The wavefield measured at the m-th sensor $y_m(t)$ is

$$y_m(t) = f(\vec{x}_m, t) = \exp\left\{j\omega^0 \left(t - \vec{\alpha}^0 \cdot \vec{x}_m\right)\right\}$$

Let the beamformer be steered to look for plane waves with slowness vector $\vec{\alpha} \in \mathbb{R}^3$. This is attained by choosing the set of delays Δ_m as

[1] With the plane wave assumption, the model is only valid in the *far-field*. Thus it is often called the *far-field array pattern*. The near-field array pattern may also be derived [21].

$$\Delta_m = -\vec{\alpha} \cdot \vec{x}_m \tag{4.2}$$

Substituting this into (4.1) we can write the delay-and-sum beamformer's output $z(t) \in \mathbb{C}$ to the monochromatic wave

$$z(t) = \sum_{m=1}^{M} w_m \exp\left\{ j\omega^0 \left(t + \left(\vec{\alpha} - \vec{\alpha}^0 \right) \cdot \vec{x}_m \right) \right\}$$

The temporal content $\exp\{j\omega^0 t\}$ may be extracted from this equation

$$z(t) = \left[\sum_{m=1}^{M} w_m \exp\left\{ j\omega^0 \left(\vec{\alpha} - \vec{\alpha}^0 \right) \cdot \vec{x}_m \right\} \right] \exp\left\{ j\omega^0 t \right\}$$

By introducing the wavenumber vector $\vec{k}^0 = \omega^0 \vec{\alpha}^0 \in \mathbb{R}^3$ for the propagating wave, we finally get the delay-and-sum beamformer's output $z(t)$ in the monochromatic case

$$z(t) = W\left(\omega^0 \vec{\alpha} - \vec{k}^0 \right) \exp\left\{ j\omega^0 t \right\} \tag{4.3}$$

where $W(\cdot)$ denotes the Fourier transform of the sensor weights

$$W\left(\vec{k} \right) = \sum_{m=1}^{M} w_m \exp\left\{ j\vec{k} \cdot \vec{x}_m \right\} \tag{4.4}$$

which is also the *array pattern* determined for the wavenumber vector $\vec{k} \in \mathbb{R}^3$.

Note the resemblance with the *frequency response* used in digital signal processing. The frequency response $H\left(e^{j\omega T} \right)$ characterizes a linear time-invariant system by examining it's output to sinusoidal inputs. The array pattern is used in a similar way in array signal processing. Through the quantity $W\left(\omega^0 \vec{\alpha} - \vec{k}^0 \right)$ in (4.3), it determines the *amplitude* and *phase* of the beamformed monochromatic plane wave signal impinging on the array form different directions expressed in $\vec{\alpha}$. Thus the array pattern (4.4) determines the array's directivity characteristics.

The resemblance between the wavenumber array pattern in (4.4) and the window functions in digital signal processing should also be emphasized. The array element weights w_m are equal to a window's filter taps. If the array transducer elements are placed regularly, we may use the familiar window function analysis directly to characterize the array's directivity. Familiar

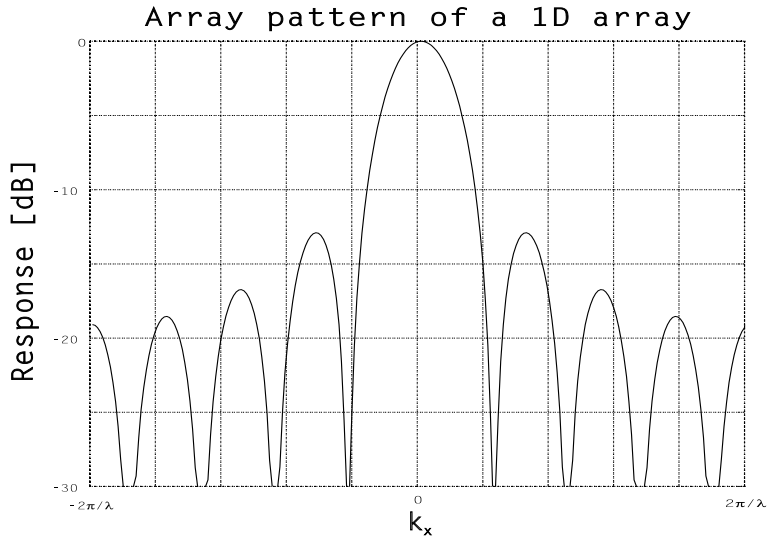

Figure 4.1: The wavenumber array pattern for a 9 element regularly $\lambda/2$-spaced linear array. The array response is plotted against the wavenumber component k_x.

windows such as the *Rectangular*, *Hamming* or *Dolph-Chebychev* windows may thus be suggested as reasonable array element weightings.

A typical array pattern is shown in figure 4.1. The magnitude of the absolute array pattern $\left|W\left(\vec{k}\right)\right|$ is plotted against the wavenumber component k_x. According to the spatial sampling theorem (3.10) the spatial distance d between the array elements is $d = \lambda/2$.

4.3 The angular array pattern

In our application, the *angular array pattern* represents a more practical definition than the previously defined wavenumber array pattern given in (4.4). It gives the array's response to waves from different spherical directions explicitly expressed with the angles ϕ and θ. The geometric interpretation of the angular array pattern is evident in figure 4.2, where a plane wave soon will impinge on the array.

To deduce the angular array pattern, it is convenient to introduce the unit direction vector in spherical coordinates $\vec{s}_{\phi,\theta} \in \mathbb{R}^3$ as

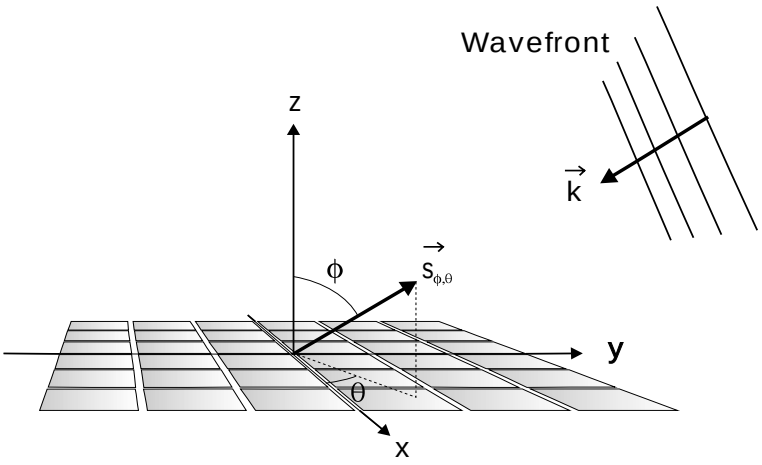

Figure 4.2: A planar 2D array transducer in a rectangular coordinate system is shown. The wavefront is described by the wavenumber vector $\vec{k}$. The array looks in the direction given by the unit direction vector $\vec{s}_{\phi,\theta}$ in spherical coordinates.

$$\vec{s}_{\phi,\theta} = -\frac{\vec{k}}{\left|\vec{k}\right|} \tag{4.5}$$

where the length of the wavenumber vector is taken to $\left|\vec{k}\right| = \frac{2\pi}{\lambda}$ for a fixed wavelength λ. The unit direction vector $\vec{s}_{\phi,\theta}$ may be interpreted as the direction in which the array looks[2]. Substituting this for $\vec{k}$ in (4.4) we obtain the general angular array pattern $W(\phi,\theta) \in \mathbb{C}$ as

$$W(\phi,\theta) = \sum_{m=1}^{M} w_m \exp\left\{-j\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot\vec{x}_m\right\} \tag{4.6}$$

The unit direction vector $\vec{s}_{\phi,\theta}$ may be expressed in rectangular coordinates as $\vec{s}_{\phi,\theta} = (\sin\phi\cos\theta, \sin\phi\sin\theta, \cos\phi)$. Let the m-th array element be located at $\vec{x}_m = (x_m, y_m, z_m)$ in space. The dot product $\vec{s}_{\phi,\theta}\cdot\vec{x}_m$ is then

$$\vec{s}_{\phi,\theta}\cdot\vec{x}_m = (\sin\phi\cos\theta)\ x_m + (\sin\phi\sin\theta)\ y_m + (\cos\phi)\ z_m \tag{4.7}$$

[2]Note the difference between the *steering* and *looking* direction. Steering is used when the delays are adjusted to give *maximum* response in a fixed direction. On the other hand, the looking direction is any possible direction within the array's response field.

which is valid for any array configuration including curved 2D arrays[3].

For simplicity, the angular array pattern (4.6) may also be written in matrix notation. We can write it as the matrix inner product

$$W(\phi,\theta) = \mathbf{w}^T \mathbf{v}(\phi,\theta) = \mathbf{v}(\phi,\theta)^T \mathbf{w} \tag{4.8}$$

where $\mathbf{w} = \begin{bmatrix} w_1 & \cdots & w_M \end{bmatrix}^T$ are the element weights and the kernel vector is $\mathbf{v}(\phi,\theta) = \begin{bmatrix} \exp\left\{-j\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot\vec{x}_1\right\} & \cdots & \exp\left\{-j\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot\vec{x}_M\right\} \end{bmatrix}^T$.

4.3.1 Spherical mapping to the $\phi\theta$-plane

When introducing the angular array pattern, the direction of the wavenumber vector $\vec{k}$ is mapped to the $\phi\theta$-plane through the unit direction vector $\vec{s}_{\phi,\theta}$. It is important to understand the underlying geometry of this nonlinear mapping, especially in the angular array pattern analysis.

The rectangular components of the wavenumber vector $\vec{k}$ is written as $\vec{k} = (k_x, k_y, k_z)$. Using the identity in (4.5) we can write the rectangular coordinate components as

$$\begin{aligned} \vec{k} &= -\tfrac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta} \\ &\Downarrow \\ (k_x, k_y, k_z) &= -\tfrac{2\pi}{\lambda}\,(\sin\phi\cos\theta, \sin\phi\sin\theta, \cos\phi) \end{aligned} \tag{4.9}$$

The last equality in (4.9) is the key in understanding the difference between the wavenumber and the angular array pattern. The unit direction vector $\vec{s}_{\phi,\theta}$ introduces a nonlinear argument to the complex exponential in the angular array pattern calculation. As a consequence of this, the wavenumber array pattern for a regularly spaced array has regular distance between the peaks, but the angular array pattern has not.

The spherical mapping to the $\phi\theta$-plane is shown in figure 4.3. The nonlinearity becomes evident in the shape of each patch. The rectangular shaped patches in the $\phi\theta$-plane corresponds to slightly curved patches on the sphere. This ambiguity is important to be aware of in the $\phi\theta$-plane analysis of the angular array pattern. Equidistant gridpoints in the $\phi\theta$-plane are for instance not equidistant when mapped to the sphere.

Another ambiguity with the $\phi\theta$-plane mapping is the possible introduction of an *invisible region* as described in [21]. With a linear array, this occurs

[3] A curved 2D array may also be classified as a 3D array.

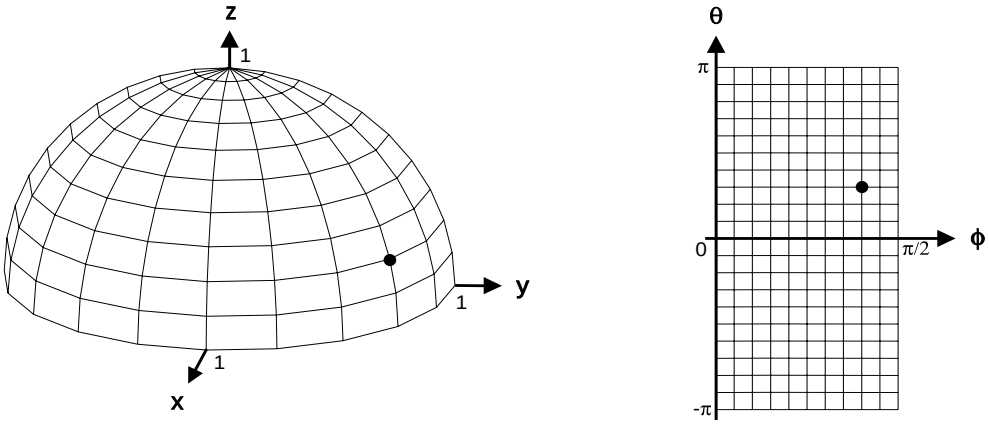

Figure 4.3: The nonlinear spherical mapping from a point on the unit sphere into the $\phi\theta$-plane is shown. Each point on the sphere corresponds to a direction in which the array looks.

when the interelement spacing is $d > \lambda/2$. Due to the spatial sampling theorem (3.10) the array is undersampled and the array pattern may be calculated for $k_x > 2\pi/\lambda$. From (4.9) this does not correspond to a *real* direction, or equally a point in the $\phi\theta$-plane, since $|\sin\phi\cos\theta| \leq 1 \Rightarrow |k_x| \leq 2\pi/\lambda$. Although the beamformer assumes energy in this region, it is invisible in the angular array pattern.

4.3.2 The mainlobe

According to the array pattern in figure 4.1, the highest peak is the *mainlobe* while the smaller peaks are *sidelobes*. The array pattern may be interpreted as the spatial filter response of an array. Thus the mainlobe is similar to the *passband* in a spatial bandpass filter, which only passes signals in these directions.[4]

The location of the passband as spherical directions and the $\phi\theta$-plane mapping of the mainlobe is shown in figure 4.4.

[4]In filter design, a bandpass filter's frequency response is often divided into pass, transition and a stopbands. In our case, the mainlobe peak would correspond to the passband while the rest of the mainlobe would be the transition band. To simlify notation, the passband will here include the transition band.

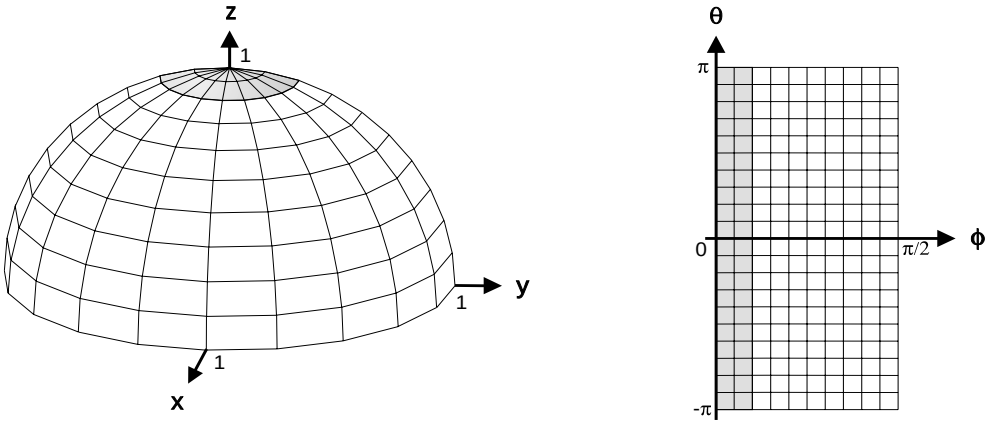

Figure 4.4: The passband of the spatial filter is the shaded region of the unit sphere. Thus the mainlobe in the angular array pattern corresponds to the shaded area in the $\phi\theta$-plane.

Based on the analogy of flashlight, the cone located at the origin enveloping the shaded passband area on the unit sphere, is sometimes called a *beam*. Similar to filter design in classical digital signal processing, we would like the beam to approach the *delta pulse* or equally an infinitely thin beam. But from array signal processing theory, this is impossible using an array with finite spatial extension.

The location of the mainlobe peak tells in which direction we get maximum response with the array. Generally, the location of the mainlobe peak of the angular array pattern $W(\phi, \theta)$ may be found as

$$\max_{\phi,\theta} |W(\phi,\theta)| = \max_{\phi,\theta} \left| \sum_{m=1}^{M} w_m \exp\left\{ -j \frac{2\pi}{\lambda} \vec{s}_{\phi,\theta} \cdot \vec{x}_m \right\} \right|$$

Another measure used to characterize the mainlobe is the *mainlobe width* or equally the *beamwidth*. Here we define it to be the full width of the mainlobe at 6 dB below the mainlobe peak on the array pattern. This corresponds to the conventional FWHM (full width at half maximum) measure in digital signal processing [13].

From the angular array pattern, we can measure at which angle ϕ^* the mainlobe has dropped 6 dB. The beamwidth is then $2\,\phi^*$ for consistency and usually measured in degrees.

4.3.3 Sidelobes

The sidelobes in the array pattern is equal to the *stopband* in a bandpass filter. As is well know from window filter design, the sidelobes can not be completely rejected using a finite aperture. But the sidelobes can be suppressed to a certain degree by adjusting the amplitude weights and element positions cleverly.

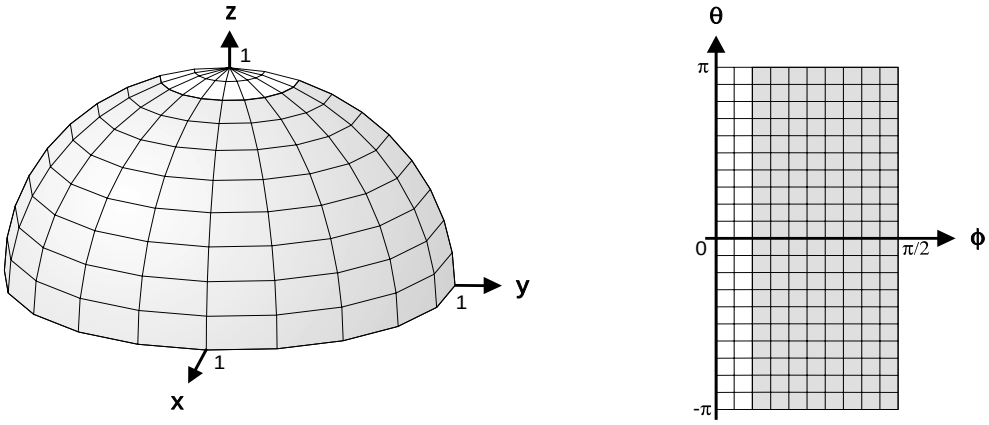

Figure 4.5: The stopband of the spatial filter is the shaded region of the unit sphere. The sidelobes are consequently located to the shaded area in the $\phi\theta$-plane.

The *sidelobe region* or equally the stopband, is conveniently defined as the area in the $\phi\theta$-plane outside the first zero crossing of the mainlobe. This is shown in figure 4.5. The *sidelobe level* is used as a measure on the height of the highest sidelobe peak in the sidelobe region and usually given in dB.

4.3.4 Energy

For practical applications, the energy present in the array pattern is a valuable measure, since it has a physical interpretation. The energy is calculated from the squared angular array pattern as

$$E = \iint\limits_{R} |W(\phi, \theta)|^2 \; d\phi \; d\theta$$

where R is the actual region in the $\phi\theta$-plane.

The total array pattern energy E_t is calculated by integrating over the region R in the $\phi\theta$-plane corresponding to the half-sphere of actual directions. The sidelobe energy E_s and the mainlobe energy E_m may also be calculated separately. As the mainlobe region and the sidelobe region together spans all directions on the half-sphere, we have in general $E_t = E_m + E_s$.

The angular array pattern $W(\phi, \theta)$ is in general a complex number. Introducing the matrix notation of the angular array pattern in (4.8) the energy may be written as[5]

$$\begin{aligned} E &= \iint_R W(\phi,\theta)\ W(\phi,\theta)^*\ d\phi\ d\theta \\ &= \iint_R \mathbf{w}^T\, \mathbf{v}(\phi,\theta)\ \left[\mathbf{v}(\phi,\theta)^T\ \mathbf{w}\right]^*\ d\phi\ d\theta \\ &= \mathbf{w}^T \left[\iint_R \mathbf{v}(\phi,\theta)\ \mathbf{v}(\phi,\theta)'\ d\phi\ d\theta\right] \mathbf{w} \\ &= \mathbf{w}^T\, \mathbf{Q}_R\, \mathbf{w} \end{aligned} \tag{4.10}$$

where $\mathbf{w}$ are the real amplitude weights and $\mathbf{v}(\phi, \theta)$ is the angular array pattern kernel. The quadratic $M \times M$ matrix $\mathbf{Q}_R = \iint_R \mathbf{v}(\phi,\theta)\ \mathbf{v}(\phi,\theta)'\ d\phi\ d\theta$. By evaluating the matrix outer product, we can write

$$\mathbf{Q}_R = \iint_R \begin{bmatrix} 1 & e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_2-\vec{x}_1)} & \dots & e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_M-\vec{x}_1)} \\ e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_1-\vec{x}_2)} & 1 & \dots & e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_M-\vec{x}_2)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_1-\vec{x}_M)} & e^{j\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot(\vec{x}_2-\vec{x}_M)} & \dots & 1 \end{bmatrix} d\phi\, d\theta$$

which may be calculated by evaluating the matrix integral for each matrix entry. The ij-th element of $\mathbf{Q}_R$ is then

$$\mathbf{Q}_R(i,j) = \iint_R \exp\left\{j\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot(\vec{x}_j - \vec{x}_i)\right\} d\phi\, d\theta$$

Since $W(\phi, \theta)$ is the Fourier transform of the weights, we might use the *Parseval relation* [35] directly for the energy calculation. But this would restrict us to regularly $\lambda/2$-spaced arrays. Anyway, equation (4.10) may be

[5] $\mathbf{A}^*$ is the conjugate of $\mathbf{A}$ while $\mathbf{A}'$ is the conjugate transpose. This notation will be used throughout the thesis.

interpreted as a scaled version of the Parseval relation where the scaling factor is the quadratic $\mathbf{Q}_R$ matrix.

Another alternative to calculating the $\mathbf{Q}_R$ matrix, is to approximate the energy in the actual region R in the $\phi\theta$-plane by numerical integration. As $|W(\phi,\theta)|^2$ is a surface defined on the $\phi\theta$-plane, we may calculate the energy by approximating the volume under this surface.

4.3.5 Symmetric linear and planar arrays

Consider now the special case with *symmetric linear* and *planar* arrays. In addition to ensure a *real* array pattern, the calculation of the array pattern is simplified. This is useful when later optimizing and simulating the response from considerably large arrays.

A linear 1D array has its elements $\vec{x}_n$ located on the x-axis, while the elements are located in the xy-plane for the 2D planar array. Let the number of elements M be *even* with $M = 2N$. If we constrain the elements to appear in symmetric pairs such that

$$\left.\begin{array}{rcl} \vec{x}_{N+n} & = & -\vec{x}_n \\ w_{N+n} & = & w_n \end{array}\right\} \quad n = 1, \cdots, N \tag{4.11}$$

we are ensured to have a real array pattern[6]. Using the trigonometric identity $2\cos(x) = e^{jx} + e^{-jx}$ and combining (4.11) with (4.6), we get the angular array pattern

$$W(\phi,\theta) = 2\sum_{n=1}^{N} w_n \cos\left(\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot\vec{x}_n\right) \tag{4.12}$$

which indeed is real for real weights $w_n \in \mathbb{R}^N$. The dot product $\vec{s}_{\phi,\theta}\cdot\vec{x}_n$ in (4.7) also simplifies in this case, since the z-component of each array element location is zero, and we have

$$\vec{s}_{\phi,\theta}\cdot\vec{x}_n = \sin\phi\ \ (x_n\cos\theta + y_n\sin\theta) \tag{4.13}$$

Note that this simplification is restricted to hold for linear and planar arrays. A concave or convex 2D curved array will for instance violate the element location symmetry in (4.11).

[6] An array with an *odd* number of elements may be represented with the center element as two distinct co-located elements.

4.4 The beampattern

When calculating and analyzing the array pattern, the influence of the element delays Δ_m is neglected. To analyze the total result of applying both amplitude weighting and temporal delaying to each element, the *beampattern* may be examined. It gives the array response to signals from different directions when the array is phase-steered in a direction opposite of the slowness vector $\vec{\alpha}$. The required delays Δ_m for steering are given in (4.2).

The beampattern is calculated through the wavenumber array pattern (4.4) and is written as

$$W\left(\omega^0\vec{\alpha} - \vec{k}^0\right) = \sum_{m=1}^{M} w_m \exp\left\{j\ \left(\omega^0\vec{\alpha} - \vec{k}^0\right) \cdot \vec{x}_m\right\}$$

where $\vec{\alpha}$ is fixed and expresses the steered direction. The wavenumber vector $\vec{k}^0 = \omega^0\vec{\alpha}^0 \in \mathbb{R}^3$ describes the wavefield acting on the array.

Note that mathematically, the beampattern is just a scaled and translated version of the wavenumber array pattern. This illustrates that the array pattern becomes the primary quantity used to evaluate array and algorithm designs.

4.5 Image quality

The ability to make high quality images depends on how accurate we can measure the field. The accuracy is closely related to the beamwidth of the array pattern, which gives the lateral image *resolution*. The resolution is taken to be the inverse of the beamwidth and tells how precise an array can localize a given source.

We would like to have as good resolution as possible, which is obtained with the smallest possible beamwidth. But as previously pointed out there is a trade-off between the beamwidth and the sidelobe level. When the beamwidth is decreased by element weighting, the sidelobe level is increased and vice versa.

The sidelobe level and especially the sidelobe energy in the array pattern affect the *contrast* in the image [1]. A high amount of sidelobe energy describes significant leakage of energy from bright into dark areas. To increase the contrast in the image, the sidelobe energy should be suppressed.

There are also other factors influencing the image quality such as aperture size, f-number[7] and the central frequency of the beam as well as the frame rate for real-time imaging. But these are already set for certain applications and thus beyond the scope of element weighting and delaying.

[7]The ratio between the focal length and the aperture diameter.

Chapter 5

Optimization

Mathematical optimization is an area of mathematics which studies ways of finding the *extremal* values of a function subject to stated constraints. Today, optimization problems arise in all sorts of areas where there are several possible, or *feasible*, solutions to a problem. Mathematical optimization, or *mathematical programming,* introduces a mathematical formalism to the problem. Naturally, we seek an *optimal* of all the feasible solution due to a mathematical criterion.

Mathematical programming is a fast growing branch of mathematics, with most of its development dated to the second half of this century. The rapid growth in mathematical programming is mainly caused by faster computers, efficient algorithms and a large amount of applications. Today, mathematical optimization may be divided in several fields, as *linear programming, nonlinear programming, discrete optimization* and *stochastic optimization.*

This chapter introduces *linear programming* and *integer linear programming.* Methods to solve such programs include the *Simplex* and the *Branch-and-bound* method. *Quadratic programming* is also considered. The main principles of the methods are introduced together with the necessary mathematical notation. Some theory of *linear systems* and *polyhedra* will be given as well.

The intention of this chapter, is to establish a framework to fit optimization problems which arise in beamforming. It is not meant to go very deep into the mathematical theory, but more to clarify the ideas. Only the most important results are included. A more thorough investigation of mathematical optimization is left to the many textbooks on the subject, as for instance [5, 6, 31, 32]

5.1 Mathematical programming

A mathematical program is an optimization problem formulated with mathematical functions. The problem is to *maximize* or *minimize* a specific quantity, mathematically formulated in the *objective function*, which depends on a finite number of variables. The variables may be independent of each other, or they may be related through one or more *constraints.* In mathematical programming, the constraints can be either *equality constraints* or *inequality constraints*. Especially the inequality constraints provide great flexibility in establishing mathematical models.

Let $\mathbf{x} \in \mathbb{R}^N$ be a column vector consisting of n real variables involved in the optimization problem. Hence, a mathematical program has the following general form

$$\begin{array}{lll} \text{Optimize} & z = f(\mathbf{x}) & \\ \text{Subject to} & & \\ & \left.\begin{array}{c} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_M(\mathbf{x}) \end{array}\right\} \begin{array}{c} \geq \\ = \\ \leq \end{array} \left\{\begin{array}{c} b_1 \\ b_2 \\ \vdots \\ b_M \end{array}\right. & \end{array} \tag{5.1}$$

where f is the objective function and $g_1, g_2, \ldots, g_M$ represents the constraints on the variable vector $\mathbf{x}$. In addition, any of the variables may be constrained to take on *integer* values.

A method to solve the general mathematical program in (5.1), strongly depends on the functions involved. The functions may be any combination of *linear/nonlinear*, *continuous/discrete* and *deterministic/stochastic.* Obviously, the large variety of function classes demand different optimization methods.

5.2 Linear systems

All the mathematical programs considered in this text have *linear constraints.* The constraints may be *equality* as well as *inequality* constraints. Together, they form a *linear system*. A linear system written in matrix notation may have the following general form

$$\mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{5.2}$$

where $\mathbf{x} \in \mathbb{R}^N$ are the real variables. The constraints on the variables are given by the $M \times N$ constraint matrix $\mathbf{A} \in \mathbb{R}^{M,N}$ and the real column vector $\mathbf{b} \in \mathbb{R}^M$. Note that any linear equality $\mathbf{a}^T\mathbf{x} = b$ may be written as two linear inequalities $\mathbf{a}^T\mathbf{x} \leq b$ and $\mathbf{a}^T\mathbf{x} \geq b$, where $\mathbf{a}^T$ may be any row in $\mathbf{A}$ with b as the corresponding element in $\mathbf{b}$. Consequently, the linear system in (5.2) is indeed a general form, as it includes both linear equality and inequality constraints.

5.2.1 Transformations

Linear systems may be represented in many different ways. The representation in (5.2) is only one. Another convenient representation of a general linear system is

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned} \tag{5.3}$$

They are both *equal* in the sense that any linear system represented in one way may also be represented the other way by suitable transformations, as long as the *solution set* remains the same.

As already mentioned, an equation may be replaced by two inequalities. There are also other transformation techniques. We may for instance transform inequality constraints to equality constraints. Any linear inequality constraint in (5.2) with the original form $\mathbf{a}^T\mathbf{x} \leq b$ may be transformed to the form in (5.3) by adding a *slack* variable $s^- \geq 0$ such that $\mathbf{a}^T\mathbf{x} + s^- = b$.

Similarly, a *surplus* variable $s^+ \geq 0$ transforms constraints with the original form $\mathbf{a}^T\mathbf{x} \geq b$ to equalities with the form $\mathbf{a}^T\mathbf{x} - s^+ = b$. Since both the slack and surplus variables are positive, they may be directly augmented to the variable space in the linear system in (5.3).

Note that all the variables in (5.2) are free while the variables in (5.3) are bounded. A *free* real variable, $x \in \mathbb{R}$, may be represented as two nonnegative bounded variables $x_1, x_2 \in \mathbb{R}^+$ as $x = x_1 - x_2$.

5.2.2 Polyhedra

The *solution set* of such a linear system is of special interest, since its points correspond to the *feasible* solutions of the mathematical program. The solution set also define objects called *polyhedra*. A *polyhedron* $P \subseteq \mathbb{R}^N$ is

given as $P = \left\{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\right\}$, which is the solution set to the linear system $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Furthermore, the solution set of each single inequality in the linear system is called a *half-space*, and given as $H_{\leq}(\mathbf{a}_m, b_m) = \left\{\mathbf{x} \in \mathbb{R}^N \,\middle|\, \mathbf{a}_m^T\mathbf{x} \leq b_m\right\}$, where $\mathbf{a}_m^T$ is the m-th row in $\mathbf{A}$ and b_m is the m-th element in $\mathbf{b}$. The polyhedron, which is the solution set of the entire linear system, is thus the intersection of all the half-spaces $P = \cap H_{\leq}(\mathbf{a}_m, b_m)$.

Consider now a linear system with only two variables $\mathbf{x} \in \mathbb{R}^2$. The variable vector $\mathbf{x}$ may then be represented as points in $\mathbb{R}^2$. An example of a polyhedron and its half-spaces is shown graphically in figure 5.1. In this case, the polyhedron is generated by a linear system with 4 inequalities, represented with $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ and $\mathbf{a}_4$. The vector $\mathbf{a}_m$ in each inequality may be interpreted as the *normal* vector to the corresponding half-space.

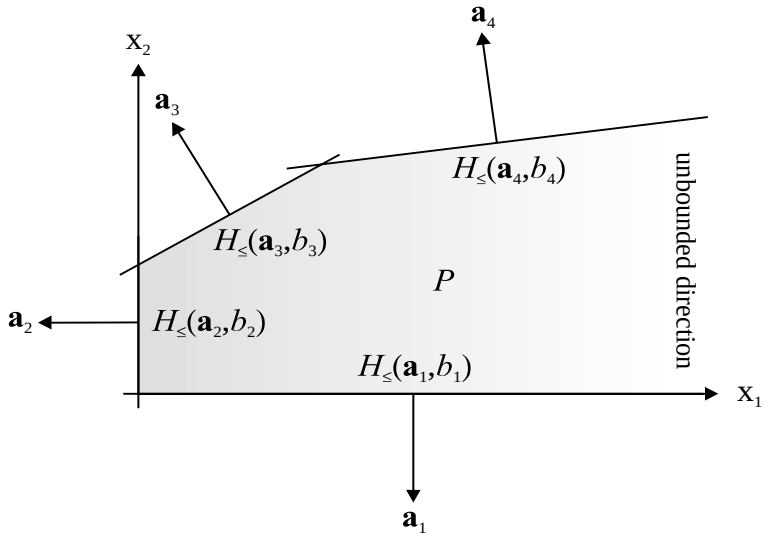

Figure 5.1: A graphical representation of a polyhedron P as a solution set of a linear system $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ in $\mathbb{R}^2$.

A polyhedron $P = \left\{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\right\}$ has some important properties, which affect the optimal solutions of mathematical programs with linear constraints.

1. Each polyhedron is a *closed convex set*. That is, if two points belong to the polyhedron, then *each* point on a line segment between the two points belongs to the polyhedron too. Furthermore, an *extreme point*

of a closed convex set *cannot* be expressed as a *convex combination*[1] of two other points in the set. Thus, the extreme points of the polyhedron P, called *vertices,* are point sets on the *boundary* of P.

2. A polyhedron is generated by a *finite* number of inequalities, each corresponding to a half-space $H_{\leq} \in \mathbb{R}^N$. Some of the inequalities in the linear system may be *redundant*, that is, the solution set is not altered by removing them. By removing the redundant inequalities, the remaining inequalities are *supporting*. Hence, *any* vertex of P may be obtained by setting one or more of the supporting *inequality* constraints to *equality*.

3. The point sets on the boundary of P are also called the *faces* of P. A *nonempty* set F is a face of P *if and only if* $F = \{\mathbf{x} \in P \mid \mathbf{A}^s\mathbf{x} = \mathbf{b}^s\}$ where $\mathbf{A}^s\mathbf{x} \leq \mathbf{b}^s$ is a subsystem of $\mathbf{Ax} \leq \mathbf{b}$. Furthermore, if $\mathbf{A}^s$ has n linearly independent rows, then the face F is a *vertex*. A vertex is also called a *minimal face* of P. As the polyhedron is finitely generated, P has a *finite* number of faces. Each *face* is too a *polyhedron* as convex sets intersect in convex sets.

4. Any polyhedron may be either *bounded* or *unbounded*. Figure 5.1 shows an *unbounded* polyhedron.

5.3 Linear Programming

Linear programming (LP) arise in many contexts. It is also one of the most common optimization methods. It provides an effective way to maximize or minimize a *linear* function of several variables subject to *linear* constraints on those variables.

Much of the popularity of LP is due to the *simplex algorithm*, which effectively solves *linear programs* numerically. The *simplex method*, which origins from some papers by the physicist Joseph Fourier about 1820, was fully developed into an efficient *algorithm* by George Dantzig about 1940. Efficient implementations of the Simplex algorithm are still among the fastest LP codes available.

[1] A point $\mathbf{x} \in \mathbb{R}^N$ is a convex combination of M points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M \in \mathbb{R}^N$ if there exist constants $\alpha_1, \alpha_2, \ldots, \alpha_M \geq 0$ with $\sum \alpha_m = 1$ such that $\mathbf{x} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \cdots + \alpha_M\mathbf{x}_M$.

5.3.1 A linear program

Let $\mathbf{c} \in \mathbb{R}^N$ be column vector with real constants and let $\mathbf{x} \in \mathbb{R}^N$ denote the variables in the optimization problem. Furthermore, let $\mathbf{A} \in \mathbb{R}^{M,N}$ and $\mathbf{b} \in \mathbb{R}^M$ represent the linear system with M constraints in N variables. *Any* linear program can then be written in the matrix *standard form*

$$\begin{array}{lrcl} \text{Minimize} & z & = & \mathbf{c}^T\mathbf{x} \\ \text{Subject to} & & & \\ & \mathbf{Ax} & = & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \tag{5.4}$$

by convenient transformations. The function $z = \mathbf{c}^T\mathbf{x}$ is called the *objective* or the *cost* function. Note that a maximization problem may be obtained by minimizing $-z$.

5.3.2 The simplex method

Based on the s*implex method,* linear programs in the standard form (5.4) are solved conveniently. In this text, the main principles of the simplex method will be presented with emphasis on the *underlying geometry.* With the theory of *polyhedra* in mind, this will hopefully contribute to the understanding of the simplex method as well as the numerical *simplex algorithm.*

An optimal solution

Recall the polyhedral theory for a moment. Assume that the constraints in (5.4) correspond to a *bounded* polyhedron $P \in \mathbb{R}^N$. Furthermore, the objective function in (5.4) defines a *hyperplane* $H_=(\mathbf{c},z) = \left\{\mathbf{x} \in \mathbb{R}^N \,\middle|\, \mathbf{c}^T\mathbf{x} = z\right\}$. A hyperplane is also a *polyhedron*, where $\mathbf{c}$ may be interpreted as the *normal* vector to the plane and z determines its *location* in $\mathbb{R}^N$. From LP theory, there is always an *optimal vertex* solution to such a linear program. Put another way, the *optimal value* z^* of (5.4) is attained when the hyperplane $H_=(\mathbf{c},z)$ intersects the polyhedron P in a *vertex.* This situation is illustrated in figure 5.2, where the hyperplane defines a line in $\mathbb{R}^2$.

From the geometry in figure 5.2, there may obviously exist more than one optimal solution to a linear program, depending on the direction of the hyperplane $H_=(\mathbf{c},z)$. If a number of *equally* optimal solutions exist, *any* one will do.

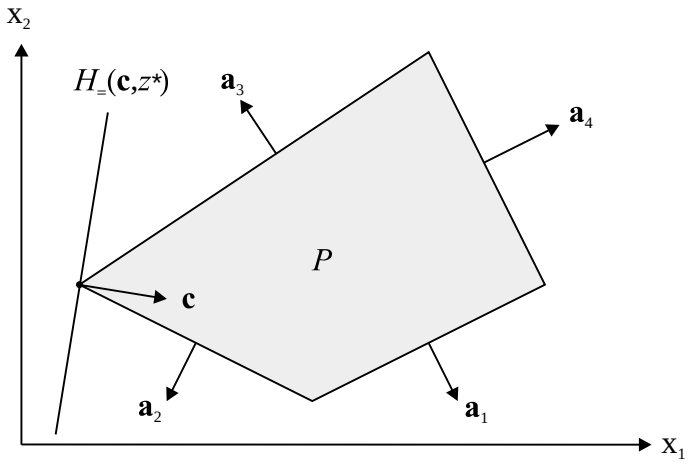

Figure 5.2: The optimal value z^* of the linear program, is attained as the hyperplane $H_=(\mathbf{c},z)$ intersects P in a vertex.

If the polyhedron P defined by the constraints in (5.4) is *unbounded*, we are *not* assured to get a *finite* optimal solution to the linear program, as the hyperplane $H_=(\mathbf{c},z)$ may disappear in the unbounded direction.

The geometric idea

It is now stated that the optimal solution to a linear program coincides with a vertex of the polyhedron P, defined by the constraints. Based on this, the simplex method solves a linear program in two stages. The geometric idea may be given as

1. Find an *initial vertex* of P corresponding to the feasible solutions of the problem[2].
2. Starting from this initial vertex, the vertex corresponding to the optimal solution z^* is found by examining *adjacent* vertices of P.

The process of examining adjacent vertices of P is precisely the underlying geometric idea of the numerical simplex algorithm.

[2]If the polyhedron has no vertices, the linear program is either *infeasible* or the optimal value is *unbounded*.

5.3.3 The simplex algorithm

The principle of the numerical *simplex algorithm*, is to maintain two sets of the constraints in (5.4). The set of *active* constraints, corresponding to a vertex of the polyhedron $P = \left\{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{A}\mathbf{x} = \mathbf{b},\, \mathbf{x} \geq \mathbf{0}\right\}$ is denoted B and called a *basis index set*. Thus, B contains N column indices of $\mathbf{A}$. All the other constraints are contained in the set N, called a *nonbasis index set*. To move from one vertex to an adjacent vertex of P, is algebraically attained by replacing one constraint in B with one constraint from the set N.

A basic feasible solution

A *basic feasible solution* of the linear program in (5.4) is the algebraic counterpart to an *initial vertex* of the polyhedron P. An algebraic method to find a basic feasible solution will now be given.

Note that *any* linear $M \times N$ system in matrix notation, $\mathbf{A}\mathbf{x} = \mathbf{b}$, may be rewritten as $x_1\mathbf{a}^1 + x_2\mathbf{a}^2 + \cdots + x_N\mathbf{a}^N = \mathbf{b}$ where $\mathbf{a}^1, \mathbf{a}^2, \ldots, \mathbf{a}^N \in \mathbb{R}^M$ are the *column* vectors in $\mathbf{A} \in \mathbb{R}^{M,N}$. Assume that $\mathbf{A}$ has *full row rank*[3] which implies that $M \leq N$. Hence, *at least* one collection of M column vectors in $\mathbf{A}$ are linearly independent.

Now, rearrange the columns in $\mathbf{A}$ using the index sets. Let $\mathbf{A}_B \in \mathbb{R}^{M,M}$ be a matrix containing M linearly independent column vectors in $\mathbf{A}$ from the basis index set B and let $\mathbf{A}_N \in \mathbb{R}^{M,N-M}$ contain the remaining column vectors. Furthermore, the variables corresponding to the indices B are called *basic variables* and denoted $\mathbf{x}_B$ while the remaining variables $\mathbf{x}_N$ are called *nonbasic variables*. With this rearrangement, the constraints in (5.4) may be replaced by the equal linear system

$$\begin{bmatrix} \mathbf{A}_B & \mathbf{A}_N \end{bmatrix} \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix} = \mathbf{b}\,, \quad \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{bmatrix} \geq \mathbf{0} \tag{5.5}$$

Since the columns in $\mathbf{A}_B$ are linearly independent by the construction above, $\mathbf{A}_B$ is *invertible* and the linear system $\mathbf{A}_B\mathbf{x}_B = \mathbf{b}$ has precisely one solution, $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$. Whenever $\mathbf{x}_B \geq 0$, we may obtain a *basic feasible solution* to (5.4) by setting the nonbasic variables to zero, $\mathbf{x}_N = \mathbf{0}$. The point

[3] A $M \times N$ matrix $\mathbf{A} \in \mathbb{R}^{M,N}$ has full row rank if the M row vectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_M \in \mathbb{R}^M$ are *linearly indepenent*. That is, the only solution to $\alpha_1\mathbf{a}_1 + \alpha_2\mathbf{a}_2 + \cdots + \alpha_M\mathbf{a}_M = \mathbf{0}$ is $\alpha_1 = 0, \alpha_2 = 0, \ldots, \alpha_M = 0$.

$\mathbf{x} \in \mathbb{R}^N$ given by $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$ and $\mathbf{x}_N = \mathbf{0}$ then solves the linear constraints in (5.4) as much as it corresponds to an initial *vertex* of the polyhedron.

Duality

The simplex algorithm also takes advantage of *duality*. That is, *every* linear program has an associated linear program, called the *dual* program. The original linear program is in this context called the *primal* program. As an example, the dual to the linear program in (5.4) is written as

$$\begin{array}{lrcl} \text{Maximize} & z & = & \mathbf{b}^T\mathbf{y} \\ \text{Subject to} & & & \\ & \mathbf{A}^T\mathbf{y} & \leq & \mathbf{c} \end{array} \tag{5.6}$$

where $\mathbf{A} \in \mathbb{R}^{M,N}$, $\mathbf{b} \in \mathbb{R}^M$ and $\mathbf{c} \in \mathbb{R}^N$ are identic in both programs. Note that the dual is a maximization problem while the primal is a minimization problem. The *dual* variables $\mathbf{y} \in \mathbb{R}^M$ are sometimes called *shadow costs*.

The form of the dual problem depends on the way the primal problem is presented. Still, the *duality theorem* is valid to any primal/dual pair of linear programs.

Theorem 1 (Duality theorem) *If a feasible solution exists to either the primal or the dual linear program, then the other program also has an optimal solution and the optimal values of the two linear programs coincide.*

Consider a feasible solution of the linear system in (5.5) which is also a basic feasible solution to the linear program in (5.4). The duality theorem effectively states whether this solution is *optimal* or not.

In accordance with the linear system in (5.5), rearrange the vector $\mathbf{c}$ such that the *value* of the primal problem $z = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N$. If the basic feasible solution with $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$ and $\mathbf{x}_N = \mathbf{0}$ is *optimal*, then the objective value in the primal and the dual program coincides. That is, $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y} \Rightarrow \mathbf{c}_B^T\mathbf{A}_B^{-1}\mathbf{b} = \mathbf{b}^T\mathbf{y}_B$ where $\mathbf{y}_B$ is called a *dual basic solution*. We may also write $\mathbf{y}_B^T = \mathbf{c}_B^T\mathbf{A}_B^{-1}$ since $\mathbf{b}^T\mathbf{y}_B = \mathbf{y}_B^T\mathbf{b}$.

Furthermore, $\mathbf{y}_B$ is a *dual basic feasible solution* if and only if it satisfies the dual constraints $\mathbf{A}^T\mathbf{y}_B \leq \mathbf{c}$. By rearranging $\mathbf{A}$, we can write the dual constraints in the transpose form as $\mathbf{y}_B^T \begin{bmatrix} \mathbf{A}_B & \mathbf{A}_N \end{bmatrix} \leq \mathbf{c}^T$. Substituting

$\mathbf{y}_B^T = \mathbf{c}_B^T \mathbf{A}_B^{-1}$, we can write the dual constraints as

$$\begin{array}{rcl} \left[\begin{array}{cc} \mathbf{c}_B^T & \mathbf{y}_B^T \mathbf{A}_N \end{array} \right] & \leq & \left[\begin{array}{cc} \mathbf{c}_B^T & \mathbf{c}_N^T \end{array} \right] \\ & \Updownarrow & \\ \mathbf{y}_B^T \mathbf{A}_N & \leq & \mathbf{c}_N^T \end{array} \tag{5.7}$$

and $\mathbf{y}_B$ is a dual basic feasible solution if and only if it satisfies the last constraint. Then $\mathbf{x}_B$ and $\mathbf{y}_B$ are both *optimal* solutions to the primal and the dual linear program, respectively.

The two basic solutions $\mathbf{x}_B$ and $\mathbf{y}_B$ are called complementary, since they satisfy the *complementary slackness* condition

$$\left(\mathbf{y}_B^T \mathbf{A} - \mathbf{c}^T\right) \left[\begin{array}{c} \mathbf{x}_B \\ \mathbf{x}_N \end{array} \right] = 0 \tag{5.8}$$

Let both the dual and the primal problem have optimal solutions. Then, if the m-th constraint of one system holds, the m-th component of the optimal solution of its dual is zero.

Using (5.7), *duality* is an effective way to check if the current basic feasible solution is an optimal solution. Furthermore, since the primal and the dual linear program have the same optimal value, the dual may sometimes be solved more effectively than the primal. The primal optimal solution can then be calculated from the dual optimal solution.

The algorithm

Given *a basic feasible solution,* the simplex algorithm then solves the primal linear program in (5.4) numerically. It moves from a *vertex* to an adjacent *vertex* of the underlying polyhedron P as long as the objective function may be improved. The *duality condition* in (5.7) serves as a stopping criterion.

A *vertex* of P is a point $\mathbf{x} \in \mathbb{R}^N$ given by the basic variables $\mathbf{x}_B \geq \mathbf{0}$ and the nonbasic variables $\mathbf{x}_N = \mathbf{0}$, where B and N are the basic and nonbasic index sets, respectively. Any element n in the exclusive basic index sets $n \in B \cup N$ corresponds to a *column* in the constraint matrix $\mathbf{A} \in \mathbb{R}^{M,N}$. To move to an adjacent vertex of P is accomplished by replacing one index element in B with one index element from N. More precisely, the algorithm may take the following form

1. Initialization: Find a basic feasible solution of the primal linear program with $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b} \geq \mathbf{0}$ and $\mathbf{x}_N = \mathbf{0}$. Calculate the corresponding dual basic solution $\mathbf{y}_B^T = \mathbf{c}_B^T \mathbf{A}_B^{-1}$ of the dual program in (5.6)

2. OPTIMALITY CHECK: Calculate $\hat{\mathbf{c}}_N^T = \mathbf{c}_N^T - \mathbf{y}_B^T \mathbf{A}_N$, which is called the reduced cost vector. If all the vector components in $\hat{\mathbf{c}}_N^T \leq \mathbf{0}$ then TERMINATE, since the current basic feasible solution is optimal by (5.7). Else, choose an element $n \in N$ corresponding to a nonnegative element in the reduced cost vector, $\hat{c}_n^T > 0$. The nonbasic variable x_n is the new variable to enter the basis as n enters the basis index set B.

3. PIVOTING: Determine which element $b \in B$ to leave the basis index set by increasing the new basic variable x_n. Let $\hat{\mathbf{b}} = \mathbf{A}_B^{-1}\mathbf{b}$ and $\hat{\mathbf{a}}^n = \mathbf{A}_B^{-1}\mathbf{a}_N^n$ where $\mathbf{a}_N^n$ is the n-th column in $\mathbf{A}_N \in \mathbb{R}^{M,N-M}$. If $\hat{\mathbf{a}}^n \leq \mathbf{0}$ in all components, then TERMINATE since the linear program is unbounded. Else, increase x_n such that $x_n = \min_n \left\{ \frac{\hat{b}_b}{\hat{a}_b^n} \,|b \in B,\ \hat{a}_b^n > 0 \right\}$ where $\hat{b}_b$ and $\hat{a}_b^n$ are the b-th element in $\hat{\mathbf{b}}$ and $\hat{\mathbf{a}}^n$, respectively. This b corresponds to the most restrictive constraint. If the minimization criterion is fulfilled for more than one b, select one.

4. UPDATING: Update the basis index sets, $B = (B \setminus b) \cup n$ and $N = (N \setminus n) \cup b$. Calculate the new basic feasible solution $\mathbf{x}_B = \mathbf{A}_B^{-1}\mathbf{b}$ and the dual basic solution $\mathbf{y}_B^T = \mathbf{c}_B^T \mathbf{A}_B^{-1}$. By construction, $x_n \geq 0$ and $x_b = 0$. Return to step 2.

This algorithm does not obviously lead to a termination, since consecutive pivots with different bases may correspond to the same vertex of the polyhedron. This is called *degeneracy*. But, following *Bland's rule* for selecting the entering and leaving variables, the algorithm will terminate. Bland's rule states that whenever we have more than one entering or leaving variable, always choose the one with *lowest index*.

Much of the success of the simplex algorithm is due to the fact that very little computational effort is required in calculating a new basic feasible solution and checking optimality. Another reason is that linear programming is also shown to have polynomial order[4] using the ellipsoid method.

5.4 Integer programming

Integer programming (IP) is an extension of linear programming. An *integer program* is simply a linear program with the additional constraint that the

[4] A method with k instances is a polynomial method if it is of order $O(k^p)$ for a fixed p.

variables are *integers*. When only some of the variables are integers, the mathematical program is often called a *mixed integer program* (MIP).

The geometric concept of *integral polyhedra* is introduced, as the *vertices* of the integral polyhedron corresponds to solutions of the integer program. The principle of the *branch-and-bound* method to solve such problems is also explained.

5.4.1 An integer program

Similar to the linear program in (5.4), let $\mathbf{c} \in \mathbb{R}^N$ be a column vector with real constants and let $\mathbf{x} \in \mathbb{R}^N$ denote the variables in the optimization problem. Furthermore, let the $M \times N$ system of constraints be represented by $\mathbf{A} \in \mathbb{R}^{M,N}$ and $\mathbf{b} \in \mathbb{R}^M$. An *integer program* in matrix form is then given as

$$\begin{array}{lrcl} \text{Minimize} & z & = & \mathbf{c}^T\mathbf{x} \\ \text{Subject to} & & & \\ & \mathbf{Ax} & = & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \\ & & & \\ & \mathbf{x} & \in & \mathbb{Z}^N \end{array} \tag{5.9}$$

where $\mathbb{Z}^N$ is the set of *integral* numbers in $\mathbb{R}^N$.

Let $P = \left\{\mathbf{x} \in \mathbb{R}^N \mid \mathbf{Ax} = \mathbf{b},\, \mathbf{x} \geq \mathbf{0}\right\}$ be the polyhedron corresponding to the feasible region of the constraints in (5.9) *ignoring* the integer condition. Note that only the *integral* points in P are *feasible* solutions to the linear program in (5.9). Define the *convex hull* C of a set $S \subseteq \mathbb{R}^N$ such that $C = \text{conv}\,(S)$ is the intersection of all convex sets containing S. Furthermore, let P_I denote the *integral polyhedron* $P_I = \text{conv}\left(P \cap \mathbb{Z}^N\right)$. An example of such a polyhedron is shown in figure 5.3. See also page 36 on polyhedra.

The *basic feasible solutions* of the integer program corresponds to the vertices of P_I. Similar to a linear program, the *optimal* solution of the integer program occurs at a vertex of P_I. But, to completely describe P_I is in most cases *extremely* difficult. For some special problems, P and P_I coincide, but this is more the exception than the rule.

One approach to solve an integer program, may be to first solve the problem in (5.9) *ignoring* the integral condition. Then, if the *optimal solution* is *integral* we are fine since the optimal vertex of P happens to be integral. Else, we may iteratively add inequality and/or equality constraints to the

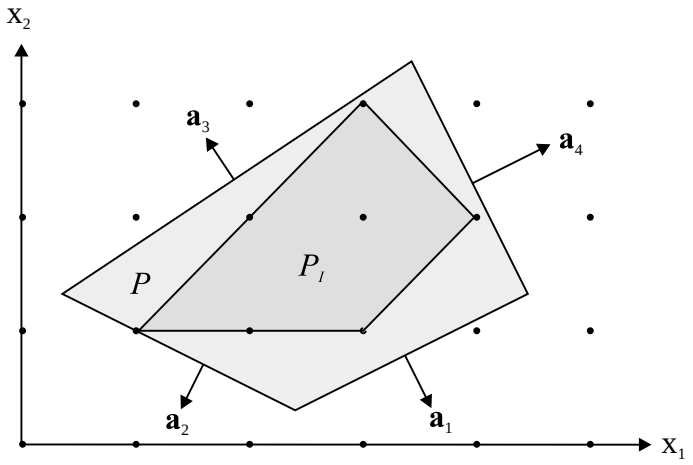

Figure 5.3: A polyhedron P and its integer polyhedron $P_I = \text{conv}\,(P \cap \mathbb{Z}^n)$.

original program in (5.9) which crops the *nonintegral* vertices of P. This concept is termed as *cut-plane methods*.

5.4.2 The branch-and-bound method

The *branch-and-bound method* is another method to solve *integer programs* as well as *mixed integer programs*. Similar to the cut-plane method, a first approach is to solve the program in (5.9) *ignoring* the integral condition. If this optimal solution $\mathbf{x}^*_{(1)} \in \mathbb{R}^N$ is *nonintegral*, we may start a *branching* process on any of its *noninteger* components.

Let x^*_n be a noninteger component in $\mathbf{x}^*_{(1)}$. Then $b_1 < x^*_n < b_2$ where b_1 and b_2 are consecutive integers. By augmenting the integer program in (5.9) with either $x_n > b_1$ or $x_n < b_2$, two new integer programs may be solved yielding the optimal solutions $\mathbf{x}^*_{(2)}$ and $\mathbf{x}^*_{(3)}$. If they still yield *nonintegral* solutions, the branching process may be continued.

Now, let each solved program correspond to a node in a *branch-and-bound tree,* with $\mathbf{x}^*_{(1)}$ as the *root* node. An example of such a tree is shown in figure 5.4. Note that the branching process preserves all possible integral solutions of the original problem.

Subsequent branching continues until the first *integral* solution[5]. As

[5]If branching does *not* yield any integer solution, then the polyhedron P has no integral

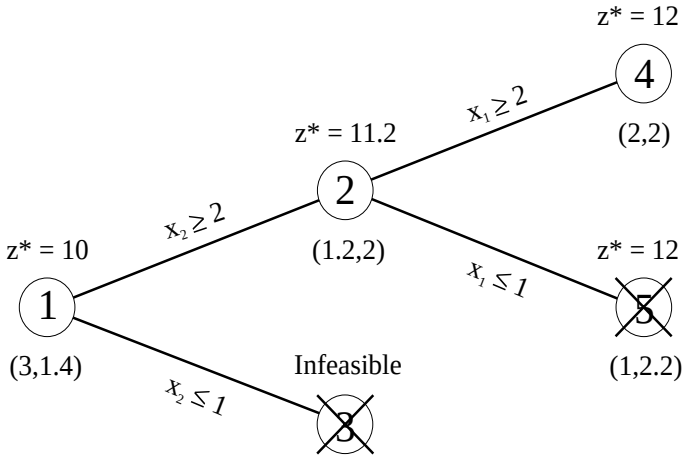

Figure 5.4: An example of a *branch-and-bound tree* of an integer minimization problem. Each node corresponds to a solved integer linear program where the integral constraints are ignored.

shown in figure 5.4, the 4-th node yields an integral solution. The value of the objective function z^* in this node becomes an *upper bound.* That is, *any* node that has an objective value *greater or equal* to this node is discarded, since it can not yield an integral solution with lower objective value. Furthermore, the 5-th node is discarded since its optimal value is equal to the upper bound obtained in the 4-th node. This problem has optimal value $z^* = 12$ with integral solution $\mathbf{x}^*_{(4)} = (2, 2)$. This is also the *optimal value* of the original program, since there does *not* exist any node with smaller objective value to branch from. If a new *integral* solution is obtained later with a smaller objective value, this becomes the new upper bound.

The integer program in (5.9) is a minimization problem. When solving a maximization problem, the first integral solution yields a *lower bound* on the objective. Thus, nodes with objective value *less or equal* to this are discarded.

Unlike the simplex algorithm used in each node, the branch-and-bound method is *not* a polynomial algorithm. The number of nodes in the branch-and-bound tree may grow very rapidly, depending on the integer problem.

points. Consequently, the integer program (5.9) is *infeasible.*

5.5 Quadratic programming

Quadratic programming (QP) is not a major part of this thesis. It is still introduced, since it applies in beamforming when we are supposed to optimize the *energy* in a signal, where the energy function involves a *quadratic form*. A method to solve such problems will not be given.

Similar to a linear program, a *quadratic program* has linear constraints and the *feasible region* describes a polyhedron P. But since the objective function is *nonlinear*, the optimal solution to such a problem is seldom attained at a vertex of P. Consequently, an algorithm to solve a quadratic program can not be based on searching for an optimal solution at the vertices of P like the simplex method. An optimal solution is still attained at the boundary of P.

5.5.1 A quadratic program

The objective function of a quadratic program involves a *quadratic form*. This can be written with a real symmetric negative semi-definite $N \times N$ matrix $\mathbf{C} \in \mathbb{R}^{N,N}$ and a column vector $\mathbf{d} \in \mathbb{R}^N$. As usual, the variables are given as the column vector $\mathbf{x} \in \mathbb{R}^N$. The $M \times N$ system of linear constraints is represented with $\mathbf{A} \in \mathbb{R}^{M,N}$ and $\mathbf{b} \in \mathbb{R}^M$. A general quadratic program is then written in the matrix form

$$\begin{array}{lrcl} \text{Maximize} & z & = & \mathbf{x}^T\mathbf{C}\,\mathbf{x} + \mathbf{d}^T\mathbf{x} \\ \text{Subject to} & & & \\ & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \tag{5.10}$$

where a minimization problem may be obtained by maximizing $-z$.

An optimal solution to (5.10) must satisfy the *Kuhn-Tucker* conditions and it may be solved with *the method of Frank and Wolfe*, which applies an extended version of the simplex algorithm [5].

Chapter 6

Optimization of weighting

This chapter proposes a *general* method to optimize the element *weighting* of ultrasound transducer arrays subject to *array pattern* constraints. More precisely, the *maximum* angular array pattern height will be *minimized* in the sidelobe region. To fit the optimization problem to a *linear program*, the method is restricted to *symmetric linear* and *planar arrays* which fulfills the constraint (4.11) and (4.12) on page 30. Still, the method is *general*, since it applies to *non-equally* as well as *equally* spaced arrays.

The optimization problem is written as a linear program using block matrices, and may be solved with the *simplex algorithm*. The necessary theory background of both beamforming and mathematical optimization should already be established through the previous chapters.

6.1 The objective

As previously stated in the discussion on image quality, there is always a trade-off between the *mainlobe width* and the *sidelobe height* of the array pattern. The objective of the proposed method is to *minimize* the *Chebyshev norm*, or equally the *maximum height,* of the angular array pattern in the sidelobe region. The mainlobe width is implicitly restricted to the boundary between the mainlobe and the sidelobe region.

In FIR filter design in classical digital signal processing, there exist several methods to optimize the filter taps. These methods apply directly to the weighting of *linear* arrays having *equal* $\lambda/2$ interelement spacing. Similar to the proposed method, the conventional *Remez algorithm* minimizes the

maximum sidelobe height of the FIR filter frequency response, yielding the common *Dolph-Chebyshev* weighting.

In [11], a generalized Remez algorithm was also proposed to solve the problem for *sparse* and *non-equally* spaced arrays. This method failed, as it only managed to control a fixed number of sidelobes. Actually, the number of sidelobes is quite variable with non-equally spaced arrays. Thus, a general method to optimize the weights of such arrays, demands a more flexible way of controlling the sidelobes. This motivates for the linear programming approach, since a linear program allows *any finite number of inequality constraints,* which may be used to control *any finite number of sidelobes.*

6.2 Problem formulation

The optimization problem is based on the symmetric angular array pattern function $W(\phi,\theta)$ derived on page 30. This function may also be written in matrix form as

$$W(\phi,\theta) = \mathbf{v}(\phi,\theta)^T \mathbf{w} \tag{6.1}$$

where $\mathbf{v}(\phi,\theta) \in \mathbb{R}^N$ is the kernel vector with $v(\phi,\theta)_n = 2\cos\left\{\frac{2\pi}{\lambda}\vec{s}_{\phi,\theta}\cdot\vec{x}_n\right\}$ as the n-th entry and $\mathbf{w} \in \mathbb{R}^N$ are half the $2N$ symmetric element weights. For fixed *linear* and *planar* arrays, the dot product simplifies to $\vec{s}_{\phi,\theta}\cdot\vec{x}_n = \sin\phi\ (x_n\cos\theta + y_n\sin\theta)$.

The idea is to *minimize* the *sidelobe level* of the array pattern in a continuous region $\mathcal{R}$ of the $\phi\theta$-plane, corresponding to the sidelobe region given in figure 4.5. Note that the half sphere of actual directions may also be covered with ϕ and θ in the intervals $\phi \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Also observe that $v(\phi,\theta)_n = v(-\phi,\theta)_n$ which implies that $W(\phi,\theta) = W(-\phi,\theta)$ and the array pattern is symmetric about the θ-axis[1]. Hence, it is only necessary to optimize the array pattern in the right half-plane. We are thus left with the optimization region $\mathcal{R}$ shown in figure 6.1.

To control the array pattern function in the continuous region $\mathcal{R}$, we choose to discretisize $\mathcal{R}$ in M gridpoints $R = \left\{\ (\phi_1,\theta_1)\ \ \cdots\ \ (\phi_M,\theta_M)\ \right\}$. The *sidelobe level,* denoted δ_s, is then defined on the discrete set R as

$$\delta_s = \max_{(\phi_m,\theta_m)\in R} |W(\phi_m,\theta_m)| \tag{6.2}$$

[1]In fact, we have even more symmetry. Because of the periodicity of the sine functions, we also have $W(\phi,\theta) = W(\phi \pm n\pi, \theta \pm n\pi)$ for any integer $n \in \mathbb{Z}$.

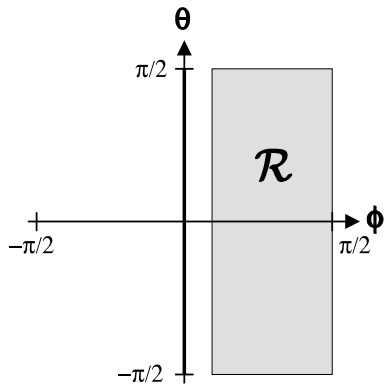

Figure 6.1: The continuous optimizaton region $\mathcal{R}$ in the $\phi\theta$-plane.

We also demand a *normalized* mainlobe in the array pattern. Since the mainlobe peak coincides with the θ-axis, we get the following normalization constraint

$$W(0,\theta) = \mathbf{2}_N^T \mathbf{w} = 1 \tag{6.3}$$

where $\mathbf{2}_N$ is the N element column vector with each element equal to 2, since we have $v(0,\theta)_n = 2$.

The optimization problem may now be stated loosely as

$$\begin{array}{ll} \text{Minimize} & \textit{Sidelobe level} \\ \text{Subject to} & \textit{Normalized mainlobe} \end{array} \tag{6.4}$$

With (6.2) and (6.3), it may be written more formally as

$$\begin{array}{lrcl} \text{Minimize} & z & = & \delta_s \\ \text{Subject to} & & & \\ & W(0,\theta) & = & 1 \\ & |W(\phi_m,\theta_m)| & \leq & \delta_s \quad \forall \quad (\phi_m,\theta_m) \in R \end{array} \tag{6.5}$$

where δ_s on the right side of the inequality is used to force the sidelobe level down in the M gridpoints in R. Observe that the equality and the M absolute values may be written as two inequalities each. The optimization problem can then be transformed to one of the linear programming matrix forms

$$\begin{array}{lrcl} \text{Minimize} & z & = & \mathbf{c}^T\mathbf{x} \\ \text{Subject to} & & & \\ & \mathbf{A}\mathbf{x} & \leq & \mathbf{b} \end{array} \tag{6.6}$$

where the variable vector $\mathbf{x}$ consists of *both* the N element weights $\mathbf{w} \in \mathbb{R}^N$ and the sidelobe level indicator δ_s. The block matrices $\mathbf{c} \in \mathbb{R}^{N+1}$, $\mathbf{A} \in \mathbb{R}^{2+2M,N+1}$ and $\mathbf{b} \in \mathbb{R}^{2+2M}$ have the indicated dimension and is explicitly written as

$$\begin{array}{l} \text{Minimize} \quad z = \begin{bmatrix} \mathbf{0}_N^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \delta_s \end{bmatrix} \\ \text{Subject to} \\ \quad \begin{bmatrix} \mathbf{2}_N^T & 0 \\ -\mathbf{2}_N^T & 0 \\ & \\ \mathbf{v}^T(\phi_1,\theta_1) & -1 \\ -\mathbf{v}^T(\phi_1,\theta_1) & -1 \\ \vdots & \vdots \\ \mathbf{v}^T(\phi_M,\theta_M) & -1 \\ -\mathbf{v}^T(\phi_M,\theta_M) & -1 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \delta_s \end{bmatrix} \leq \begin{bmatrix} 1 \\ -1 \\ \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \end{array} \tag{6.7}$$

where $\mathbf{0}_N$ and $\mathbf{2}_N$ are column vectors with all the N elements equal to 0 and 2, respectively. The kernel vector $\mathbf{v}(\phi_m,\theta_m) \in \mathbb{R}^N$ is given in (6.1) and determined for all the M gridpoints (ϕ_m,θ_m) on $\mathcal{R}$.

6.2.1 The dual problem

Sometimes it is more efficient to solve the *dual* than the *primal* program. Here, the $\mathbf{A}$ matrix in the primal program (6.7) has $2M+2$ rows and $N+1$ columns, where M is the number of discrete points on $\mathcal{R}$ and N are half the $2N$ element weights by symmetry. For most purposes $M > N$. With 2D arrays, we even have $M \gg N$. With this kind of problem, it is more efficient to solve the *dual* problem [6], which is

$$\begin{array}{lrcl} \text{Maximize} & z & = & \mathbf{b}^T\mathbf{y} \\ \text{Subject to} & & & \\ & \mathbf{A}^T\mathbf{y} & = & \mathbf{c} \\ & \mathbf{y} & \leq & \mathbf{0} \end{array} \tag{6.8}$$

where $\mathbf{c}, \mathbf{x}, \mathbf{A}$ and $\mathbf{b}$ are identic in the two problems.

From the *duality theorem* on page 41, we have that the optimal value z^* in the primal and the dual program coincides. Let $\mathbf{y}^* \in \mathbb{R}^{2M+2}$ be an optimal solution to (6.8). Note that each element in $\mathbf{y}^*$ corresponds to a column in $\mathbf{A}^T$, or equally a row in $\mathbf{A}$. Let $\mathbf{y}_B^*$ contain the $N+1$ nonzero elements in $\mathbf{y}^*$ and let $\mathbf{A}_B \in \mathbb{R}^{N+1,N+1}$ contain the corresponding rows in $\mathbf{A}$. Also let $\mathbf{b}_B \in \mathbb{R}^{N+1}$ correspond to these rows. Hence, an optimal solution $\mathbf{x}^* \in \mathbb{R}^{N+1}$ of the *primal* program (6.7) may be established as

$$\mathbf{x}^* = \mathbf{A}_B^{-1}\mathbf{b}_B$$

where $\mathbf{x}^* \in \mathbb{R}^{N+1}$ is a vertex, defined by $\mathbf{A}_B$ and $\mathbf{b}_B$, of the underlying polyhedron.

6.3 Implementation

The weight optimization program was implemented in ULTRASIM which is a MATLAB TOOLBOX for simulating ultrasound waves from different array transducers. The large matrices in (6.7) are conveniently set up in MATLAB. The optimal weights are obtained as a solution to the dual problem (6.8). This problem is solved with the *simplex* algorithm in CPLEX, which is used as an optimization engine. A C-program was written to transfer the large amount of data between MATLAB and CPLEX.

Chapter 7

Optimization of thinning and weighting

When producing array transducers, the large number of elements obviously adds complexity to the design. This becomes a significant problem, especially with 2D arrays. The total costs are also related to the number of array elements.

Thus, an interesting approach to the array design problem is to *minimize* the *number* of array elements with constraints on the array pattern. In this chapter, such a method is proposed. In addition, it *simultaneously* optimizes the element *weighting*, keeping a fixed sidelobe level δ_S. The problem is formulated as a *mixed integer program* using block matrices, which may be solved with the *branch-and-bound* method.

Similar to the weight optimization in the previous chapter, the method is restricted to *symmetric linear* and *planar arrays* which fulfills the constraint (4.11) and (4.12) on page 30. Consequently, the thinning will reduce the total number of elements in pairs, ensuring a real array pattern.

7.1 Objective

From the previous chapter, a method to optimize the *weighting* of general symmetric planar and linear arrays is established. But it requires an array with *fixed* element positions. The weight optimization method yields the lowest sidelobe level with *a* particular element configuration. *But is this the optimal configuration?*

In 2D array transducer design, element weighting is probably not that significant, compared to element placement. One reason is the desire for sparse arrays, which demands a way to perform thinning. The *element placement* or *thinning* problem is probably more significant in this case.

The element placement problem is elaborated in many articles, in addition to the proposed method in this thesis. For instance, in [24] a dynamic programming method is proposed, while a simulated annealing inspired algorithm is given in [30]. In [43] a genetic thinning algorithm is suggested. A similar problem also arise in the design of maximally sparse antennas, which may be formulated as a nonlinear mathematical program [25].

All methods applied to optimize the thinning are affected by the combinatorial nature of the element placement problem. This necessarily results in a rapid growth in calculation time for large arrays. Such methods are thus naturally restricted to smaller arrays.

Still, the proposed method may hopefully be used to determine the gain of *optimized* thinning compared to *random* thinning, which is an alternative in the design of large sparse arrays. Solutions to this problem may also reveal some fundamental questions about the possible reduction of array elements.

7.2 Problem formulation

The proposed simultaneous *weighting and thinning* method is a natural extension of the element *weighting* method in the previous chapter. The symmetric angular array pattern function $W(\phi,\theta)$ will be the basis for this method too. It is written as

$$W(\phi,\theta) = \mathbf{v}(\phi,\theta)^T \mathbf{w} \tag{7.1}$$

and restricted to the symmetric *linear* and *planar* arrays on page 30. The kernel vector is $\mathbf{v}(\phi,\theta) \in \mathbb{R}^N$ with the n-th entry $v(\phi,\theta)_n = 2\cos\left\{\frac{2\pi}{\lambda}\,\vec{s}_{\phi,\theta}\cdot\vec{x}_n\right\}$ and $\mathbf{w} \in \mathbb{R}^N$ are half the $2N$ symmetric element weights. With the restriction above, the dot product simplifies to $\vec{s}_{\phi,\theta}\cdot\vec{x}_n = \sin\phi\,(x_n\cos\theta + y_n\sin\theta)$.

Since the objective is to *minimize* the *number* of array elements, we may introduce a *binary* variable $y_n \in \{0,1\}$ corresponding to each element $\vec{x}_n$. This variable is intended as a counter variable with $y_n = 1$ indicating that the element is present and $y_n = 0$ when the element is removed by thinning. Such a variable may be written in an *integer programming* form as

$$\begin{gathered} 0 \leq y_n \leq 1 \\ y_n \in \mathbb{Z} \end{gathered} \tag{7.2}$$

where $\mathbb{Z}$ is the set of integral numbers.

In this problem, we will also introduce *upper* and *lower* bounds on the weights $\in \mathbb{R}^N$. For simplicity, we assume equal bounds on each weight w_n. In addition, the weight value will be bounded by the binary counter variable $y_n \in \{0, 1\}$ as

$$\alpha \cdot y_n \leq w_n \leq \beta \cdot y_n \tag{7.3}$$

where α is the lower and β is the upper bound. This is sometimes referred to as the *dynamic weight range*.

The array pattern will be controlled in a continuous region $\mathcal{R}$ of the $\phi\theta$-plane, the *sidelobe region*. Similar to the weighting problem, we only need to consider the optimization region $\mathcal{R}$ shown in figure 6.1, since the array pattern is symmetric about the θ-axis. The continuous region $\mathcal{R}$ is discretisized in M gridpoints $R = \left\{ \begin{array}{ccc} (\phi_1, \theta_1) & \cdots & (\phi_M, \theta_M) \end{array} \right\}$ and the sidelobe level δ_s is defined on this set as

$$\delta_s = \max_{(\phi_m, \theta_m) \in R} |W(\phi_m, \theta_m)| \tag{7.4}$$

Note that the *objective* is here to *minimize* the *number* of array elements, rather than minimizing the sidelobe level δ_s. The sidelobe level will consequently be a *fixed* parameter in this problem.

Furthermore, the mainlobe in the array pattern may be normalized by the constraint

$$W(0, \theta) = \mathbf{2}_N^T \mathbf{w} = 1 \tag{7.5}$$

where $\mathbf{2}_N$ is the N element column vector with each element equal to 2, since we have $v(0, \theta)_n = 2$.

The optimization problem may now be stated loosely as

$$\begin{array}{ll} \text{Minimize} & \textit{Number of elements} \\ \text{Subject to} & \\ & \textit{Weight bounds} \\ & \textit{Normalized mainlobe} \\ & \textit{Fixed sidelobe level} \end{array} \tag{7.6}$$

and with the constraints in (7.2), (7.3), (7.4) and (7.5), it may be written more formally as

$$\begin{array}{lrcll} \text{Minimize} & \sum\limits_n y_n & & & \\ \text{Subject to} & & & & \\ & w_n & \in & [\alpha \cdot y_n,\, \beta \cdot y_n] & \forall \quad n \\ & W(0,\theta) & = & 1 & \\ & |W(\phi_m,\theta_m)| & \leq & \delta_s & \forall \quad (\phi_m,\theta_m) \in R \\ & y_n & \in & \{0,1\} & \forall \quad n \end{array} \tag{7.7}$$

where $W(\phi_m, \theta_m)$ is the symmetric angular array pattern determined by the N element weights w_n and the associated counter variables y_n. The interval $[\alpha \cdot y_n,\, \beta \cdot y_n]$ is the dynamic weight range and δ_s is the fixed sidelobe level. Note that y_n is a binary variable.

By writing the equality and the M absolute values as two inequalities each, the optimization problem can be transformed to a *mixed integer programming* form

$$\begin{array}{lrcl} \text{Minimize} & z & = & \mathbf{c}^T\mathbf{x} \\ \text{Subject to} & & & \\ & \mathbf{Ax} & \leq & \mathbf{b} \\ & & & \\ & \mathbf{x}^s & \in & \mathbb{Z} \end{array} \tag{7.8}$$

where the variable vector $\mathbf{x}$ consists of *both* the element weights $\mathbf{w} \in \mathbb{R}^N$ and the associated counter variables $\mathbf{y} \in \mathbb{R}^N$. The variable vector $\mathbf{x}^s \subseteq \mathbf{x}$ only includes the *integer* variables, which in this particular problem are the counter variables $\mathbf{y} \in \mathbb{R}^N$. The block matrices $\mathbf{c} \in \mathbb{R}^{2N}$, $\mathbf{A} \in \mathbb{R}^{2N+2+2M+2N,2N}$ and $\mathbf{b} \in \mathbb{R}^{2N+2+2M+2N}$ have the indicated dimension and is explicitly written as

$$
\begin{aligned}
&\text{Minimize} \quad z = \begin{bmatrix} \mathbf{0}_N^T & \mathbf{1}_N^T \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{y} \end{bmatrix} \\
&\text{Subject to} \\
&\begin{bmatrix}
-\mathbf{I}_{N\times N} & \alpha\,\mathbf{I}_{N\times N} \\
\mathbf{I}_{N\times N} & -\beta\,\mathbf{I}_{N\times N} \\
\\
\mathbf{2}_N^T & \mathbf{0}_N^T \\
-\mathbf{2}_N^T & \mathbf{0}_N^T \\
\\
\mathbf{v}^T(\phi_1,\theta_1) & \mathbf{0}_N^T \\
-\mathbf{v}^T(\phi_1,\theta_1) & \mathbf{0}_N^T \\
\vdots & \vdots \\
\mathbf{v}^T(\phi_M,\theta_M) & \mathbf{0}_N^T \\
-\mathbf{v}^T(\phi_M,\theta_M) & \mathbf{0}_N^T \\
\\
\mathbf{0}_{N\times N} & -\mathbf{I}_{N\times N} \\
\mathbf{0}_{N\times N} & \mathbf{I}_{N\times N}
\end{bmatrix}
\begin{bmatrix} \mathbf{w} \\ \mathbf{y} \end{bmatrix}
\leq
\begin{bmatrix}
\mathbf{0}_N \\ \mathbf{0}_N \\ \\ 1 \\ -1 \\ \\ \delta_s \\ \delta_s \\ \vdots \\ \delta_s \\ \delta_s \\ \\ \mathbf{0}_N \\ \mathbf{1}_N
\end{bmatrix} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{y} \in \mathbb{Z}
\end{aligned}
\tag{7.9}
$$

where $\mathbf{0}_N$, $\mathbf{1}_N$ and $\mathbf{2}_N$ are column vectors with all the N elements equal to 0, 1 and 2, respectively. The $N \times N$ identity matrix is denoted $\mathbf{I}_{N\times N}$ while $\mathbf{0}_{N\times N}$ is the $N \times N$ element zero matrix. The kernel vector $\mathbf{v}(\phi_m, \theta_m) \in \mathbb{R}^N$ is given in (7.1) and determined for all the M gridpoints (ϕ_m, θ_m) on $\mathcal{R}$.

The constraint matrix has $2M + 4N + 2$ rows and $2N$ columns, which is a considerable size. This size and the nature of the problem restricts this method to only smaller arrays.

7.3 Implementation

The simultaneous weighting and thinning optimization program was implemented as an additional part to ULTRASIM. The large matrices in (7.9) are conveniently set up in MATLAB. The optimal array thinning and weighting is obtained by solving the block matrix system in (7.9) with the *branch-and-bound* method in CPLEX, which is used as an optimization engine. Actually,

CPLEX allows binary variables in the mixed integer programming form. Consequently, the last block with the constraints on the counter variables $\mathbf{y} \in \mathbb{R}^N$ is not necessary in this special implementation. A C-program was also written to keep track of the data transfer between MATLAB and CPLEX.

Chapter 8

Computer simulations

This chapter is intended to verify that the proposed methods may be used to both optimize the *weighting* and the simultaneous *thinning and weighting* of general symmetric linear and planar arrays. Both methods are demonstrated on different array types. Some interesting results obtained from several computer simulations are also reported in this chapter. These results will help supporting the conclusion in the next chapter.

8.1 Weighting

The weight optimization method has been tested on many arrays, including 1D linear as well as 2D planar arrays. The proposed method also applies to general sparse arrays with irregular array patterns. A drawback with this method is the large matrices required for optimizing the largest transducer arrays. Still, an optimal solution is obtained within reasonable time thanks to the efficiency of the simplex algorithm.

Some examples will now be shown. The parameters describing the sidelobe region is given together with the optimization time. The latter refers to this particular implementation and the equipment described in appendix A. It will hopefully give an idea of the complexity of the problems at hand.

In each example, the element weight amplitudes are shown. The resulting array patterns are also given, with the array pattern of the unweighted array transducer included as a reference.

8.1.1 Full linear array

The weighting program was first applied to a 64 element regular linear array with $\lambda/2$ interelement spacing. The sidelobe region was chosen to $\phi \in [2.5°, 90°]$ and the weights were obtained after 2.6 sec with $M = 256$ control points. The resulting weight amplitudes are shown together with the array pattern in figure 8.1.

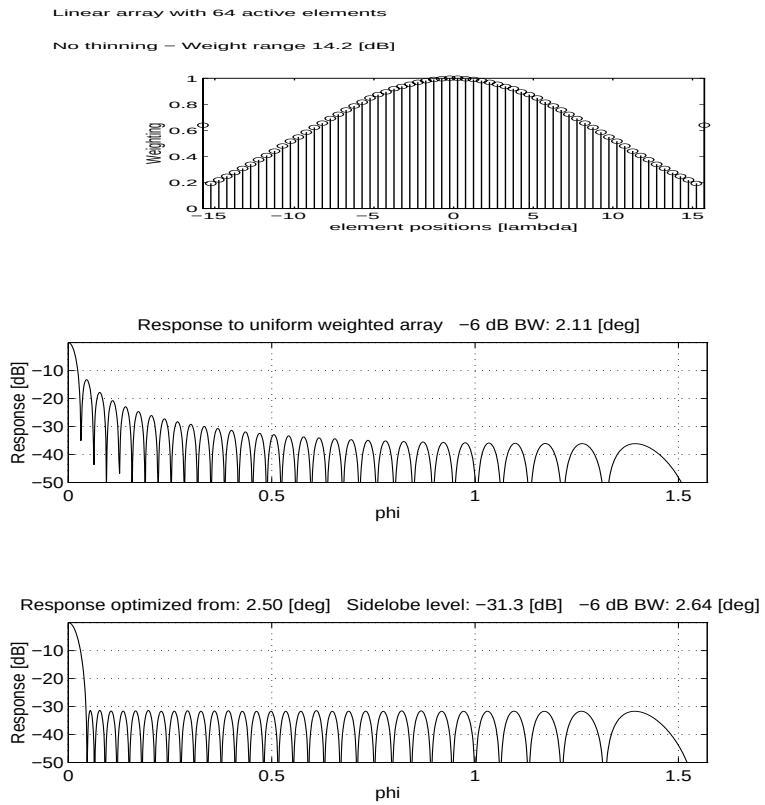

Figure 8.1: Weighting of a 64 element regular linear array. The weights (upper) resemble the well-known Dolph-Chebyshev window function. The array pattern of the uniform weighted array (middle) has falling sidelobes, while the array pattern of the optimally weighted array (lower) has sidelobes with equal height.

8.1.2 Randomly thinned linear array

The 64 element regular linear array was then thinned 25% randomly. A new weighting was performed. Again, the sidelobe region was initially set to $\phi \in [2.5°, 90°]$ with $M = 256$ control points. The optimal weight amplitudes and the array pattern are shown figure 8.2. The weights were obtained after 1.6 sec.

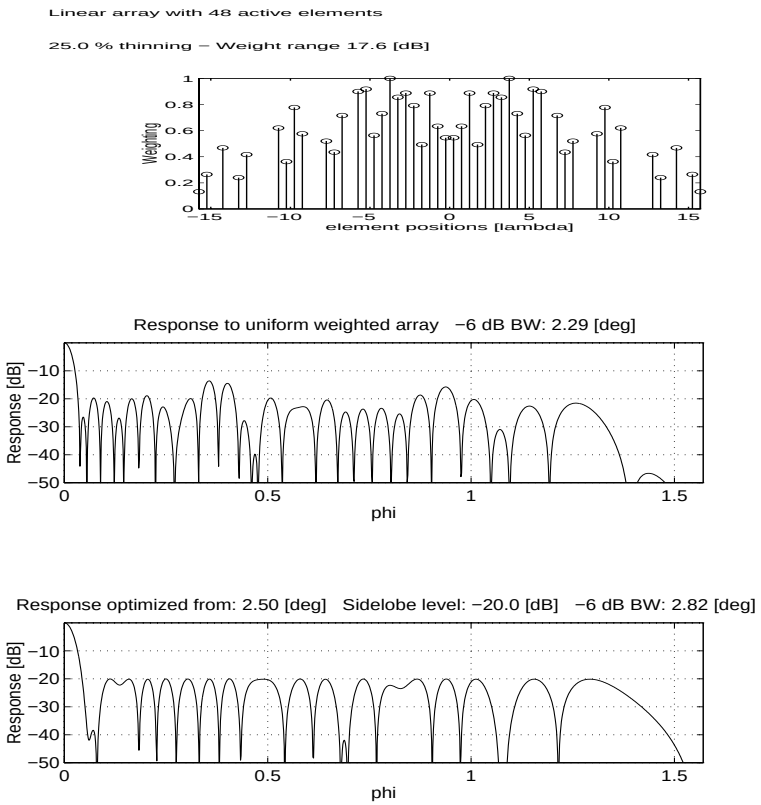

Figure 8.2: Weighting of a 25% thinned 64 element regular linear array. The weights (upper) are more varying than in the previous example. The array pattern of the uniform weighted array (middle) has a higher sidelobe level than the array pattern of the optimally weighted array (lower).

8.1.3 Full 2D planar array

The weighting program was then applied to a 2D planar array. The array was rectangular with 24×24 elements inscribed in a circle, a total number of 448 elements. The interelement spacing was $\lambda/2$ in both the x and y direction. The sidelobe region was chosen to $\phi \in [6°, 90°]$ and the weights were obtained after about 5 minutes with $M = 2352$ control points. The resulting weights and array pattern is shown in figure 8.3.

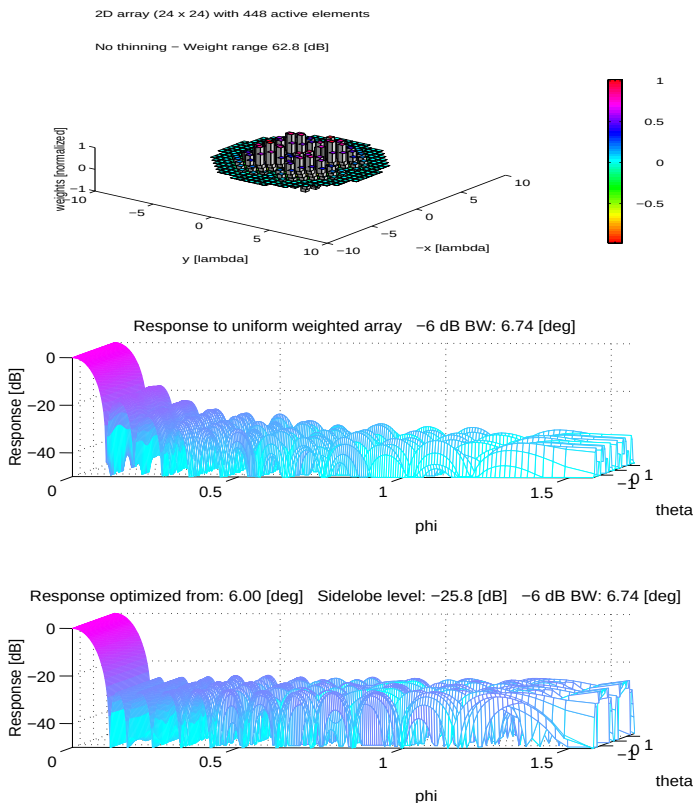

Figure 8.3: Weighting of a 24×24 element circle inscribed planar regular array. The weights (upper) are seen to have the highest amplitudes near the array centre, dropping off at the ends. Both positive and negative weight values are observed. The array pattern of the uniform weighted array (middle) has falling sidelobes, while the array pattern of the optimally weighted array (lower) has sidelobes with nearly equal height.

8.1.4 Randomly thinned 2D planar array

The previous 24×24 element circle inscribed array was then thinned 75% randomly, resulting in 112 remaining elements. The sidelobe region was again chosen to $\phi \in [6°, 90°]$ and the optimal weights were obtained after 9.1 sec with $M = 2352$ control points. The weighting and array pattern is shown in figure 8.4.

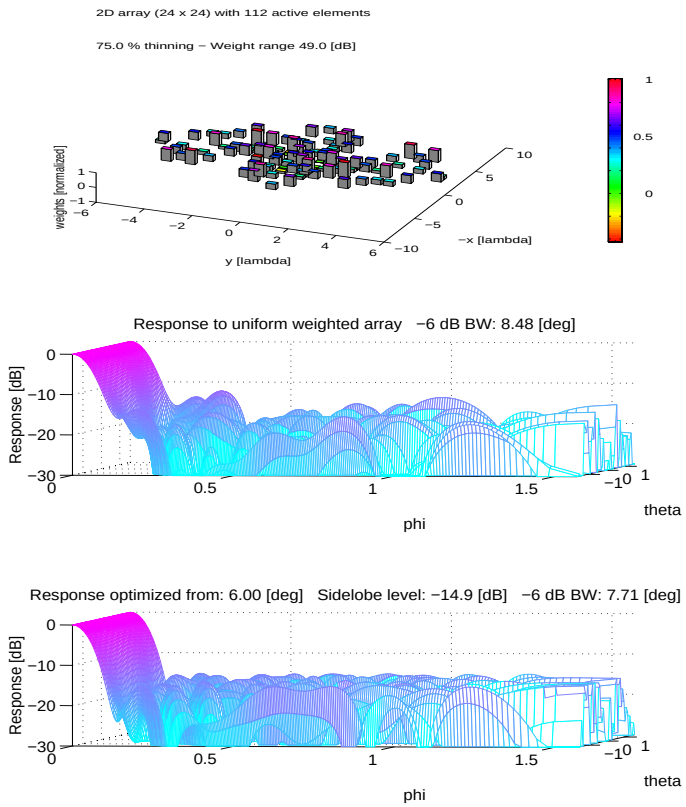

Figure 8.4: Weighting of a 75% randomly thinned 24×24 element circle inscribed array. The weights (upper) are both positive and negative. The array pattern of the uniform unweighted array (middle) is more varying than the array pattern of the optimally weighted array (lower).

8.2 Thinning and weighting

The simultaneous thinning and weighting program has been tested on both linear and planar arrays. With this program, the computation time rather than the size of the matrices limits its applicability. Consequently, only smaller arrays have been optimized. Two examples of simultaneously thinned and weighted arrays are given.

Similar to the previous examples, both the element weight amplitudes and the resulting array patterns are given. The array pattern of the unweighted array is also included as a reference.

8.2.1 Linear array

The simultaneous thinning and weighting program was first applied to a 64 element regular linear array with $\lambda/2$ interelement spacing. The sidelobe region was chosen to $\phi \in [2.5°, 90°]$ and the sidelobe level was upper bounded to $\delta_s = -20$ dB. The optimal thinning and weighting was obtained after 47 minutes! The number of control points was set to $M = 256$. With these initial parameters, the optimization resulted in a 31% reduction of array elements. The resulting weighting and thinning is shown together with the array pattern in figure 8.5.

8.2.2 2D planar array

The simultaneous thinning and weighting program was then applied to a 2D planar array. The array was rectangular with 12×12 elements inscribed in a circle, a total number of 112 array elements. The interelement spacing was $\lambda/2$ in both the x and y direction. The sidelobe region was set to $\phi \in [12.5°, 90°]$ and the sidelobe level was upper bounded to $\delta_s = -20$ dB. With $M = 600$ control points, an optimal thinning and weighting was first obtained after more than 2 hours! But it resulted in an element reduction of 45%. The weighting and thinning is shown with the corresponding array pattern in figure 8.6.

8.3 Beamwidth versus sidelobe level

The resulting weight amplitudes and array pattern characteristics depend strongly on the initial parameters in the optimization program. By deciding the start of the sidelobe region p_0, the -6 dB beamwidth can be closely predicted. This is just a consequence, since p_0 bounds the beamwidth at the sidelobe level. Deciding p_0 also affects the minimum sidelobe level δ_s obtained with the weight optimization program. This is a result of the trade-off between the beamwidth and sidelobe level which is explained with Fourier theory. From the discussion on image quality, an array pattern with both a narrow beamwidth and a low sidelobe level is preferred. But which one to prefer is application dependent.

By varying the sidelobe region and optimizing the weights of one *particular* array several times, the relationship between the beamwidth and the sidelobe level may be revealed for this particular array. Such a relationship has been explored for several arrays. The trends are clear. With regular $\lambda/2$ interelement spaced arrays, the sidelobe level decreases smoothly as p_0 is increased. But, this is *not* true with sparse arrays obtained either with random or optimized thinning. For these arrays, the sidelobe level decreases smoothly *just* to a certain level as p_0 is increased. Dependent on the thinning, these arrays seem to have a *lower bound* on the sidelobe level. This characteristic seem to be more extreme with 1D linear arrays than with 2D planar arrays. The following examples show such relations.

8.3.1 Linear arrays

The relationship between the beamwidth and the sidelobe level has been explored for both a 64 and a 48 element linear regularly $\lambda/2$ spaced array. In figure 8.7, these are compared with four 25% randomly thinned 64 element arrays, each with 48 elements. These curves reveal the difference in characteristics between regularly spaced and thinned arrays.

In this figure, arrays with curves to the lower left are preferred, since they have both a narrow beamwidth and a low sidelobe level. The 25% sparse arrays are obviously limited to the left by the full 64 element array. When the curves of the 25% sparse arrays lie between the solid curves, there *is* a gain with these arrays. The sparse array curves are characteristic as they often follow the full array curve closely to a certain level where they are separated.

8.3.2 2D planar arrays

The beamwidth versus sidelobe level has also been explored for both a 24×24 and a 12×12 element circle inscribed rectangular planar array, with 448 and 112 elements, respectively. These are compared with 75% randomly thinned 24×24 circle inscribed arrays and the beamwidth versus sidelobe level curves are shown in figure 8.8.

Similar to the previous example, the 75% sparse array curves are obviously limited to the left by the full 24×24 element circle inscribed array. The beamwidth versus sidelobe level curves for 2D planar sparse arrays seem to share the same characteristics as the curves for linear arrays, but not that extreme.

8.3.3 Optimally thinned arrays

The relationship between the beamwidth and the sidelobe level has finally been explored for the *optimally* thinned planar array in figure 8.6 with 62 elements. This is compared with two *randomly* thinned 62 element planar arrays obtained from the same initial array. The beamwidth versus sidelobe level curves for these arrays are plotted in figure 8.9.

This figure shows that the optimally thinned array is significantly better than the randomly thinned arrays with respect to a narrow beamwidth and a low sidelobe level.

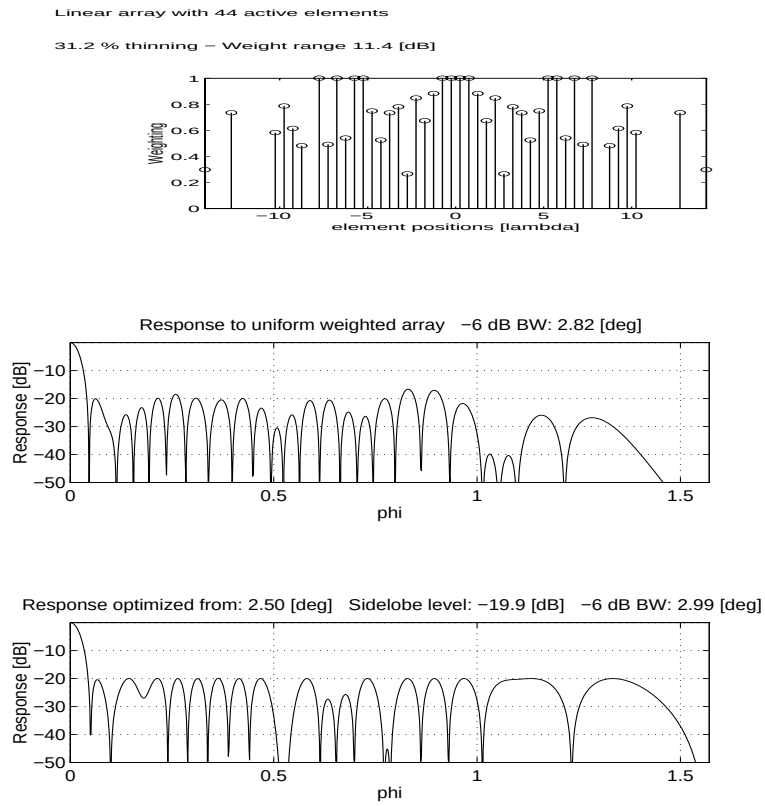

Figure 8.5: Simultaneous thinning and weighting of a 64 element regular linear array. The weights remaining after the optimized thinning (upper) have variying amplitudes. The array pattern of the optimally thinned, but uniform weighted array (middle) has a slightly higher sidelobe level than the array pattern of the simultaneously thinned and weighted array (lower).

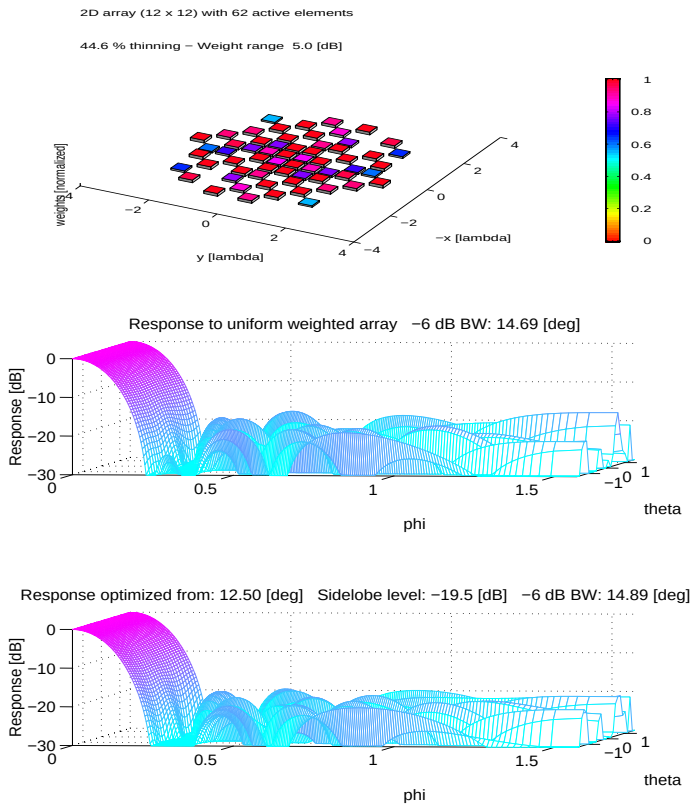

Figure 8.6: Simultaneous thinning and weighting of a 12×12 element circle inscribed planar regular array. The optimally thinned weights (upper) have slightly varying amplitudes. The array pattern of the optimally thinned, but uniform weighted array (middle) has almost the same sidelobe level as the array pattern of the simultaneously thinned and weighted array (lower).

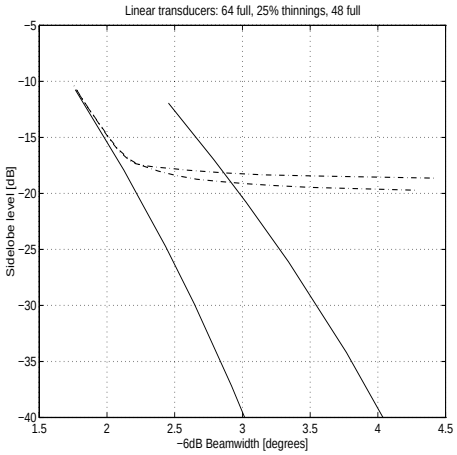

Figure 8.7: The beamwidth versus sidelobe level of the 64 and 48 element regularly spaced linear arrays are shown as solid curves. These are compared to 25% random thinnings of the 64 element array. The 25% sparse arrays are plotted with dotted curves.

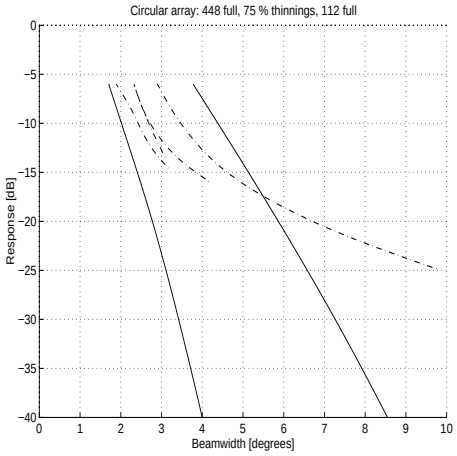

Figure 8.8: The beamwidth versus sidelobe level of both the 24×24 and 12×12 element circle inscribed rectangular planar arrays are shown as solid curves, to the left and right, respectively. These are compared with 75% random thinnings of the 24×24 element circle inscribed array. The 75% sparse arrays are plotted with dotted curves.

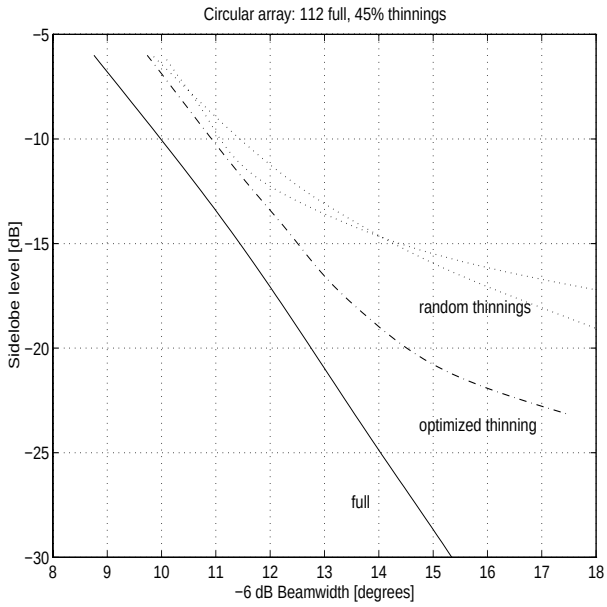

Figure 8.9: The beamwidth versus sidelobe level of a full 12×12 circle inscribed rectangular planar array is shown as the solid curve to the left. This array has been *optimally* thinned, resulting in a 62 element sparse array. The beamwidth versus sidelobe level curve of this array is plotted as a dash-dotted curve in the middle. This curve is significantly better than the dotted curves corresponding to *randomly* thinned 62 element arrays.

Chapter 9

Summary

This thesis started with some observations concerning the design of ultrasound array transducers for medical use. The development of 3D ultrasound seems to require 2D phased array transducers, allowing electronic steering in each direction of a volume. To maintain good resolution and contrast in the images, these arrays should use from $64 \times 64 = 4096$ to $128 \times 128 = 16384$ elements, which complicates the design and implementation.

With this in mind, the work in this thesis has been concentrated on analyzing sparse arrays, obtained from regular arrays by either random or optimal thinning. Optimal ways to perform the weighting or simultaneously thinning and weighting have been presented and implemented. The brief introduction of both ultrasound wave theory and optimization theory has hopefully given the necessary background to evaluate the optimization methods. In the previous chapter, these methods were demonstrated on different arrays.

After simulating several arrays, the relationship between the beamwidth and sidelobe level has been analyzed. This relation appears to have different characteristics whether a sparse array or a full regular array is concerned. Consequently, the applicability of weighted sparse arrays may be limited, since they seem to have a lower bound on the sidelobe level. Weighted full regular arrays are more flexible in respect of weighting, as they do not show such a lower bound.

A further discussion on weighting and optimal thinning is given in this chapter. An answer to the underlying question of the applicability of sparse arrays is given as a conclusion. A section of proposed further work is also included.

9.1 Weighting considerations

The computer simulations of full regular arrays have revealed that an arbitrary sidelobe level may be obtained by adjusting the beamwidth, which is implicitly bounded by the sidelobe region. Unfortunately, this flexibility seems to be limited to full regular arrays. The sparse arrays show the same flexibility just to a certain sidelobe level, which is probably related to the degree of thinning.

For full regularly spaced linear arrays, the proposed element weighting method yields the well-known Dolph-Chebyshev weights shown in figure 8.1. This weighting is characterized with its large weight amplitudes at the end elements. This is undesirable for many applications. However, such spikes were rarely observed with sparse and planar arrays.

As previously mentioned, a major problem with element weighting of 2D phased arrays is the low SNR (signal to noise ration). For sparse arrays with a reduced number of elements, this problem may be even more significant. The element weighting also leads to additional costs and complexity, since each element channel necessarily requires an amplifier.

With sparse arrays, the weighting appeared to smooth the array pattern in the sidelobe region. But the significance of the element weighting seemed to depend on the previously performed element thinning. Some thinnings yielded a few high sidelobes which were conveniently damped with the weight optimization method. Still other thinnings seemed to be less affected by the optimal weighting. An example of the latter is the optimally thinned arrays in figure 8.5 and figure 8.6.

9.2 Optimal thinning

An optimal thinning necessarily yields a better array pattern than a random thinning. Unfortunately, the complexity of the simultaneous thinning and weighting program restricts it to smaller arrays. The interesting question is still how much can be gained with an optimal thinning.

After several simulations of randomly thinned arrays, the thinning was clearly observed to influence the array pattern. In figure 8.9 an optimally thinned array is compared to randomly thinned arrays. This figure shows that there is a significant gain with the optimal thinning. This hypothesis is not statistically verified, but it is supported by the results obtained from the

many computer simulations.

Yet another important observation with the simultaneously thinned and weighted arrays can be made from the examples in figure 8.5 and figure 8.6. The array patterns of the *uniformly weighted* but *optimally thinned* arrays behave surprisingly well in the sidelobe regions.

9.3 Conclusion

The conclusion of this work may now be stated. First of all, the proposed optimization programs solved the given optimization problems, but the simultaneous thinning and weighting program is restricted to smaller arrays. The programs also handle sparse arrays with irregular array patterns.

An answer to the underlying question of whether a 2D sparse array may work satisfactory depends on both the degree of thinning and the thinning method. In respect to a narrow beamwidth and a low sidelobe level, the answer may be found in the figures $8.5 - 8.9$. If a sparse array is compared to a regular array, each having the same number of elements, there may clearly be a gain with the sparse array. It should be noted that sparse arrays seem to have a lower bound on the sidelobe level. But if a higher sidelobe level is acceptable, a sparse array may be almost as good as a full array having significantly more elements, at least in respect to a narrow beamwidth and a low sidelobe level.

If the problem with element weighting of 2D phased arrays is taken into account, optimally thinned but uniformly weighted sparse arrays may even be suggested. This is based on the array patterns of the optimally thinned but unweighted arrays in figure 8.5 and figure 8.6. These show a lower sidelobe level than the full unweighted arrays and nearly the same beamwidth.

A final conclusion based on the computer simulations and the weighting considerations may be that the array *thinning* problem is more important than the element *weighting* problem in the design of 2D arrays for 3D medical ultrasound.

9.4 Further work

The methods presented in this work are restricted to symmetric planar arrays and the optimization criterion was chosen to minimize the maximum sidelobe

level. The optimization problems were then conveniently fit into a linear programming and a mixed integer programming formulation, respectively. This allowed the problems to be solved effectively with the simplex algorithm. Still, the simultaneous thinning and weighting method only applies to smaller arrays.

The restriction to symmetric arrays is certainly a drawback with the given methods. The simultaneous thinning and weighting program might yield a better thinning without this restriction. The methods would even apply to *any* array including 3D arrays. But to control the sidelobe level or the sidelobe energy with asymmetric arrays requires *quadratic* constraints. This suggests other methods than the linear constraint methods given in this thesis.

The optimization criterion chosen probably not yields the best compromise between the resolution and contrast in the image. A better optimization criterion would be to minimize the energy in the sidelobe region rather than minimizing the sidelobe level [1]. But this would lead to a *quadratic* weighting program on the form in (5.10) with the energy given in (4.10). Although it is an interesting approach, it would require even larger matrices and a higher computation time.

As stated in the conclusion, the thinning problem is probably the most important for 2D ultrasound array transducer design. The effort should thus be concentrated on further development of optimal thinning methods. A problem with such methods is the restriction to only smaller arrays.

One attempt to optimize the thinning of larger arrays may be to extend the simultaneous thinning and weighting method proposed in this thesis. By finding mathematical relations between the array elements, a number of equations may be augmented to the model, hopefully reducing the computation time dramatically. This is termed as a cut plane method. The problem is then to find such relations without loosing the freedom of the thinning.

Appendix A

Equipment

This appendix presents the software and hardware used to implement, solve and visualize the results from the optimization problems.

A.1 Software

The software packages used includes MATLAB, CPLEX and ULTRASIM which is a local MATLAB toolbox.

A.1.1 Matlab

MATLAB is a technical computing environment for numeric computation and visualization. It integrates numerical analysis, matrix computation, signal processing, and graphics. The name MATLAB stands for matrix laboratory, as it was originally written to provide easy access to matrix software developed by the Linpack and Eispack projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB is used for research and to solve practical engineering and mathematical problems. Typical uses include general purpose numeric computation, algorithm prototyping, and special purpose problem solving with matrix formulations that arise in disciplines such as automatic control theory, statistics, and digital signal processing.

MATLAB also features a family of application-specific solutions, called toolboxes. Toolboxes are collections of MATLAB functions that extend the MATLAB environment in order to solve particular classes of problems. Areas

in which toolboxes are available include signal processing, control systems design, dynamic systems simulation, systems identification, neural networks, and others.

A.1.2 Cplex

CPLEX is a collection of large-scale mathematical programming software, which features linear, mixed-integer and quadratic programming solvers. It is particularly intended to solve large or difficult problems.

The CPLEX Base System is a linear programming environment which includes fast optimization algorithms as well as a full set of utilities to support solving linear programming problems. The state-of-the-art algorithms include problem reduction algorithms, a modified primal simplex and dual simplex optimizer. In addition a fast network optimizer is included.

CPLEX features an interactive environment as well as external access to the optimization routines through the C programming language.

A.1.3 Ultrasim

ULTRASIM is a local MATLAB toolbox for ultrasound wave simulation. ULTRASIM serves as a standard platform for simulation programs concerning ultrasonic imaging systems. It provides a flexible tool for transducer design – now including *optimized thinning and weighting.* It features different types of wave simulation with facilities as electronic focussing, continuous/pulsed waves and arbitrary frequency. It also provides wave simulation in a homogenous media, as well as a layered media with smooth surfaces.

A.2 Hardware

The optimization programs were compiled and solved on a 64-bit SGI POWERCHALLENGE, using the IRIX Release 6.1 operating system.

Appendix B

User guide to optimization programs

This appendix will explain how to set the initial parameters of the implemented optimization programs. Two guided examples will be given, one from the weighting and the other from the simultaneous thinning and weighting program.

Note that in both programs, the default values (in brackets) are conventionally obtained by hitting ENTER. Prior to starting the optimization programs, an appropriate transducer must be set up.

B.1 Weighting

This example is performed on a 16×16 element rectangular array transducer inscribed in a circle. The weighting program is started from the CONFIGURATION $\rightarrow$ TRANSDUCER $\rightarrow$ WEIGHTING (LP) submenu in the ULTRASIM CONFIGURATION window. It automatically determines the transducer type.

```
Method     :  Minimize Sidelobe Level
Transducer :  2D - symmetric planar array

The symmetric cosine-response sidelobe level is optimized
for (phi, theta) in the area

phi   :  from  p0 to p1 [degrees]
theta :  from -90 to 90 [degrees]
```

```
-> Enter p0 [10] : 12
-> Enter p1 [90] :

-> Do you accept the area [n] ? y
```

The first parameters to be chosen are p_0 and p_1 which describes the *sidelobe region* given as $\phi \in [p_0, p_1]$. This also defines the *mainlobe width* implicitly. A reasonable value for p_0 is chosen close to the ϕ-value of the first *zero crossing* of the *unweighted* array pattern. This value may be found by examining the plot obtained from the CALCULATIONS → FAR-FIELD RESPONSE submenu in the ULTRASIM CONFIGURATION window. The value of p_1 is normally chosen to 90°.

```
Totally 32 x 33 [phi x theta] = 1056 control points

-> Do you accept the controlpoints [y] ?
```

This is the discretisizing of the continuous sidelobe region $\mathcal{R}$, shown in figure 6.1, termed as control points. These values may be increased in either direction for a better sidelobe peak control, or decreased, which speeds up the optimization particularly for large 2D array transducers.

```
Setting up problem ..
Solves problem with cplex ..

-> Do you want to presolve [y] ?
```

A *fast* presolving routine is implemented which finds a linearly independent start basis to the LP problem by a simplified Gauss-elimination procedure. This normally speeds up the optimization.

```
Loading problem ..
Presolving ..

Basis with 105 linearly independent rows found
after checking 224 of totally 2114 rows.

Basis added ..
```

The large LP problem matrices are then transferred from MATLAB to CPLEX, which immediately starts the optimization. The start basis is automatically added to CPLEX if chosen.

```
Optimizing ..

Totally 3466 simplex iterations.

Solution successful.
```

The solution is then returned to MATLAB and ULTRASIM, which shows the optimization statistics

```
-----------------------------------------------------------
Optimization statistics:

Elapsed LP flops:             Not available
Elapsed LP time:                       19.7 [sec]

Weight variation:                  -7.52e-01 (min)
                                    1.09e-03 (absmin)
                                    1.00e+00 (max)

Weight dynamic range:                 59.25 [dB]
SQRW (normalized):                   0.1030
Sidelobe level:                      -37.11 [dB]
-----------------------------------------------------------
```

Only half the $2N$ symmetric weights are optimized. The min, absmin and max weight values correspond to $\min_n(w_n)$, $\min_n|w_n|$ and $\max_n(w_n)$, respectively, and the weight range is given by $20\log_{10}(\max_n|w_n| \,/\, \min_n|w_n|)$. The weight statistics are given for *amplitude* normalized weights, $\max_n(w_n) = 1$. The squared weight value is $\sum_{n=1}^{2N} w_n^2$ where the weights are *sum* normalized to $\sum_{n=1}^{2N} w_n = 1$. The sidelobe level is given as the highest sidelobe peak in the sidelobe region.

Finally, the plot parameters are set and both the weighting and the array pattern is plotted. The scroll-bars may be used to change the 3D view for 2D arrays.

B.2 Thinning and weighting

This example shows a 8×8 element rectangular array transducer inscribed in a circle with 52 remaining elements. The simultaneous thinning and weighting program is started from the CONFIGURATION → TRANSDUCER → WEIGHT AND THIN (CPLEX) submenu in the ULTRASIM CONFIGURATION window.

```
Method     :  Minimize number of elements (ILP).
Transducer :  2D - symmetric planar array

The symmetric cosine-response sidelobe level is controlled
for (phi, theta) in the area

phi   :  from  p0 to p1 [degrees]
theta :  from -90 to 90 [degrees]

-> Enter p0 [10] : 20
-> Enter p1 [90] :
```

Similar to the weighting program, the first parameters to be chosen are p_0 and p_1 describing the *sidelobe region* given as $\phi \in [p_0, p_1]$. This also defines the *mainlobe width* implicitly. A reasonable value for p_0 is chosen close to the ϕ-value of the first *zero crossing* of the *unweighted* array pattern. This value may be found by examining the plot obtained from the CALCULATIONS → FAR-FIELD RESPONSE submenu in the ULTRASIM CONFIGURATION window. The value of p_1 is normally chosen to 90°.

```
-> Enter max sidelobe level in dB [-20] :
```

The sidelobe level δ_s is a fixed parameter in this program while it is a variable in the previous weighting program. A reasonable δ_s may be found by first performing a *weighting* with the sidelobe region $\phi^w \in [p_0^w, p_1^w]$ yielding the minimum sidelobe level δ_s^w. Then, to add freedom to the thinning, we should either choose $\delta_s > \delta_s^w$ or $p_0 > p_0^w$.

```
-> Enter weight LOWER bound (sum w=1/2) [0] :
-> Enter weight UPPER bound (sum w=1/2) [0.03846] :
```

The weights may be bounded by the interval $w_n \in$ [lower,upper] where the scaling factor is given as $2\sum_{n=1}^{N} w_n = 1$.

```
-> Enter cutoff LOWER bound [0] :
-> Enter cutoff UPPER bound [26] :

-> Do you accept the given values [n] ? y
```

Cutoff is explained as *bounding* in the branch-and-bound method. Reducing the cutoff upper bound may reduce optimization time. But care must be taken − as it may lead to an infeasible problem.

```
Totally 16 x 17 [phi x theta] = 272 control points

-> Do you accept the controlpoints [y] ?
```

This is the discretisizing of the continuous sidelobe region $\mathcal{R}$, shown in figure 6.1, termed as control points. These values may be increased in either direction for a better sidelobe peak control, or decreased, which speeds up the optimization particularly for large 2D transducers.

```
Setting up problem ..

Problem written to ..
/home/byleist/a/bjornar/matlab/ultrasim/results/wntprob.mat

-> Do you want to solve it now [y] ?
```

Since this problem class may be very time consuming, the problem can be set up and stored on a file. The problem may for instance be solved as a background process later.

```
Loading problem ..
Lower cutoff set to 0.00.
Upper cutoff set to 26.00.
Optimizing ..

Totally 208700 iterations.

Solution successful.
```

The large LP problem matrices are then transferred from MATLAB to CPLEX, which immediately starts solving the problem. The solution is then returned to MATLAB and ULTRASIM, which shows the optimization statistics

```
------------------------------------------------------------
Optimization statistics:

Elapsed optimization time:            738.9 [sec]

Last cutoff value:                     17.0
Upper cutoff value:                    26.0
Lower cutoff value:                     0.0

Number of MIP nodes:                  12778
Number of unexplored nodes:               0

Elements thinned:                        16

Weight variation:                  2.54e-01 (min)
                                   2.54e-01 (absmin)
                                   1.00e+00 (max)

Weight dynamic range:                  11.89 [dB]
SQRW (normalized):                    0.0302
Max sidelobe level (opt):             -20.00 [dB]
------------------------------------------------------------
```

The last cutoff value is the last upper bound used in the branch-and-bound tree. The number of MIP nodes corresponds to the number of LP subproblems created. It grows very rapidly with the number of array elements.

The number of elements thinned corresponds to the $2N$ number of initial elements. The min, absmin and max weight values correspond to $\min_n(w_n)$, $\min_n|w_n|$ and $\max_n(w_n)$, respectively, and the weight range is given by $20\log_{10}(\max_n|w_n| \,/\, \min_n|w_n|)$. The weight statistics are given for *amplitude* normalized weights, $\max_n(w_n) = 1$. The squared weight value is $\sum_{n=1}^{2N} w_n^2$ where the weights are *sum* normalized to $\sum_{n=1}^{2N} w_n = 1$. The sidelobe level is given as the highest sidelobe peak in the sidelobe region.

Finally, the plot parameters are set and both the weighting and the array pattern is plotted.

Bibliography

[1] J. W. Adams, "A new optimal window", IEEE Transactions on signal processing, vol. 39, no. 8, August 1991.

[2] B. A. J. Angelsen, "Waves, Signals and Signal Processing in Medical Ultrasonics", Ch. 2,12. Department of Biomedical Engineering, University of Trondheim, 1991.

[3] B. A. J. Angelsen, H. Torp, S. Holm, K. Kristofferen and T. A. Whittingham, "Which Transducer Array is best?", European Journal of Ultrasound, vol. 2, pp. 151-164, 1995.

[4] H. Anton, "Elementary linear algebra", John Wiley & Sons, Inc., 6th edition, 1991. ISBN 0-471-54439-6.

[5] R. Bronson, "Operations Research", McGraw-Hill Inc., 1982. ISBN 0-07-007977-3.

[6] V. Chvatal, "Linear programming", Ch. 1-10, 17. W. H. Freeman and Company, 1983. ISBN 0-7167-1195-8.

[7] G. Dahl, "An introduction to convexity, polyhedral theory and combinatorial optimization", Kompendium 67, University of Oslo, Department of Informatics, 1996.

[8] E. S. Ebbini and M. O'Donnell, "Compensation for Missing Elements by Direct Beam Synthesis", Proc. IEEE Ultrasonics symposium, pp. 645-650, 1991.

[9] C. H. Edwards, Jr. and D. E. Penney, "Calculus and Analytic Geometry", Prentice-Hall International, Inc., 3rd edition, 1990. ISBN 0-13-111006-3.

[10] B. Elgetun and S. Holm, "A Method to Optimize Weighting of General Planar Arrays", Proc. Norwegian Signal Processing Conference, NORSIG-95, September 1995.

[11] J. O. Erstad, "Design of sparse and non-equally spaced arrays for medical ultrasound", Cand. Scient thesis, Department of informatics, University of Oslo, 1994.

[12] J. O. Erstad and S. Holm, "An Approach to the Design of Sparse Array Systems", Proc. IEEE Int. Ultrasonic Symp., Cannes, November 1994.

[13] F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", Proc. IEEE, vol. 66, pp. 51-83, Jan 1978.

[14] E. Hecht, "Optics", Addison-Wesley Publishing Company, 2nd edition, 1987. ISBN 0-201-11611-1.

[15] S. Holm, "Bilde og signalbehandling ved Vingmed Sound AS", Pixel'n, 1994.

[16] S. Holm, "Medical Ultrasound Transducers and Beamforming", Proc. Int. Congress on Acoustics, Trondheim, Norway, June 1995.

[17] S. Holm, "Simulation of Acoustic Fields from Medical Ultrasound Transducers of Arbitrary Shape", Nordic Symposium in Physical Acoustics, Ustaoset, Norway, January 1995.

[18] S. Holm, "Ultralydteknikker", Review, October 1995.

[19] S. Holm and B. Elgetun, "Optimization of the Beampattern of 2D Sparse Arrays by Weighting", Proc. IEEE Int. Ultrasonics Symposium, Seattle, Washington, November 1995.

[20] J. A. Jensen and N. B. Svendsen, "Calculation of Pressure Fields from Arbitrarily Shaped, Apodized and Excited Ultrasound Transducers", IEEE Trans. on ultrasonics, ferroelectrics, and frequency control, vol. 39, no. 2, march 1992.

[21] D. H. Johnson and D. E. Dudgeon, "Array signal processing", Prentice Hall, 1993. ISBN 0-13-048513-6.

[22] L. E. Kinsler et al., "Fundamentals of Acoustics", John Wiley & Sons Inc., 3rd edition, 1982.

[23] G. A. Lampropoulos and M. M. Fahmy, "A New Technique for the Design of two-dimensional FIR and IIR Filters", Proc. IEEE Int. Ultrasonics Symposium, Seattle, Washington, November 1995.

[24] J. Lan, R. K. Jeffers and S. G. Boucher, "Optimum Unequally Spaced Arrays and their Amplitude Shading", Proc. IEEE Int. Ultrasonics Symposium, Seattle, Washington, November 1995.

[25] R. M. Leahy and B. D. Jeffs, "On the design of Maximally Sparse Beamforming Arrays", IEEE Trans. on antennas and propagation, vol. 39, no. 8, August 1991.

[26] J. T. Lewis and R. L. Streit, "Real Excitation Coefficients Suffice for Sidelobe Control in a Linear Array", IEEE Trans. on antennas and propagation, vol. AP-30, no. 6, November 1982.

[27] G. R. Lockwood and F. S. Foster, "Design of Sparse Array Imaging Systems", Proc. IEEE Int. Ultrasonics Symposium, Seattle, Washington, November 1995.

[28] G. R. Lockwood and F. S. Foster, "Optimizing sparse two-dimensional transducer arrays using an effective aperture approach", Proc. IEEE Ultrasonics symposium, IEEE Catalog no. 94CH3468-6, pp. 1497-1501, 1994.

[29] J.-Y. Lu, Z. Hehong and J. F. Greenleaf, "Biomedical ultrasound beam forming", Ultrasound in Med. & Biol., vol. 20, no. 5, pp. 403-428, 1994.

[30] V. Murino, A. Trucco and C. S. Regazzoni, "Synthesis of Unequally Spaced Arrays by Simulated Annealing", IEEE Trans. on signal processing, vol. 44, no. 1, January 1996.

[31] G. L. Nemhauser and L. A. Wolsey, "Integer and Combinatorial Optimization", Wiley, New York, 1988.

[32] C. H. Papadimitriou, K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Ch. 1-2, Prentice Hall, 1982. ISBN 0-13-152462-3.

[33] R. Peach, "The Use of Linear Programming for the Design of SAW Filters and Filterbanks", IEEE Trans. on ultrasonics, Ferroelectrics, and Frequency Control, vol. 41, no. 4, July 1994.

[34] L. R. Rabiner, J. H. McClellan and T. W. Parks, "FIR Digital Filter Design Techniques Using Weighted Chebyshev Approximation", Proc. IEEE, vol. 63, pp. 595-610, April 1975.

[35] R. A. Roberts, C. T. Mullis, "Digital signal processing", Addison-Wesley Publishing Company, 1987. ISBN 0-20-16350-0.

[36] M. S. Sherrill and R. L. Streit, "In Situ Optimal Reshading of Arrays with Failed Elements", IEEE Journal of oceanic engineering, vol. OE-12, no. 1, January 1987.

[37] S. W. Smith et al., "Update on 2D Array Transducers for Medical Ultrasound", Proc. IEEE Int. Ultrasonics Symposium, Seattle, Washington, November 1995.

[38] R. L. Streit, "Solution of systems of complex linear equations in the Chebyshev norm with constraints on the unknown", SIAM J. Sci. Stat. Comp., vol. 7, no. 1, January 1986.

[39] R. L. Streit and A. H. Nuttall, "A general Chebyshev complex function approximation procedure and an application to beamforming", J. Acoust. Soc. Am. 72(1), pp. 181-190, July 1982.

[40] R. L. Streit and A. H. Nuttall, "A Note on the Semi-Infinite Programming Approach to complex Approximation", Mathematics of Computation, vol. 40, number 162, pp. 599-605, April 1983.

[41] "The new Grolier Multimedia Encyclopedia", release 6, Grolier Inc., 1993.

[42] C.-Y. Tseng and L. J. Griffiths, "A simple Algorithm to Achieve Desired Patterns for Arbitrary Arrays", IEEE Trans. on signal processing, vol. 40, no. 11, November 1992.

[43] P. K. Weber, R. M. Schmitt, B. D. Tylkowski, J. Steck, "Optimization of Random Sparse 2D Transducer Arrays for 3D Electronic Beam Steering and Focusing", Proc. Int. Congress on Acoustics, Trondheim, Norway, June 1995.

[44] L. Ødegaard, S. Holm and F. Teigen, "Acoustic field simultion for arbitrarily shaped transducers in an aberrating medium", Presentation IEEE Ultrasonic Symp., Nice, France, November 1994.

Index