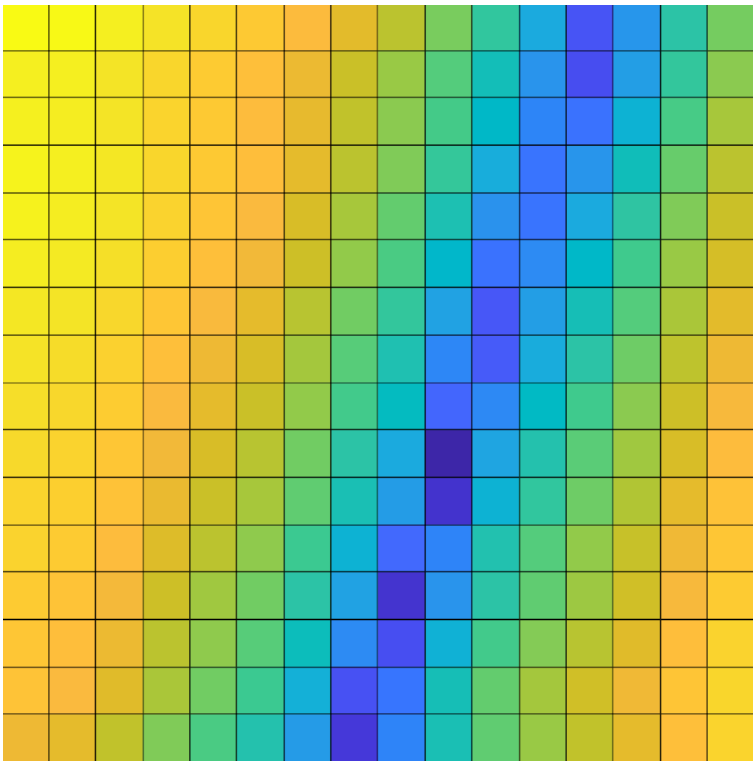


Full Duplex Antenna Array Systems for SatCom Applications

A methodology for modelling and finding optimum beamformer weights for full duplex array systems

Wireless, Photonics and Space Engineering

Teanette van der Spuy
Max Behrens

Department of Electrical Engineering
Chalmers University of Technology
Göteborg, Sweden
2022-10-31

Master's thesis 2022

Full Duplex Antenna Array Systems for SatCom Applications

A methodology for modelling and finding optimum beamformer weights for full duplex array systems

Teanette van der Spuy
Max Behrens

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Communications, Antennas and Optical Networks
Antenna Research Group
Chalmers University of Technology
Göteborg, Sweden 2022

Full Duplex Antenna Array Systems for SatCom Applications
A methodology for modelling and finding optimum beamformer weights for full duplex array systems
Teanette van der Spuy
Max Behrens

Supervisors: Marianna Ivashina, Rob Maaskant, Department of Electrical Engineering
Examiner: Marianna Ivashina, Department of Electrical Engineering

Master's Thesis 2022:01
Department of Electrical Engineering
Communications, Antennas and Optical Networks
Antenna Research Group
Chalmers University of Technology
SE-412 96 Göteborg
Telephone +46 31 772 1000

Cover: Amplitude distribution for a CFM beamformer on a 256 element half-wave dipole array, scanned to 45 degrees phi, 40 degrees theta. The scale is from 0 to -30 dB.

Typeset in LaTeX
Printed by Chalmers Reproservice
Göteborg, Sweden 2022

Full Duplex Antenna Array Systems for SatCom Applications
A methodology for modelling and finding optimum beamformer weights for full duplex array systems
Teanette van der Spuy
Max Behrens
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The increasing need for constant global internet connectivity has pushed the satellite communication (SatCom) industry, which was based on GEO satellites in the past, to expand into LEO and MEO constellations. To utilise these constellations effectively, antennas with strict requirements on bandwidth, frequencies, cost, polarisation, and scanning capabilities need to be developed. Furthermore, the terrestrial user-end transceiver terminals have to comply with radiation emission standards and size/weight requirements, while providing the specified performance at a reasonable cost.

This work presents a signal processing model of a phased array transceiver system for SatCom applications. From this model, optimum antenna element excitations for the array antenna-transceiver system can be derived and key performance and design trade-offs can be identified. The model makes use of a Method of Moments (MoM) electromagnetic solver to characterise the array antenna and a microwave system solver to analyse the signal propagation through the system from the antenna to the transceiver circuitry.

A full duplex 256-element halfwave dipole antenna-transceiver is simulated and studied using the developed model. The system is dual-band, with the receiver (Rx) operating in the 26 GHz band and the transmitter (Tx) in the 29 GHz band. The array beamformer weights providing the desired element excitations for Rx or Tx are calculated for both bands to optimise for gain, isolation, and side-lobe level (SLL) mask compliance respectively over a beam scan angle of up to 60°. It is shown that good isolation and side-lobe level (SLL) mask compliance can be achieved for only a slight reduction in gain performance, when applying optimal beamforming weights.

Keywords: SatCom, phased array, antenna, beamformer

Acknowledgements

We would like to extend our sincere gratitude to our supervisors for supporting us in this endeavour. This work would not have been possible without the guidance and feedback of our supervisors at Chalmers, Marianna Ivashina and Rob Maaskant. We thank you for your patience, encouragement, many fruitful discussions, and the wealth of knowledge and insight that you lent to this work. We are also grateful to Lukas Nyström, our supervisor at Satcube, for his involvement, support, and guidance throughout the thesis.

The majority of the thesis work was conducted in the Gothenburg Satcube offices. We would like to thank the Satcube staff for welcoming us to the team and sharing in the excitement and challenges of our thesis. We look forward to our continued work relationship with them.

Finally, we owe a debt of gratitude to our families who, despite being far away, have supported and encouraged us throughout the thesis. Not only have they enabled us to complete this work, but they provided us with a firm foundation on which to build our skills, the tools to overcome the challenges we faced, and the dream to pursue this work in the first place. We extend this thanks to the friends we have made at Chalmers, who have been our found family during our time in Sweden.

Max Behrens and Teanette van der Spuy, Göteborg, June 2022

Contents

List of Figures

List of Tables

1

Introduction

This chapter will describe the research problem and goals of this thesis. Initially, a brief background of the problem is provided as well as an overview of previous work conducted in this field. Next, the problem goals and specifications for the project outcomes are presented. This is followed by the SatCom specifications and the required adjustments to these specifications. These adjustments are due to computational power limitations and time constraints which limited the maximum amount of radiating elements that could be simulated. The overall scope of the thesis and an outline is presented at the end of this chapter.

1.1 Background and Motivation

Modern communication systems place strict performance requirements on phased array antenna systems. Therefore factors such as the antenna element design, array feeding structures, materials, and implementation technology of these systems have received extensive research attention in recent years. However, one aspect that is often overlooked in the process of the array antenna design is the beamformer coefficients, which can be generally different as compared to the uniform coefficients of the conventional phased-steered arrays.

The conventional approach to design beamformers for phased-array antennas is to apply a uniform amplitude to all channels, with a constant phase shift if beam scanning is required. However, this approach fails to take into account the effects of mutual coupling between antenna elements of the phased array, which means that the corresponding cross-talk effects between the transceiver channels are also neglected in this analysis. Furthermore, while it may be feasible to decouple the antenna elements for a system that is narrow-band, this approach fails for wideband systems that require a wide scan range.

The main purpose of this thesis is to show that, by considering mutual coupling effects, which can be modelled using the EEPs and the full impedance matrix of the antenna, optimal beamformer coefficients can be calculated that produce higher antenna gain their conventional counterparts.

Furthermore, by taking the effects of a microwave system connected to the antenna into account, beamformer coefficients that provide an optimum realised system gain can be found. Lastly, additional beamfomer functionality, such as mask adherence and increased isolation between beamformer ports, are investigated.

1.2 Literature Study

The growing demand for broadband data links cannot be met by current communications systems due to limited spectrum availability. In response to this, high throughput satellite communication (SatCom) systems operating in the K/Ka-band are being designed [3]. However, to meet the demand in this context, the cost, size, and ease of use of SatCom user terminals will have a crucial impact. A promising solution to meet these requirements is a two-dimensional shared aperture phased array with electronic steering capabilities. This work provides a complete mathematical model satellite transceiver system, including the phased array, the connected transceiver system, and the beamforming network. The model makes use of both electromagnetic and signal vector wave calculations to fully characterize the system. Fundamental trade-offs between gain, scan angle, self interference cancellation and applied mask restraints are investigated.

1.2.1 Full Duplex System Modelling

By definition, the proposed satellite terminal makes use of a frequency division duplex (FDD) wireless communication scheme, which is the process of sending and receiving data at the same time, but on different frequency bands.

Several works have have demonstrated duplex functionality (in which data is received and transmitted in the same frequency band) in phased array systems. In both [4] and [5], a design and implementation of a full duplex phased array transceiver is presented at 60 GHz and 2.4 GHz respectively. The full duplex functionality is enabled through hybrid beamforming techniques. In the latter, the beamforming gain in the desired direction is maximised while the overall self interference (SI) power is also reduced. Furthermore, a full-duplex massive MIMO system is investigated in [6] in terms of inter-user interference. In [7] a complete, state-of-the-art SatCom frontend for K/Ka band operation is demonstrated, in which the key the full-duplex dual-band communication that is enabled through the novel on-chip duplexer. However, none of these works investigate the complete satellite transceiver system which includes both the electromagnetic performance of the phased array and the microwave signal interactions between the transceiver circuitry and the antenna.

There are examples where a combined electromagnetic and microwave approach to system modelling was implemented for only the receive or transmit case. In [2], a combined system simulator for radio astronomy receivers is presented. The simulator makes use of both an electromagnetic (EM) simulator and a microwave simulator (MW) for modelling the signal and noise characteristics of each receiver component. Similar analysis strategies are employed in [8], [9] and [10]. However, the transmit

case is not considered. Similarly, the transmit case is thoroughly considered for the 28 GHz case in [11], in which the joint effects of PA nonlinear behaviour and array mutual coupling are investigated. The mutual coupling model in this work is limited to the coupling between the antenna element ports, excluding the direct over-the-air electromagnetic coupling effects between the antenna and power amplifiers. The latter effect has been studied in [12]. Furthermore, in this work was shown that the conventional scan element pattern as used to design array antennas can be extended to the active scan element pattern to account for the excitation-dependent (hence, scan angle dependent) nonlinear effects of power amplifiers in array transmitters.

This combined EM-MW system level approach is useful in the design of complex active array antenna systems. In [13], this approach is used to identify beamformers that provide maximum beam sensitivity as well as maximum continuity of a field of view (FOV) for a multi-beam array receiver system. Furthermore, the work in [14] illustrates the usefulness of this technique for transmit systems, in which the PA and antenna are co-designed to provide improved system performance.

Although several systems exist in which both transmit and receive functionality in the same frequency band are demonstrated [15] [16], a complete model of a dual band full duplex system is lacking. This work investigates and models the behaviour of a full duplex phased array transceiver system with a shared aperture between Rx and Tx. Special attention is given to the the signal propagation through the system from the antenna to the transceiver circuitry.

1.2.2 Beamformer Functionality

Beamforming is a fundamental enabling feature of phased array systems without which many use-cases would not be realisable. For example, the well-known Maximum Gain Beamformer (MGB) (see pages 144-145 in [17]) is necessary to meet the demands for SatCom communications.

For duplex systems, beamforming is also necessary to reduce self-interference (SI) - when a node's transmitting signal generates a significant amount of interference to its own receiver. Even though the Rx and Tx bands are far apart in frequency for this investigation, the power from the transmit signal has the potential to saturate the receive circuitry or to overpower the faint receive signal. One way to mitigate this is through the use of beamforming techniques, generally known as interference cancellation or mitigation techniques.

The work in [18] investigates the usefullness of beamformers in SI cancellation, and in particular points out the drawbacks of having constant amplitude constraints on the beamformers. Furthermore, it's argued that a system with separate Rx and Tx antenna arrays is more flexible and effective in terms of suppressing SI. However, in a system that needs to be small and portable, it is not feasible to double the physical area. Instead, this work will investigated a shared aperture model in which analog beamformers without constant amplitude constraints is present. In [19] the mutual

coupling effects in a phased array is reviewed with a special focus on the spatial interference suppression of arrays by adjusting the beam nulls (this is a type of zero-forcing beamforming). In [5], the beamforming algorithm achieves both reduced SI power as well as a maximised beamforming gain in the desired direction.

Lastly, beamforming is a necessary tool to achieve compliance with stringent SatCom transmit and receive masks.

In this work, the beamforming goals of optimal gain/sensitivity, reduced self interference and mask adherence are investigated and combined with the ambition to achieve these goals simultaneously.

1.3 Project Goals

The chief goal of this work is to develop a model of a dual-band full duplex antenna array system, implement it in Matlab, and use it for the fast analysis of large array antennas which include active components and a Tx/Rx beamforming network. This is especially useful in the analysis of K/Ka-band SatCom phased array communication systems that can integrate with the LEO satellite constellations. This model also takes into account the mutual coupling effects between array elements and the effect that this has on the beamformer design.

The beamformer methods must provide the following performance:

- Maximum realised system gain
- Improved isolation between the transmit and receive beamformer ports (with reference to the inherent isolation present in the system)
- Adherence to the regulatory radiation pattern masks. This includes regulation of the main beam shape as well as SLL constraints

This thesis will not consider the design of the array elements, nor the circuit design of the transceiver. Instead, it is focused on calculating the optimum beamformer coefficients for a specified system that could reasonably fulfil the performance requirements of a standard SatCom system. The system specifications are elaborated on in the following section.

1.4 System Specifications

The operating conditions and performance specifications that are required for integration with existing SatCom frameworks are summarised in Table 1.1.

However, due to time constraints and the limited scope of the thesis, the SatCom system specifications are relaxed for this work. The adjusted specifications are shown in Table 1.2. Notably, the frequency ranges for the receive and transmit bands are changed to fall within the bandwidth of a single-band radiating element,

since the design of a suitable dual-band radiating element falls outside the scope of this project.

Table 1.1: SatCom System Specifications

Specification		Value	
Receive (Rx)			
S1	Frequency	17.7 - 20.2	GHz
S2	Axial Ratio (nominal)	1.0	dB
	(minimum)	1.6	dB
S3	Scanning Angle	0 - 60	deg
S4	G/T (peak)	13.2	dB/K
	(minimum elevation)	9.0	dB/K
S5	Gain (minimum)	30	dBi
S6	Gain Flatness (maximum)	2.5	$\mathrm{dB}_{\mathbf{p-p}}$[1]
S7	Group Delay Ripple (maximum)	4	$\mathrm{ns}_{\mathbf{p-p}}$[1]
Transmit (Tx)			
S8	Transmit Frequency	27.5 - 30.0	GHz
S9	Axial Ratio (nominal)	1.0	dB
	(minimum)	1.6	dB
S10	Scanning Angle	0 - 60	deg
S11	EIRP (peak)	45.3	dBW
	(minimum elevation)	41.0	dBW
S12	Group Delay Ripple (maximum)	4	$\mathrm{ns}_{\mathbf{p-p}}$[1]
System			
S13	Available Area (maximum)	30 x 30	cm

Furthermore, the size of the aperture was reduced to accommodate the limited simulation resources of the available computers while also providing results within a reasonable time frame. Nonetheless, the results of this adjusted system are still useful to analyse beamformer performance and identify trade-offs that can be generalised to larger, real-life systems.

[1] Value taken over any 400 MHz bandwidth

Table 1.2: Adjusted System Specifications

Specification		Value	
Receive (Rx)			
S14	Frequency	25.8 - 26.2	GHz
S15	Scanning Angle	0 - 60	deg
S16	Gain (minimum)	30.8	dBi
Transmit (Tx)			
S17	Transmit Frequency	28.8 - 29.2	GHz
S18	Scanning Angle	0 - 60	deg
S19	EIRP (peak)	43.3	dBW
	(minimum elevation)	39.0	dBW
System			
S20	Available Area	24 x 24	cm

1.5 Thesis Outline

The rest of the thesis is organised as follows:

Chapter 2 presents the fundamental theory and definitions used in this work. This includes basic antenna and array antenna theory. A brief overview of super-directive beamformers is also provided, as well as a section on the signal and noise wave representations that are used in the network model of the transceiver system.

In Chapter 3, the network analysis methods used to model the antenna-transceiver system are discussed. First the modelling of the array antenna and the transceiver circuitry are discussed separately. Thereafter an explanation is given on how the complete system and the microwave interactions between sub-components are modelled. Lastly, the analysis of the system in transmit and receive modes (here called the transmit and receive subsystems) is discussed.

Chapter 4 is focused on the formulation of various beamformers and the algorithms that were used to calculate them. First the maximum realised gain beamformers for both the transmit and receive cases are derived. Next a description of the herein-proposed isolation beamformer and mask adherence beamformer algorithms are provided. Lastly, the method used to combine the maximum gain beamformers as well as the mask and isolation constraints is described.

In Chapter 5 the most relevant results of the system are provided. These include the results of each of the beamformer algorithms implemented individually, as well as the results of the combined algorithm.

The thesis is concluded in Chapter 6, where the results are summarised and the key trade-offs between performance metrics of the system are identified. This chapter also provides recommendations for future work.

2

Theory

This chapter aims to describe and explain the principles and theory on which our work is built. We start by presenting basic antenna and array antenna theory and definitions. An in-depth discussion on embedded element patterns and their formulation is provided here. Furthermore, a brief overview of superdirective beamformers is given. The signal and noise wave representation in the system network model that is used to model the transceiver circuitry and to charactize the interactions between components in the system is discussed. Lastly, the practical consideration and limitations that were taken into account in this work are provided, including regulatory aspects as well as well as component and software limitations.

2.1 Basic Antenna Theory

Consider a single antenna element, excited by a voltage V_1, as shown in Figure 2.1. The current I_1 is defined as going into the antenna load with admittance Y_{ant} such that $I_1 = Y_{\text{ant}} V_1$.

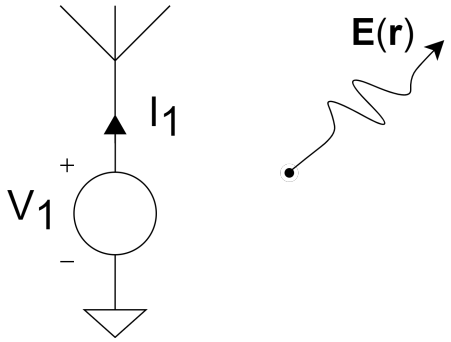


Figure 2.1: An antenna element with voltage excitation V_1, generating an electric field $\mathbf{E}(\mathbf{r})$

When an excitation is applied to the antenna, a radiating electric field $\mathbf{E}(\mathbf{r})$ is generated. If the field is analysed in a region where $r \geq 2D^2/\lambda$, where D is the largest antenna dimension and λ is the free-space wavelength, then the radiated

field reduces to the far-field pattern:

$$\mathbf{E}(\mathbf{r}) = \mathbf{F}(\hat{\mathbf{r}})e^{-jkr}/r \tag{2.1}$$

where $\mathbf{F}(\hat{\mathbf{r}})$ is the far-field pattern and k is the wavenumber. In this region the radiated waves are considered planar and the radiation pattern's shape remains constant over distance.

It is convenient to define the electric field region in terms of spherical coordinates, as shown in Figure 2.2 below. In this figure $\hat{\mathbf{r}}(\theta, \phi) = \sin\theta\cos\phi\hat{\mathbf{x}} + \sin\theta\sin\phi\hat{\mathbf{y}} + \cos\theta\hat{\mathbf{z}}$.

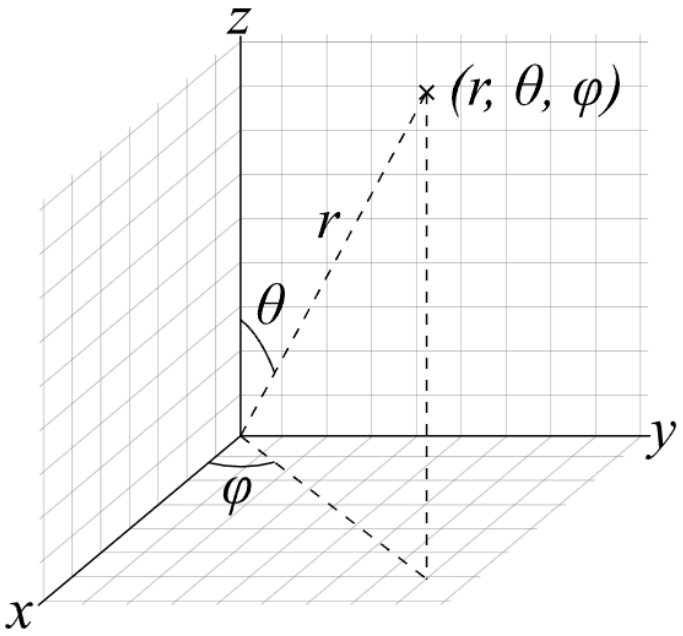
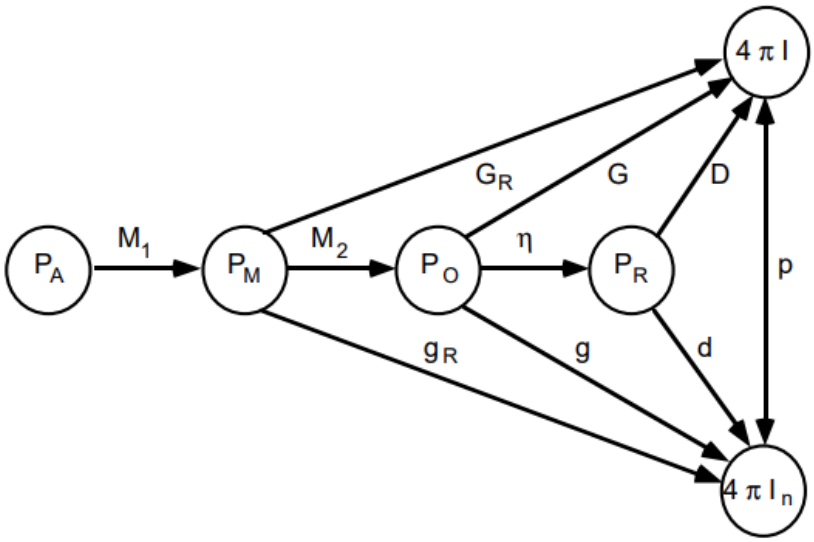


Figure 2.2: Coordinate system definition for the antenna far-field analysis

2.1.1 Directivity and Gain Definitions

Antenna gain can be defined as the ratio between the power radiated by the antenna to the input power, relative to an isotropic element. However, it is important to define the reference plane for the input power in a way that is applicable to the system analysis. Therefore, when defining the gain of an antenna, one needs to consider the effects of radiation efficiency, impedance mismatches and polarization. The IEEE defines the interrelationships of these factors and the various gain definitions that take them into account in the following flowchart [20]. In this work, we consider both the gain of the antenna, which defines the power accepted by the antenna as the input power, and the realised gain of a complete transceiver-antenna system, which defines the power provided to the transceiver system as the input power.



P_A = power available from the generator
P_M = power to matched transmission line
P_O = power accepted by the antenna
P_R = power radiated by the antenna
I = radiation intensity
I_n = partial radiation intensity
M_1 = impedance mismatch factor 1
M_2 = impedance mismatch factor 2

η = radiation efficiency
G_R = realized gain
G = gain
D = directivity
g_R = partial realized gain
g = partial gain
d = partial directivity
p = polarization efficiency

Figure 2.3: Definitions for directivity and gain © [1993] IEEE

With reference to Figure 2.1, the radiated power is defined as

$$P_{\text{rad}}(\theta_0, \phi_0) = \frac{1}{2\eta}|\mathbf{E}(\mathbf{r})r|^2 = \frac{1}{2\eta}|\mathbf{F}(\theta_0, \phi_0)|^2 \tag{2.2}$$

where $\eta = 120\pi$ is the free-space impedance. The time-averaged input power accepted by the antenna is defined as

$$P_{\text{in}} = \frac{1}{2}\text{Re}(V_1 I_1^*) = \frac{1}{2}|V_1|^2\text{Re}\{Y_{\text{ant}}\} \tag{2.3}$$

Therefore the antenna gain can be written mathematically as

$$G(\theta_0, \phi_0) = \frac{P_{\text{rad}}(\theta_0, \phi_0)}{P_{\text{in}}/4\pi} = \frac{4\pi|\mathbf{F}(\theta_0, \phi_0)|^2}{\eta|V_1|^2\text{Re}\{Y_{\text{ant}}\}} \tag{2.4}$$

Because $\mathbf{F}(\theta_0, \phi_0)$ is proportional to V_1, the voltage excitation terms in equation 2.4 cancel each other out. Therefore antenna gain for a single element is independent of the excitation and only a function of antenna geometry.

2.1.2 Polarization

The polarization of an electromagnetic wave describes the orientation of the oscillating electric field. When defined in Cartesian coordinates, a wave travelling in an

arbitrary direction direction may contain x-, y- and z-components. These components are orthogonal and can be considered independently.

2.2 Array Antenna Theory

An array antenna refers to an antenna made up of multiple antenna elements that should work jointly and coherently to enhance the performance with respect to a single antenna. Planar array antennas outperform isolated antenna elements in terms of gain, beam shaping flexibility, and beam steering capabilities. However, their design and integration with a transceiver circuit is more complex than that of a single antenna element.

2.2.1 Embedded Element Patterns

The term *Embedded Element Pattern* (**EEP**) refers to the field radiated by an array when a single element is driven by a source, while the other elements are terminated with a specific impedance. This radiated pattern takes into account the effect of scattering from neighbouring elements as well as any effects from the array structure and materials. In this way, the mutual coupling between elements and the array response is captured in the **EEP**s.

Different **EEP** definitions exist for different loading conditions of the nondriven element ports and the corresponding source of the driven port. Common choices include short circuit loads (where a voltage source is used), open circuit loads (where a current source is used), and matched loads (where either a voltage, current, or power source can be used). In this work we will refer to these definitions as $\mathbf{EEP}^{\mathrm{sc}}$, $\mathbf{EEP}^{\mathrm{oc}}$ and $\mathbf{EEP}^{\mathrm{Zo}}$. In addition, we will also refer to the case where the system is terminated in loads that are not necessarily matched as $\mathbf{EEP}^{\mathrm{Zg}}$.

In this work, all three of the above mentioned **EEP** definitions and the transformations between them are used. To facilitate a matrix treatment of the **EEP**s, the polarizations of the radiated fields are considered independently. Furthermore, the **EEP** corresponding to the excitation of element n is referred to as $\mathbf{EEP}_n$, and gathered into a N-element matrix with the x, y, and z polarized components in different rows:

$$\mathbf{EEP} = \begin{bmatrix} \hat{x} \cdot \mathbf{EEP}_1 & \hat{y} \cdot \mathbf{EEP}_1 & \hat{z} \cdot \mathbf{EEP}_1 \\ \vdots & \vdots & \vdots \\ \hat{x} \cdot \mathbf{EEP}_N & \hat{y} \cdot \mathbf{EEP}_N & \hat{z} \cdot \mathbf{EEP}_N \end{bmatrix} \tag{2.5}$$

In [1], the fundamental relationships between the different definitions of **EEP**s are illustrated. Their work is summarised in the flow chart below. Note that $\mathbf{Z}_{\mathrm{A}}$ and $\mathbf{Y}_{\mathrm{A}}$ refer to the array antenna impedance and admittance matrix respectively, while $\mathbf{Z}_{\mathrm{g}}$ and $\mathbf{Y}_{\mathrm{g}}$ refer to the antenna loading. It is assumed that $\mathbf{Z}_{\mathrm{g}} = \mathrm{Z}_{\mathrm{g}}\mathbf{I}$, where $\mathbf{I}$ is the identity matrix, with a similar formulation for $\mathbf{Y}_{\mathrm{g}}$. Lastly, V_0 and I_0 refer to the magnitude of the voltage and current excitations used to calculated the respective **EEP**s.

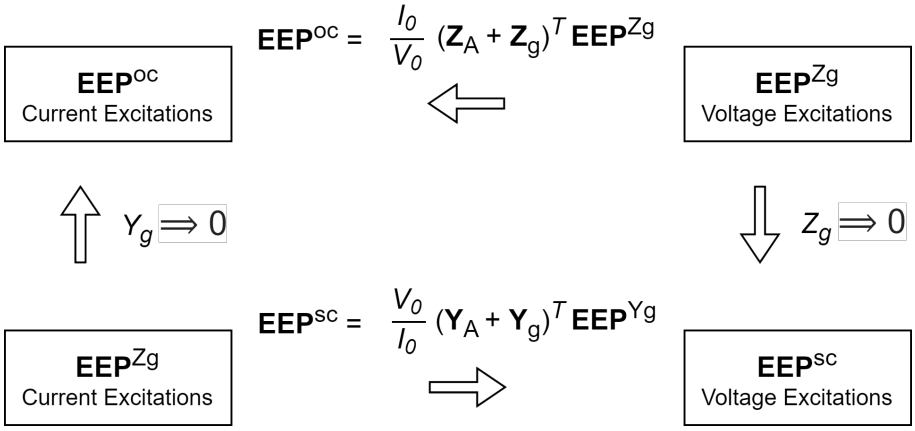


Figure 2.4: Relationships between different **EEP** loading conditions [1]

A further useful discussion from [1] relates the pattern overlap integral, or **EEP** overlap matrix, to the loading conditions. This result is important to calculate the optimal excitation beamformers. These relationships are summarized in the table below, where A represents the pattern overlap integral, $\mathbf{R}_{\mathbf{A,\ loss}}$ and $\mathbf{Y}_{\mathbf{A,\ loss}}$ represent the losses in the array antenna, and $\mathbf{S}_\mathrm{A}$ represents the scattering parameters of the antenna.

Table 2.1: Pattern Overlap Integrals

Loading Condition	Pattern Overlap Integral
Open Circuit	$\mathrm{Re}(\mathbf{Z}_\mathrm{A}) = \frac{2}{\lvert I_0\rvert^2}\mathrm{A}^{\mathrm{oc}} + \mathbf{R}_{\mathbf{A,\ loss}}$
Short Circuit	$\mathrm{Re}(\mathbf{Y}_\mathrm{A}) = \frac{2}{\lvert V_0\rvert^2}\mathrm{A}^{\mathrm{sc}} + \mathbf{Y}_{\mathbf{A,\ loss}}$
Matched Load	$\mathrm{A}^{\mathbf{m}} = \frac{\lvert V_0\rvert^2}{8Z_0}(\mathbf{I} - \mathbf{S}_\mathrm{A}\mathbf{S}_\mathrm{A}^H)$

An important result from the above table is that the pattern overlap integral for an array antenna is proportional to $(\mathbf{I} - \mathbf{S}_\mathrm{A}\mathbf{S}_\mathrm{A}^H)$ [21, 22]. This is especially useful in the calculation of beamformers in which the loaded element patterns are considered.

2.2.2 Active Impedance

Isolated antenna elements can be completely characterised by their self-impedance. However, in an array antenna, the mutual coupling between antenna elements affect their input impedance. Therefore the input impedance of each element, as well as the input of the array antenna itself, is dependent not only on the element geometry, but also the array geometry and the excitation of the system (that is, the excitation

of the other elements). This phenomenon is referred to as the active impedance of the m^{th} element port in an N element array can be defined as:

$$\mathrm{Z}_{\text{act},m} = \mathrm{Z}_{\text{act},mm} + \frac{1}{i_m} \sum_{n=1,\ n\neq m}^{N} \mathrm{Z}_{\text{act},mn} i_n \tag{2.6}$$

2.2.3 Array Gain Definitions

The single element analysis from Section 2.1.1 can be expanded to array antennas with N elements and N voltage excitations, denoted by $\mathbf{v}$. Note that, when considering voltage excitations, we use the corresponding short circuit element pattern defintion $\mathbf{EEP}^{\text{sc}}$. The resulting radiated electric field is rewritten as

$$\mathbf{E}(\mathbf{r}) = \sum_{n=1}^{N} V_n \mathbf{EEP}_n^{\text{sc}}(\mathbf{r}) = \mathbf{v}\mathbf{EEP}^{\text{sc}}(\mathbf{r}) \tag{2.7}$$

where V_n refers to the voltage excitation per element. Similarly, the radiated power per solid angle in the direction of $\hat{\mathbf{r}}$ is defined as:

$$\begin{aligned} P_{\text{rad}}(\hat{\mathbf{r}}) &= \frac{1}{2\eta} |\mathbf{v}\mathbf{EEP}^{\text{sc}}(\hat{\mathbf{r}}) r|^2 \\ &= \frac{r^2}{2\eta} \mathbf{v}^H \mathbf{EEP}^{\text{sc}}(\hat{\mathbf{r}})^H \mathbf{EEP}^{\text{sc}}(\hat{\mathbf{r}}) \mathbf{v} \end{aligned} \tag{2.8}$$

The input power is defined as

$$P_{\text{in}} = \frac{1}{2} \sum_{n=1}^{N} V_n I_n^* = \frac{1}{2} \mathbf{v}^H \mathrm{Re}\{\mathbf{Y}_{\text{ant}}\} \mathbf{v} \tag{2.9}$$

Lastly, the gain of an array antenna is defined as

$$\begin{aligned} \mathrm{Gain}(\theta_0, \phi_0, \mathbf{v}) &= \frac{P_{\text{rad}}(\hat{\mathbf{r}})}{P_{\text{in}}/4\pi} \\ &= \frac{4\pi r^2}{\eta} \frac{\mathbf{v}^H \mathbf{EEP}^{\text{sc}}(\hat{\mathbf{r}})^H \mathbf{EEP}^{\text{sc}}(\hat{\mathbf{r}}) \mathbf{v}}{\mathbf{v}^H \mathrm{Re}\{\mathbf{Y}_{\text{ant}}\} \mathbf{v}} \end{aligned} \tag{2.10}$$

Note that the admittance matrix for an antenna array $\mathbf{Y}_{\text{ant}}$ takes the coupling between elements into account. It is interesting to see that the gain of a phased array is not independent of the excitation, as was the case with a single antenna element.

2.2.4 Beamsteering and Grating Lobes

With reference to equation 2.10, it is clear that antenna excitations can be chosen to maximise the gain in an arbitrary (θ_0, ϕ_0) direction. This effectively shifts the main beam of the antenna to a desired scan angle, and is known as electronic beamsteering. Both the element geometry and array geometry affect an array antenna's beamsteering capabilities.

Antenna performance at extreme scan angles ($> 45°$) is fundamentally degraded due to a decrease in effective aperture area. This decrease is due to the cosine projection of the aperture to the normal of the scanning direction. Due to this cosine factor, a 3 dB reduction in directivity is typically observed when scanning to 45 degrees. The figure below illustrates the reduction in effective aperture area when scanning.

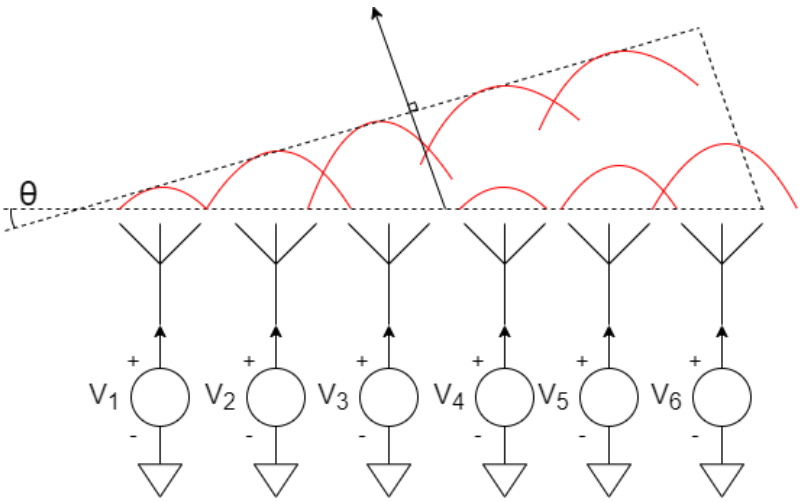


Figure 2.5: Cosine aperture projection when scanning a phased array

Furthermore, due to spatial aliasing, so-called grating lobes can appear at these scan angles. These are unwanted sidelobes at high gain levels which reduce the directivity of the main lobe. The effects of grating lobes cannot be reduced through a choice of antenna excitation, but are exclusively a function of element (i.e. EEP taper) and array geometry (i.e. the number of and spacing between the elements). A well-known condition for no grating lobes is to regulate the spacing of the antenna elements such that [17]:

$$kd \leq \pi \quad \Rightarrow \quad d \leq \frac{\lambda}{2} \tag{2.11}$$

where k is the wavenumber, d is the element spacing, and λ is the wavelength. However, even if this condition is met for a broadside beam, grating lobes can still appear at extreme scan angles. This is the result of the antenna aperture being under-sampled due to the an effectively coarser element spacing when scanning.

2.3 Antenna Reciprocity for Finite Arrays

It is common to model an antenna using transmitting characteristics, such as radiation pattern and gain. When considering a receiving antenna, it is convenient to make use of the well-known reciprocity theorem [as described in chapters 5 and 6 of [17]], which allows us to relate the performance of a receiving antenna to a

reciprocal transmitting antenna. This is particularly useful in the development of an equivalent Thévenin circuit representation of a receive array antenna. In this circuit model, the effect of an incident radiated field is represented by internal voltage sources whose properties are related to the amplitude, phase and polarization of the field. This model will be expanded upon in Section 3.5.1.

In this work, the array antenna is considered reciprocal, and is analysed using the reciprocity theorem. However, the antenna-transceiver system, as shown in Figure 3.2, is **not** reciprocal. Therefore the analyses of the antenna and that of the antenna-transceiver system must be handled separately.

2.4 Superdirective Beamformers

According to [20], superdirectivity is defined as *the condition that occurs when the antenna illumination efficiency significantly exceeds 100%.* One important observation is that superdirectivity is defined with reference to a uniformly excited antenna aperture. In other words, if a beamformer solution can realise a radiation pattern whose directivity exceeds that of a uniformly excited antenna array, it is known as a superdirective beamformer. This can be achieved through choosing beamformers that exploit the mutual coupling between elements for higher gain. However, it is well known that maximum gain beamformers are less robust and more sensitive to array uncertainty than conventional beamformers.

2.5 Signal and Noise Wave Representation

In this work, it is assumed that only small signals are modelled, and therefore a small signal S-parameter matrix approximation can be used. This means that nonlinear components, such as amplifiers and filters, can be modelled as linear components within a small frequency range [23]. To facilitate this approximation, we make use of the following noisy two-port model:

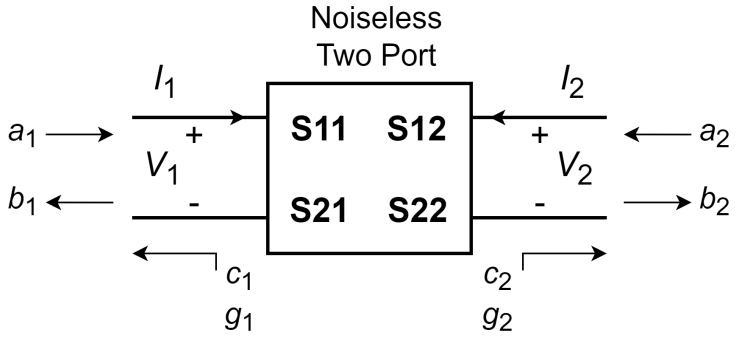


Figure 2.6: Signal and noise wave model

In this model [2], the S-parameter matrix represents a noiseless two port component

which interacts with the complex-valued a- and b-waves. External noise is modelled as c-waves, while internal signal generators are represented as g-waves.

This work considers a noiseless system such that all c-waves $= 0$ (although it can readily be expanded to include noise in the analysis). In the absence of noise and internal signals, the a- and b-waves (also referred to as power waves) are related to the electrical properties of the ports as follows [24]:

$$\begin{aligned} a_n &= \frac{V_n + Z_{0n}^{\text{ref}} I_n}{2\sqrt{\text{Re}(Z_{0n}^{\text{ref}})}} \\ b_n &= \frac{V_n - (Z_{0n}^{\text{ref}})^* I_n}{2\sqrt{\text{Re}(Z_{0n}^{\text{ref}})}} \end{aligned} \tag{2.12}$$

Here Z_{0n}^{ref} refers to the complex-valued normalization impedance for the port. The asterisk denotes the complex conjugate. Correspondingly, the inverse relationships are given by [24]:

$$\begin{aligned} V_n &= \frac{(Z_{0n}^{\text{ref}})^* a_n + Z_{0n}^{\text{ref}} b_n}{\sqrt{\text{Re}(Z_{0n}^{\text{ref}})}} \\ I_n &= \frac{a_n - b_n}{\sqrt{\text{Re}(Z_{0n}^{\text{ref}})}} \end{aligned} \tag{2.13}$$

This model can be expanded to a matrix formulation by collecting the signal waves into column vectors such that

$$\mathbf{b} = \mathbf{S}\mathbf{a} + \mathrm{g} + \mathbf{c} \tag{2.14}$$

where $\mathbf{c} = 0$ in this model. Lastly, from the above definitions, the accepted input power of an N-port device is defined as:

$$\begin{aligned} P_{\text{acc}} &= \mathbf{a}^H\mathbf{a} - \mathbf{b}^H\mathbf{b} \\ &= \mathbf{a}^H(\mathbf{I} - \mathbf{S}^H\mathbf{S})\mathbf{a} \end{aligned} \tag{2.15}$$

2.6 Practical Considerations and Limitations

The implementation and design considerations of the method are influenced by realistic practical considerations and limitations, such as the dynamic range of the LNAs and discrete values of the beamformer ICs. The following subsections will discuss the practical considerations and motivate the chosen implementation.

2.6.1 Receiver Saturation

The coupling between the transmit and receive side results in power present on the receiver LNAs which can saturate the devices resulting in a loss in received information. The saturation can be due to power present at another frequency, such as the transmit frequency, and thus can be mitigated by sufficient filtering.

The isolation can be improved greatly after the signal recombination which occurs after the beamformer ICs, which will remove interference and noise due to the transmit side. However, the issue of saturation still remains regardless of the improvement in isolation.

To determine the maximum allowable power to be coupled to the receiver side from the transmit side a specific beamformer IC has to be chosen and the dynamic range determined. Next, the frequency response of the PAs, coupling, and LNAs have to be considered to determine the total power present at the input of the LNAs. Thereafter, the appropriate filter requirements can be synthesised to prevent saturation.

A trade-off can also be made by reducing the transmit power on each element by either increasing the amount of elements or increasing the total size of the aperture. This will cause an overall lowering of the coupled power from the transmit to the receive side.

Careful consideration must thus be made between the choice of beamformer ICs for dynamic range, transmit, and receive filters while placing additional constraints on the absolute transmitted power per element.

2.6.2 Radiation Pattern Masks

For practical implementation in broadband satellite communication the antenna must adhere to standards set by international organisations and satellite operators. One such standard places restrictions on the antenna beam shape, grating lobes, and side lobe levels (SLL) known as a radiation mask. This limitation is typically set separately for transmit and receive, where the transmitter masks are specified in terms of EIRP and the receiver masks in terms of antenna gain.

However, the masks are specified with the constraints and specifications stated by Table 1.1 and since our specifications are adjusted to better suit a smaller aperture area, given by Table 1.2, the masks are also adjusted. This is to accommodate a lower overall gain, higher relative sidelobes, and a wider main beam. Given below are the transmitter masks from ITU-R S.524-9 and 47 CFR §FCC 25.218(i), and the receiver mask from ETSI EN 301 459 both with their adjusted counterparts.

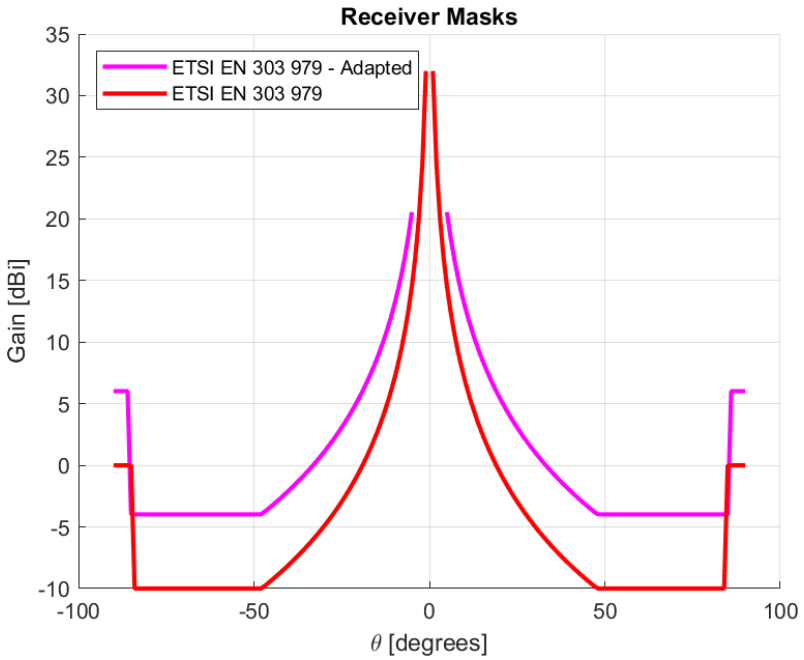


Figure 2.7: Receive Antenna Gain Masks

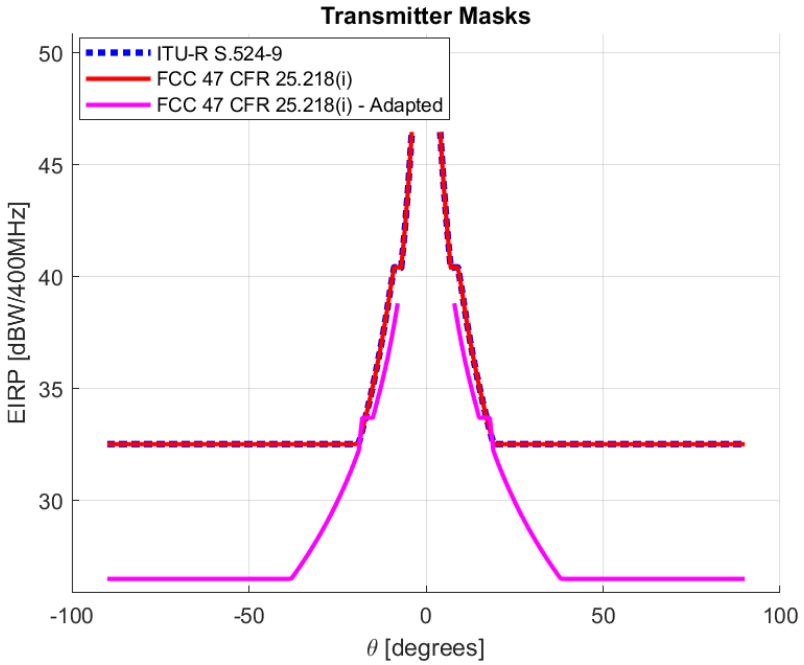


Figure 2.8: Transmitter EIRP Masks

From Figure 2.8 above it can be shown that when normalised to the same bandwidth the FCC and ITU masks are identical. Thus only one kind of adjusted mask is required for the transmitter.

2.6.3 Convex Problem Statements and Beamformers

In this work, the beamformer techniques were based on defining set performance requirements as convex optimisation problems. However, several challenges arose when implementing this in a convex solver in MATLAB (in this work, the CVX toolbox was used).

One such problem arose from the need to keep the desired beamformer ℓ_2-norm equal to 1. The reason for this was to avoid adding gain or attenuation to the system through the beamformers (which would in turn affect the solution space of the convex solver). However, the same beamformer is declared in the solver as a convex problem variable. Adding the constraint that a convex variable's ℓ_2-norm must be larger than a certain value breaks the rules of the convex solver. This condition is also true when constraining the ℓ_2-norm of the beamformer equal to 1.

The result of omitting this constraint is that the magnitude of the beamformer can be reduced to achieve the desired performance metrics. A work around to this problem is to loop the convex problem while normalising the beamformer after every solution, checking the conditions, and re-running the convex problem now comparing the beamformer to the previous normalised solution. The pseudo code below illustrates the concept (the full code is provided in Appendix D).

```
conditionsMet = false
while !conditionsMet
    minimise l2-norm( beamformerConditions - beamformerReference )
    subject to
        beamformerConditions * matrix <= conditions

    normalise( beamformerConditions )

    if ( beamformerConditions * matrix <= conditions )
        conditionsMet = true
    else
        beamformerReference = beamformerConditions
            or
        adjust conditions
```

This allows the solution to converge to the correct answer. The convergence can be aided by applying an offset to the conditions, or to adjust the conditions as the loop progresses. A final result is obtained by meeting the requirements, by setting the limit of iterations, or by comparing the resulting beamformer from iteration to iteration and stopping when the change between beamformers are negligible. The former method is referred to as a fixed iteration convex loop while the later is referred to as a convergence convex loop.

Lastly, the convex solver includes a Disciplined Convex Problem DCP [25] ruleset against which all convex problem statements are checked [26]. This ruleset is a set

of sufficient, but not necessary, rules for convexity. The challenge that this poses it that some problem statments may be rejected in the solver even if they are strictly convex. To overcome this, the problem needs to be restated in a way that adheres to the DCP ruleset.

3

Full Duplex Array System Modelling

This chapter describes the mathematical modelling of the antenna-transceiver system. First the modelling of the array antenna is discussed. This includes a motivation behind the element geometry and the array geometry chosen for this work. Next insight is given into the modelling of the transceiver system and components, as well as a description of the full duplex structure. Lastly, a description of how the system behaviour is modelled for both the transmit and receive scenarios (here described as the transmit and receive subsystems) is provided. Some results that validate the modelling of the system behaviour are included in these sections. Several figures are used to show different representations of the antenna-transceiver system. Each figure is meant to describe a different perspective of the system, which emphasises some aspects thereof while abstracting away from others. These provide a high-level view that is meant to guide the reader through the details of the modelling process.

3.1 Array Antenna Modelling

The electromagnetic antenna results were calculated from a Method-of-Moments (MoM) solver in the CAESAR simulation tool developed in [2].

The MATLAB code is developed such that various element geometries (including the port geometry) and various array geometries (such as linear, rectangular, or hexagonal) can be specified. Based on the element and array geometry, a mesh is generated per frequency for a specified number of points over a frequency bandwidth. Note that all of the antenna and system calculations are carried out for each of these specified frequency points.

The meshed array is used to calculate the admittance matrix of the antenna as well the short-circuited embedded element patterns. This is done by exciting each port of the antenna with a voltage signal while all the other ports are short-circuited. The admittance matrix is the ratio of the resulting total current through the port to the magnitude of the voltage excitation over the port. In this way, the mutual coupling between the antenna elements is taken into account.

3.1.1 Element Geometry

Simple metal strip dipoles are used as array antenna elements for this project. They are chosen for their simple geometry and their ease of modelling. However, it is noted that the relative bandwidth of half-wave dipoles (in which they are well matched and effective radiators) is 5-15%, depending on their exact geometry. This limits the frequency ratio between the Rx and Tx bands modelled in this system. A specialized dual-band element must be designed to study the performance of the system at the operational frequency specified in Table 1.1, which is outside the scope of this work.

Instead, the Rx and Tx frequencies are chosen to be 26 and 29 GHz respectively. These frequencies lie on the edges of the K- and Ka-bands, but are close enough to fall within the typical 10% bandwidth of a strip dipole. The dipoles are modelled such that they are half-wave at 29 GHz. This geometry was chosen specifically to avoid grating lobes in the Tx band. Whether grating lobes are present or not depends on the array geometry (specifically the inter-element spacing), but in the case of half-wave dipole array elements, the spacing between elements is limited by the element size. Therefore, to allow for an inter-element spacing that meets the grating lobe criteria in both Rx and Tx bands, the dipoles must be small enough at the higher frequency band (Tx) to allow for half-wave spacing.

Although wideband or dual-band planar antenna elements exist [27] [28] [29], most of these rely on dielectrics within their structure. However, the MoM solver used in this project cannot process dielectric structures, and therefore these designs were disregarded. Some fully metalized K-/Ka-band elements were also considered [30] [31], but were ultimately disregarded due to challenging design of a realisable feed network within the fully metallic structure that allows per element amplitude and phase control. Although the performance of the system with dipole elements is not representative of a practical system with specialized elements, it is still useful in that it allows us to design optimisation algorithms, draw conclusions about system performance, and identify critical trade-offs.

3.1.2 Array Geometry

A square aperture array geometry is chosen in alignment with the available aperture area for the SatCom terminal, as specified in Table 1.1. To fully utilize the area, a 54x54 element array is required. However, simulating an array of this size with the given MoM solver is both RAM and time intensive. Instead, it was elected to simulate a 16x16 element array for this project, which is sufficient to evaluate general system performance and identify critical trade-offs.

The elements are spaced half a wavelength apart in the y-dimension, and 0.52 wavelengths apart in the x-dimension (such that the adjacent elements do not touch one another.)

3.2 Transceiver Circuit Modelling

The transceiver structure for this project is based on a proposed shared aperture system for SatCom terminals that is being developed for the company Satcube. It consists of both an Rx and a Tx subsystem that are connected through an ideal power splitter to the antenna elements, as shown in Figure 3.1.

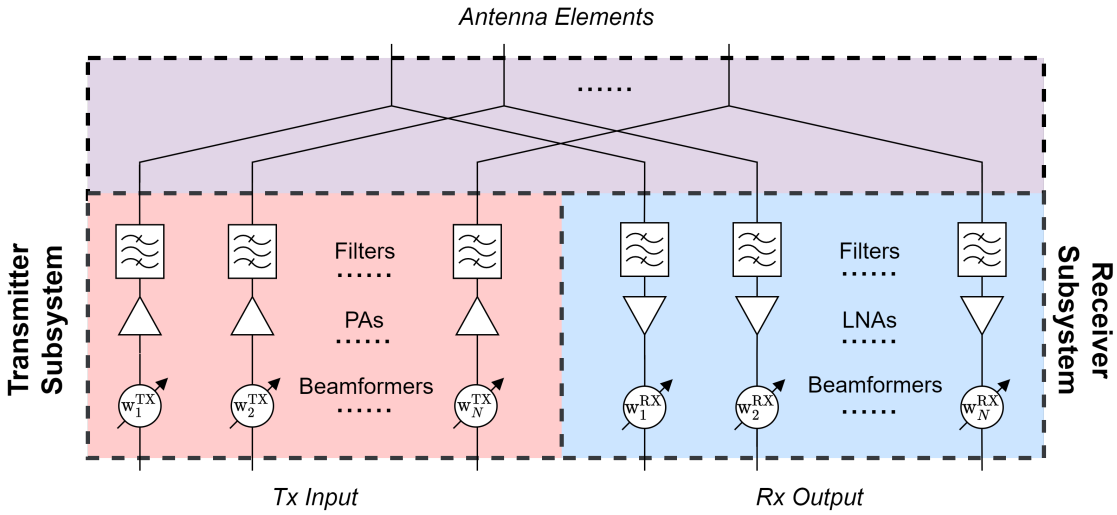


Figure 3.1: The general structure of the modelled system and the way in which the various components / subsystems are connected is illustrated.

This structure is chosen to try and minimize the space required for the transceiver circuitry such that the spacing of the antenna elements is not affected. It is worth noting that the power splitter that connects the Rx and Tx subsystems is modelled as an ideal, lossless parallel connection with a flat frequency response. This is done to model a "worst case scenario" in terms of coupling between the subsystems, as practical power splitters could offer some isolation between the two ports.

3.2.1 Component Modelling

The individual components of the transceiver system are modelled in MATLAB by making use of the Microwave System solver developed in [2]. This allows the S-parameters and noise performance of each component to be specified for any arbitrary frequency range. For this project, the components are assumed to be noiseless and to have a flat frequency response withing the Rx and Tx bands respectively. Furthermore, it is assumed that there is no coupling between the electronics of adjacent transceiver channels. This is done to simplify the development and validation of the system. However, the modification of the system model to include a varied frequency response, noisy components, or coupling between transceiver channels is arbitrary - in fact, the framework for this is already present in the CEASAR software package.

The S-parameters of the power amplifiers and low noise amplifiers were based on the quoted performance of existing on-chip SatCom components available in the K-/Ka-bands as provided by Sivers Semiconductors. Note that the focus here was not to provide a completely accurate model of each component, but instead to gain an understanding of the system behaviour if when active components are included in the analysis. The component models can be expanded in future work to provide a more accurate characterisation.

Based on the performance requirements given in Tables 1.1 and 1.2 and taking the array size of the system into account, the requried input dynamic range of the LNAs and the required output power of the PAs can be estimated. This is done by considering the required gain and EIRP of the system and calculating the required power per channel based on the array size. Based on this, the required filtering is estimated such that the output of the PAs does not saturate the LNAs. The required filtering is split equally between the Rx and Tx filters. For a realisable system, planar passive filters that are space efficient will be required. Although this will certainly require a concerted design effort, it is beyond the scope of this thesis. Instead, it is assumed that the required filtering can be achieved through the use of microstrip filters (or similar).

Lastly, the beamformers are modelled as ideal, impedance-matched and lossless components. They are also assumed to have an infinite phase and amplitude resolution. This is done to allow the development of a beamformer optimisation algorithm without phase or amplitude restraints. From the results of this optimisation algorithm it will be evaluated whether an implementation in practical beamformer chips with limited resolution is achievable.

The S-parameters of the components used in the modelling are summarised in the Table 3.1 below. Note that the values are defined for a reference impedance of 50 Ω, and are valid only for 400 MHz bandwidth in the Rx and Tx bands respectively.

Table 3.1: Scattering Wave Parameters of Transceiver Components [dB, rad]

Component	Rx Band Performance	Tx Band Performance
LNA	$\begin{bmatrix} -20 & -50 \\ 20 & -15 \end{bmatrix}$	$\begin{bmatrix} -10 & -50 \\ 18 & -10 \end{bmatrix}$
PA	$\begin{bmatrix} -3 & -50 \\ 20 & -3 \end{bmatrix}$	$\begin{bmatrix} -20 & -50 \\ 28 & -10 \end{bmatrix}$
Rx Bandpass Filter	$\begin{bmatrix} -10\angle 1.9 & -0.5\angle 0.3 \\ -0.5\angle 0.3 & -10\angle 1.9 \end{bmatrix}$	$\begin{bmatrix} -0.05\angle 1.9 & -20\angle 0.3 \\ -20\angle 0.3 & -0.05\angle 1.9 \end{bmatrix}$
Tx Notch Filter	$\begin{bmatrix} -0.05\angle 1.9 & -20\angle 0.3 \\ -20\angle 0.3 & -0.05\angle 1.9 \end{bmatrix}$	$\begin{bmatrix} -10\angle 1.9 & -0.5\angle 0.3 \\ -0.5\angle 0.3 & -10\angle 1.9 \end{bmatrix}$

3.3 Complete Microwave System Model

The signal interactions between the antenna and the receiver/transmitter subsystems, as well as the internal interactions of the electronic transceiver components, are calculated using scattering matrix representations. This is done through the combined electromagnetic-microwave simulation tool (CAESAR) developed in [2]. Notably, the beamformers are excluded from this microwave model and are instead modelled as ideal components without reflections. Figure 3.2 is adapted from Figure 3.1 to illustrate how the model is implemented in MATLAB, and in particular how the port number and a- and b-waves are defined.

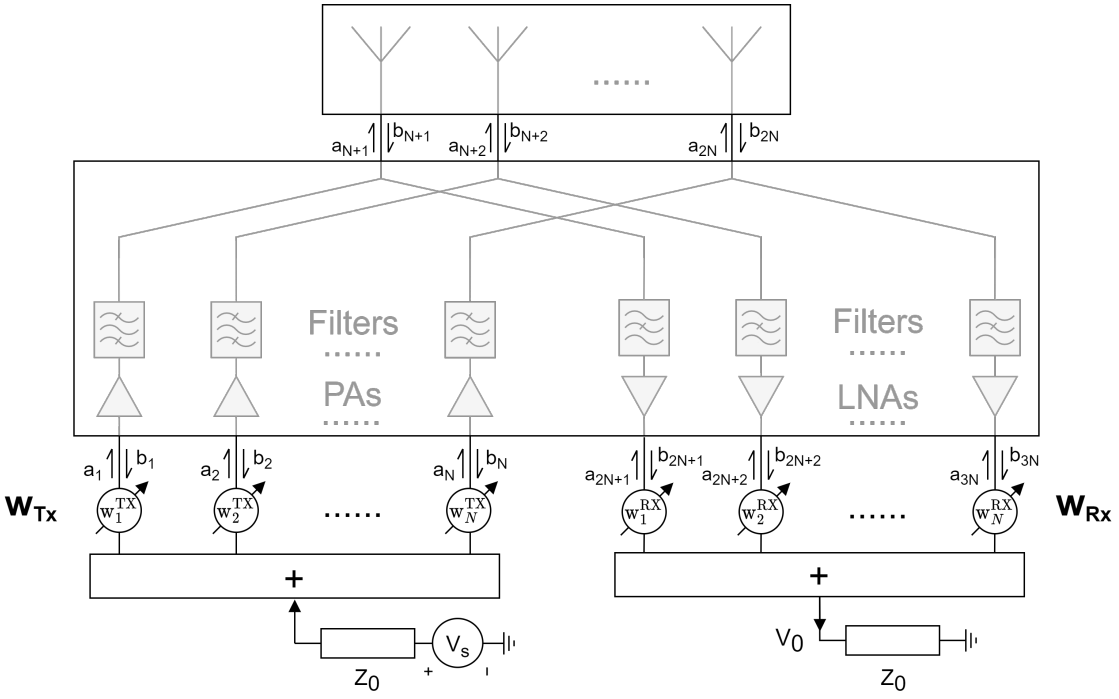


Figure 3.2: A microwave view of the full transceiver system. Note the port numbering, which is defined for the Rx and Tx ports, as well as the antenna ports. The transceiver channels can be modelled as a single device in MATLAB

This model is useful to calculate the system response over a set frequency bandwidth as seen from the Rx and Tx ports respectively. In addition, the element patterns under realistic loading conditions (hereafter referred to as the loaded embedded element patterns, or $\mathbf{EEP}^{\mathrm{Zg}}$), which consider not only the antenna but also the effects of the transceiver system as a whole, can be determined from this model.

3.4 Transmitter Subsystem Modelling

To consider the system from the transmitter beamformer perspective, the model from Figure 3.2 must be adjusted. In this new model, the antenna, the internal

microwave interactions and the Rx subsystem are abstracted into a "black box" that is connected to the Tx beamformers, as shown in Figure 3.3.

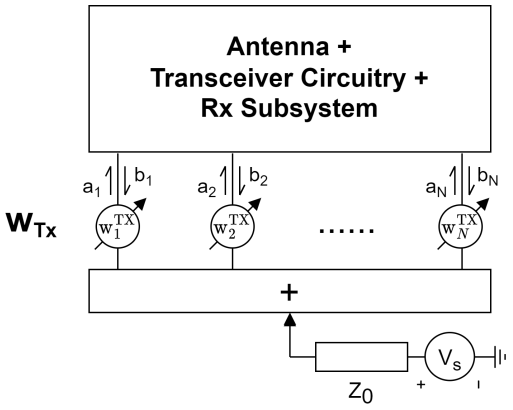


Figure 3.3: The abstracted transmitter system. This view of the system is useful for developing beamformer techniques to optimise the system performance in the transmit scenario.

With this model, we can account for the loading of the antenna by the Rx subsystem and transceiver circuitry as well as the effects of impedance mismatches and reflections.

3.4.1 Loaded Embedded Element Patterns

With this model, the loaded embedded element patterns with reference to the transmitter beamformers can be calculated. First, the transmitter and receiver ports are terminated in matched loads. The transmitter ports are then excited independently with a specified power excitation. For each independent power excitation, the resulting voltage excitations at the array antenna ports are measured and stored. These are then multiplied with the $\mathbf{EEP}^{\mathrm{sc}}$ of the antenna to find the resulting E-field pattern, which is the $\mathbf{EEP}^{\mathrm{Zg}}$ corresponding to the transmitter port excitation. In this way, the reference plane for the embedded element patterns is effectively shifted from the input of the array antenna to the transmitter beamformers.

3.4.2 Transmitter Gain Calculation

The transmitter gain calculation is based on the abstracted model shown in Figure 3.3. Using this simplified model, the gain calculation is similar to the one described in equation 2.10. Note, however, that the beamformer no longer describes a voltage signal, but a power signal (or an outgoing b-wave as used in the signal vector representation). Therefore the input power is not calculated as $\frac{1}{2}(\mathbf{w^H} \cdot \mathrm{Re}\{\mathbf{Y_{ant}}\} \cdot \mathbf{w})$, but simply as $\frac{1}{2}\mathbf{w}^H\mathbf{w}$. The radiated power is calculated by multiplying the $\mathbf{EEP^{sc}}$ with the voltages on the antenna ports.

The resulting gain represents the transmitter system gain, which can be decomposed into the loaded antenna gain and the microwave system gain.

3.4.3 Validation

To verify the accuracy of the modelled transmitter subsystem, the following outputs were analysed: the $\mathbf{EEP}^{Zg}$ of the antenna array, and the radiation pattern of the array when loaded with the transceiver system for a given beamformer.

A simple 3-element array consisting of 20 GHz half-wave dipoles was built in Ansys HFSS to provide a reference point against which the MATLAB antenna results could be compared.

3.4.3.1 Embedded Element Patterns

The $\mathbf{EEP}^{Zg}$ and gain patterns for both models are shown in Figures 3.4-3.5. These were calculated by exciting a single element with a source wave of 1 W amplitude, while the ports of both models were terminated in 50 Ω loads.

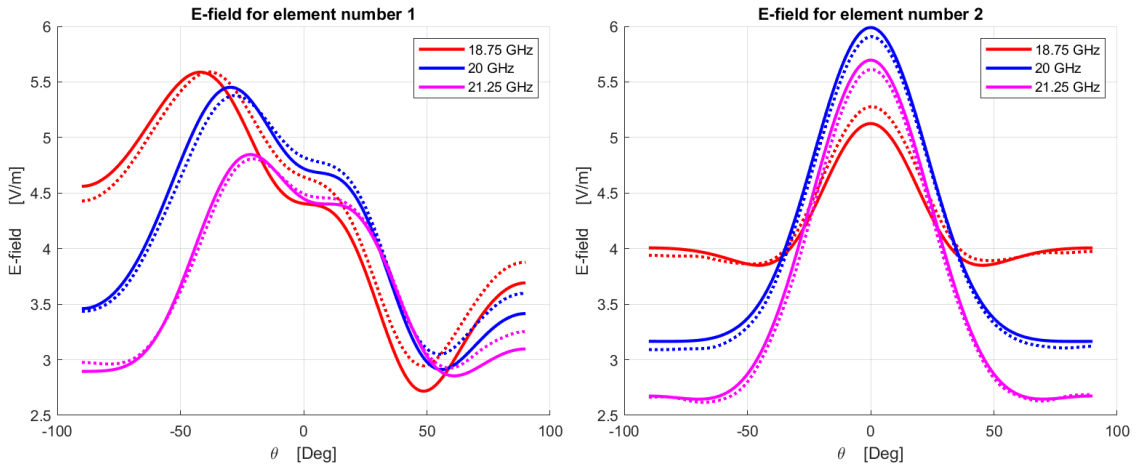


Figure 3.4: Embedded element E-field pattern results from MATLAB (solid line) and Ansys HFSS (dotted line)

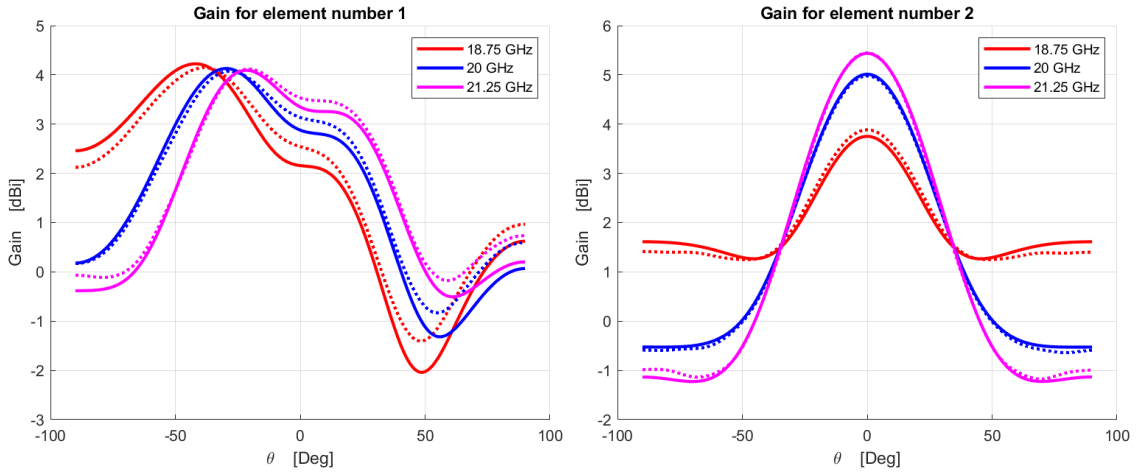


Figure 3.5: Embedded element gain pattern results from MATLAB (solid line) and Ansys HFSS (dotted line)

The graphs show the excellent correlations between the Ansys HFSS and MATLAB results. The discrepancies between the two results are likely due to the fact that the meshing is not optimzed in either case. Furthermore, the MATLAB results are based on a MoM solver, while Ansys HFSS uses a Finite Element Method (FEM) field solver. Although better correlation could be achieved with finer meshing, these initial results are adequate to validate that the $\mathbf{EEP}^{\mathrm{Zg}}$ and gain calculation methodology and implementation is correct in the MATLAB transmitter model.

3.5 Receiver Subsystem Modelling

To analyse the receiver subsystem, it is useful to modify the system model from Figure 3.2 to consider the system form the receiver beamformer perspective. In this new model, the transceiver circuitry, the Tx subsystem and the antenna are abstracted into a "black box", as shown in Figure 3.6.

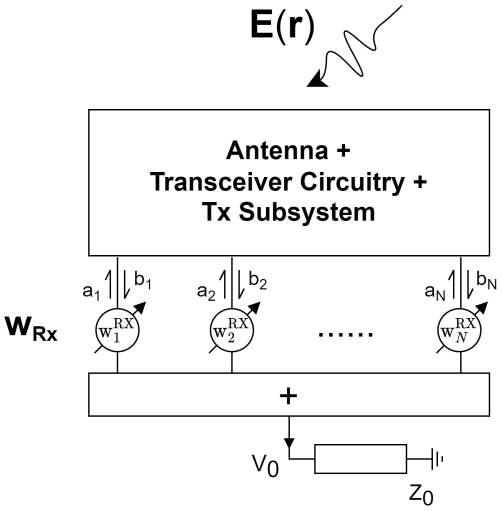


Figure 3.6: The abstracted receiver system. This model is useful for optimising the performance of the system in the receive scenario.

The system is analyzed only from the accessible ports (the Rx beamformers), where the incoming signals are used to calculate optimal beamformer distributions.

3.5.1 Incident Plane Wave Analysis

To characterize the receiving system, we start by considering a completely polarized plane wave incident on the array antenna from a direction $\hat{\mathbf{r}}_{\mathrm{i}}$ such that

$$\mathbf{E}^{\mathrm{i}}(\mathbf{r}_{\mathrm{i}}) = \mathbf{E}_0 e^{j\mathbf{k}_0 \cdot \mathbf{r}_{\mathrm{i}}} \tag{3.1}$$

Ultimately, this plane wave will be related directly to the induced excitation at the receiver beamformers, as shown in Figure 3.6. However, to understand how this

incident excitation propagates through the system, we start by first considering the complete microwave model, as shown in Figure 3.2.

The incident excitations are determined firstly by the interaction of the array antenna with the incident plane wave. Using the principle of antenna reciprocity, the open-circuited voltage on the n^{th} antenna element port can be determined through the projection [32], [33]:

$$V_n^{\text{oc}} = \frac{1}{j\omega\mu_0}\mathbf{E}_0 \cdot \mathbf{e}_n^T(\hat{\mathbf{r}}_\text{i}) \tag{3.2}$$

where μ_0 refers to the permeability of free space, $\mathbf{E}_0$ describes the polarity of the incident wave, and $\mathbf{e}_n(\hat{\mathbf{r}}_\text{i})$ refers to the n^{th} normalized far-field element pattern, which is based on the open-circuited embedded element pattern ($\mathbf{EEP}^{\text{oc}}$) calculated for the transmit case. The superscript ^{T} here denotes the transpose of the vector. Mathematically, this is written as:

$$\mathbf{e}^T = 4\pi r e^{jk_0 r}\mathbf{EEP}^{\text{oc}} \tag{3.3}$$

The $\mathbf{EEP}^{\text{oc}}$ can be readily calculated from the $\mathbf{EEP}^{\text{sc}}$ provided by the the MoM solver.

$$\mathbf{EEP}^{\text{oc}} = \frac{I_0}{V_0}\mathbf{Z}_\text{A}\mathbf{EEP}^{\text{sc}} \tag{3.4}$$

where I_0 and V_0 refer to the current and voltage excitations used to calculate the respective **EEP**s, and $\mathbf{Z}_\text{A}$ refers to the antenna input impedance matrix.

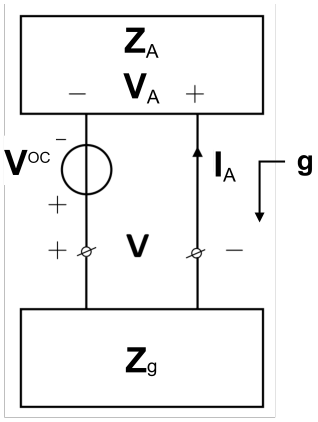


Figure 3.7: A circuit representation of a loaded receiving array antenna [2]

Once the V^{oc} are known for each antenna port, the output signal-wave vector which arises due to the incident wave, denoted by $\mathbf{g}$, can be derived as shown in [2] (reproduced here for clarity). With reference to Figure 3.7 and Kirchoff's voltage law, it is clear that:

$$\mathbf{V}^{\text{oc}} = \mathbf{V} + \mathbf{Z}_\text{A}\mathbf{I}_\text{A} \tag{3.5}$$

Furthermore, assuming real-valued and equal normalization impedances the equation can be rewritten as:

$$\mathbf{V}^{\text{oc}} = \sqrt{Z_0^{\text{ref}}}(\mathbf{a}_\text{L} + \mathbf{b}_\text{L}) + \frac{\mathbf{Z}_\text{A}}{\sqrt{Z_0^{\text{ref}}}}(\mathbf{a}_\text{L} - \mathbf{b}_\text{L}) \tag{3.6}$$

where the subscript in $_{\mathrm{L}}$ indicates that the a- and b-waves are defined with regard to the load $\mathbf{Z}_{\mathrm{s}}$. From this, the value for $\mathbf{a}_{\mathrm{g}}$ can be isolated as follows:

$$\begin{aligned}\mathbf{a}_{\mathrm{L}} &= (\mathbf{Z}_{\mathrm{A}} - \mathbf{Z}_0^{\mathrm{ref}})(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_0^{\mathrm{ref}})^{-1}\mathbf{b}_{\mathrm{L}} + (\sqrt{\mathbf{Z}_0^{\mathrm{ref}}}\mathbf{V}^{\mathrm{oc}})(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_0^{\mathrm{ref}})^{-1} \\ &= \mathbf{S}_{\mathrm{A}}\mathbf{b}_{\mathrm{L}} + \sqrt{\mathbf{Z}_0^{\mathrm{ref}}}\mathbf{V}^{\mathrm{oc}}(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_0^{\mathrm{ref}})^{-1}\end{aligned} \tag{3.7}$$

where the diagonal matrix $\mathbf{Z}_0^{\mathrm{ref}}$ contains the characteristic impedances of the transmission lines between the antenna and the load.

Lastly, because $\mathbf{a}_{\mathrm{L}} = \mathbf{b}_{\mathrm{A}}$ and $\mathbf{b}_{\mathrm{L}} = \mathbf{a}_{\mathrm{A}}$ in this system (where the subscript $_{\mathrm{A}}$ indicates that the a- and b-waves are defined with regard to the array antenna), equation 3.7 can be rewritten as $\mathbf{b}_{\mathrm{A}} = \mathbf{S}_{\mathrm{A}}\mathbf{a}_{\mathrm{A}} + \sqrt{\mathbf{Z}_0^{\mathrm{ref}}}\mathbf{V}^{\mathrm{oc}}(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_0^{\mathrm{ref}})^{-1}$. From this, we define $\mathbf{g}$ as the component of the outgoing signal-wave vector $\mathbf{b}_{\mathrm{A}}$ that is induced by $\mathbf{V}^{\mathrm{oc}}$, i.e.

$$\mathbf{g} = \sqrt{\mathbf{Z}_0^{\mathrm{ref}}}\mathbf{V}^{\mathrm{oc}}(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_0^{\mathrm{ref}})^{-1} \tag{3.8}$$

Conceptually, this derivation illustrates that $\mathbf{g}$ is added to the reflected signal from the antenna and is independent of $\mathbf{Z}_{\mathrm{g}}$ [2].

For the abstracted system model of Figure 3.6, the relationship between the incident plane and the resulting signal at the beamformers is of interest. To calculate this result, the signal $\mathbf{g}$ is added to the signal-wave vector representation of the microwave system in MATLAB as an internal source, which is then propagated to the beamformers. For a complete result, $\mathbf{g}$ is calculated per orthogonal polarization of the incident plane wave and stored independently.

3.5.2 Embedded Element Patterns

As with the transmit case, the $\mathbf{EEP}^{\mathrm{Zg}}$ with the reference plane at the beamformers are calculated. To define these $\mathbf{EEP}^{\mathrm{Zg}}$, we first expand further on the relationship between the plane-wave induced voltages on the system and the related **EEP**s. From Section 2.2.1, it is known that

$$\mathbf{EEP}^{\mathrm{oc}} = \frac{I_0}{V_0}(\mathbf{Z}_{\mathrm{A}} + \mathbf{Z}_{\mathrm{g}})\mathbf{EEP}^{\mathrm{Zg}} \tag{3.9}$$

Note that, for the normalized **EEP** case that is used in this analysis, $I_0 = 1$ A and $V_0 = 1$ V. Furthermore, using the voltage division rule together with Figure 3.7, we can write that

$$\mathbf{V} = \mathbf{V}^{\mathrm{oc}}\mathbf{Z}_{\mathrm{g}}(\mathbf{Z}_{\mathrm{g}} + \mathbf{Z}_{\mathrm{A}})^{-1} \tag{3.10}$$

By substituting in equations 3.2 and 3.9, and by only considering the polarisation case $\hat{p}$ as defined by $\mathbf{E}_0$, the relationship between the voltage over the loaded system

and the loaded embedded element patterns can be derived as follows:

$$\begin{aligned}\mathbf{V} &= \frac{4\pi r e^{jk_0 r}}{j\omega\mu_0}\hat{p}\cdot\mathbf{EEP}^{\mathrm{oc}}\mathbf{Z}_{\mathrm{g}}(\mathbf{Z}_{\mathrm{g}}+\mathbf{Z}_{\mathrm{A}})^{-1} \\ &= \frac{4\pi r e^{jk_0 r}}{j\omega\mu_0}\hat{p}\cdot(\mathbf{Z}_{\mathrm{A}}+\mathbf{Z}_{\mathrm{g}})\mathbf{EEP}^{\mathrm{Zg}}\mathbf{Z}_{\mathrm{g}}(\mathbf{Z}_{\mathrm{g}}+\mathbf{Z}_{\mathrm{A}})^{-1} \\ &= \frac{4\pi r e^{jk_0 r}}{j\omega\mu_0}\hat{p}\cdot\mathbf{EEP}^{\mathrm{Zg}}\mathbf{Z}_{\mathrm{g}}\end{aligned} \tag{3.11}$$

Therefore each of the loaded voltages can be directly related to the corresponding $\mathbf{EEP}^{\mathrm{Zg}}$, appropriately scaled and evaluated at the relevant angle [1]. Inversely, if the loaded voltages per orthogonal polarization at the receiver beamformers are known, the corresponding $\mathbf{EEP}^{\mathrm{Zg}}$ per polarization can be calculated as follows:

$$\mathbf{EEP}^{\mathrm{Zg}}_{\hat{p}} = \frac{j\omega\mu_0}{4\pi r e^{jk_0 r}}\mathbf{Z}_{\mathrm{g}}^{-1}\mathbf{V}_{\hat{p}} \tag{3.12}$$

In this way, the loaded voltages at the receiver beamformers can be related to the corresponding $\mathbf{EEP}^{\mathrm{Zg}}$ of the abstracted system in Figure 3.6, if the impedance matrix of the system is known.

3.5.3 Receiver Gain Calculation

When considering the full transceiver system with active components, the system is no longer reciprocal. Hence the method used for determining the system gain for the transmit scenario cannot be used for the receive scenario. Instead, the receiver system gain is calculated by considering plane wave sources at every desired ϕ and θ scan angle, and calculating the corresponding received power at the receiver beamformer ports.

Once these values are known, a mathematical expression for the receiver system gain is required. We start by considering the gain expression for a receive antenna, which is a function of its effective aperture area:

$$G(\phi,\theta) = A_{\mathrm{e}}\frac{4\pi}{\lambda^2} \tag{3.13}$$

However, the effective aperture area of an array antenna is not easily calculated. Instead, it can be described as the power ratio between output and input power of the antenna:

$$A_{\mathrm{e}} = \frac{P_{\mathrm{rec}}(\phi,\theta)}{S_{\mathrm{planeWave}}} \tag{3.14}$$

where $S_{\mathrm{planeWave}}$ is the power flux density of the incident plane wave. By substituting equation 3.14 into equation 3.13, we can redefine the gain as:

$$\begin{aligned}G(\phi,\theta) &= \frac{P_{\mathrm{rec}}(\phi,\theta)}{S_{\mathrm{planeWave}}}\frac{4\pi}{\lambda^2} \\ &= \frac{P_{\mathrm{rec}}(\phi,\theta)2\eta}{|E_{\mathrm{planeWave}}|^2}\frac{4\pi}{\lambda^2} \\ &= \frac{8\pi\eta P_{\mathrm{rec}}(\phi,\theta)}{|E_{\mathrm{planeWave}}|^2\lambda^2}\end{aligned} \tag{3.15}$$

Since the amplitude of the incident plane wave can be chosen within the mathematical model, it is set to 1 V/m. This allows equation 3.15 to be simplified to:

$$G(\phi, \theta) = \frac{8\pi\eta}{\lambda^2} P_{\text{rec}}(\phi, \theta) \tag{3.16}$$

This gain definition assumes that the incident wave is polarisation matched to the antenna, and that the antenna is terminated in conjugate matched loads (i.e. that no reflections are present).

We note that, by choosing to define P_{rec} at the receiver ports instead of the antenna ports, equation 3.16 can also be used to describe the receive system gain. In accordance with the signal wave model, the beamformed output power at the receiver outputs can be calculated as:

$$P_{\text{rec}}(\phi, \theta) = \frac{1}{2}(\mathbf{w}^T \mathbf{b}(\phi, \theta)\mathbf{b}^H(\phi, \theta)\mathbf{w}^*) \tag{3.17}$$

where $\mathbf{b}$ refers to the b-waves present at the receiver ports. In this equation these signal waves are assumed to be peak values and thus the division by 2 is required in this multiplication to produce RMS values. No a-waves are considered in this equation due to the fact that the beamfomers are assumed to be ideal and reflectionless (i.e. $\mathbf{a} = 0$).

By using the signal wave model, we can take the impedance mismatches of the system into account. Furthermore, if signal vectors $\mathbf{g}$ that arise due to the incident plane waves are calculated for x, y and z polarisations respectively, they can be used to calculate the full polarisation gain of the system. In this way the limitations of equation 3.16 are overcome

3.5.4 Validation

To verify the accuracy of the modelled receiver subsystem, the following outputs were considered: the plane-wave induced open circuit voltages on the array antenna and the $\mathbf{EEP}^{\text{Zg}}$ of the subsystem with reference to the voltages at the receiver beamformers.

3.5.4.1 Induced Open-Circuit Voltages

Apart from the method presented in Section 3.5.1, the induced open-circuit voltages on an antenna element can also be calculated analytically based on the effective aperture of the antenna. For the simple case of a half-wave dipole, V^{oc} is calculated as follows:

$$V^{\text{oc}} = \mathbf{E}^{\text{i}} \cdot \boldsymbol{\ell}_{\text{eff}} \tag{3.18}$$

where $\boldsymbol{\ell}_{\text{eff}}$ is the effective length of the dipole.

This well-known result is used as a reference point against which the implementation of the method in Section 3.5.1 can be verified. Simulated results from Ansys

HFSS are also used for validation purposes. Figure 3.8 shows the magnitude of the simulated and calculated plane wave induced V^{oc} for a half-wave dipoles over a frequency range. The plane wave that was used to generate these results is incident from broadside. Note that each data point corresponds to a half-wave dipole at the specified frequency point. Furthermore, we can consider the V^{oc} for single half-wave dipole over a frequency range, as shown in Figure 3.9. Note that, for this comparison, the analytical results are omitted.

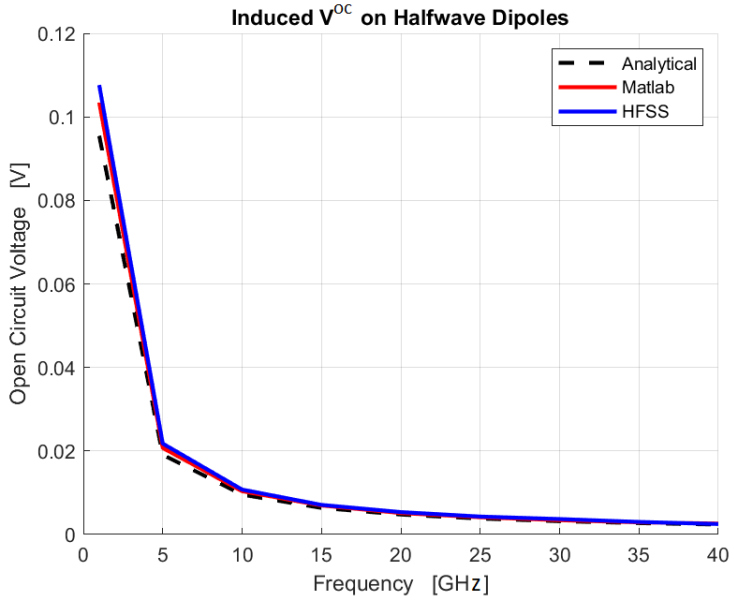


Figure 3.8: Plane wave induced open-circuit voltages on half-wave dipoles at various frequencies

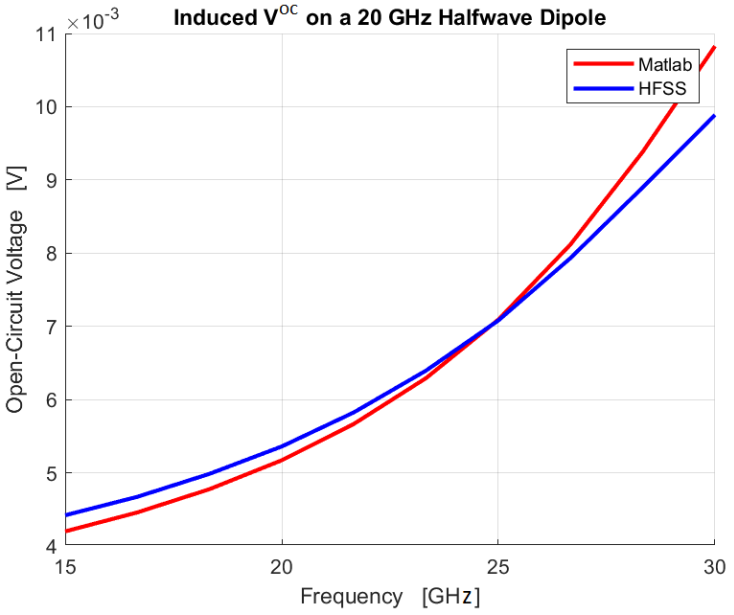


Figure 3.9: Plane wave induced open-circuit voltages on a 20 GHz half-wave dipole

Both of the previous figures show excellent correlation between the Ansys HFSS, MATLAB, and analytical results. Some discrepancy can be seen between the MATLAB and HFSS results in the higher end of the frequency range in Figure 3.9. It is important to note that both of these models were meshed for the 20 GHz case, so it is expected that the results start to diverge as we move away from the meshed frequency.

3.5.4.2 Embedded Element Patterns

To verify the loaded element pattern calculation methodology as presented in Section 3.5.2, the 3-dipole validation model used in Section 3.4.3 is revisited. In this case, the receive situation is considered. Since the model is completely reciprocal, it is expected that the $\mathbf{EEP}^{Zg}$ calculated using the transmit methodology should be exactly the same as the $\mathbf{EEP}^{Zg}$ calculated from the receive methodology. The two embedded element patterns are shown in Figure 3.10 below.

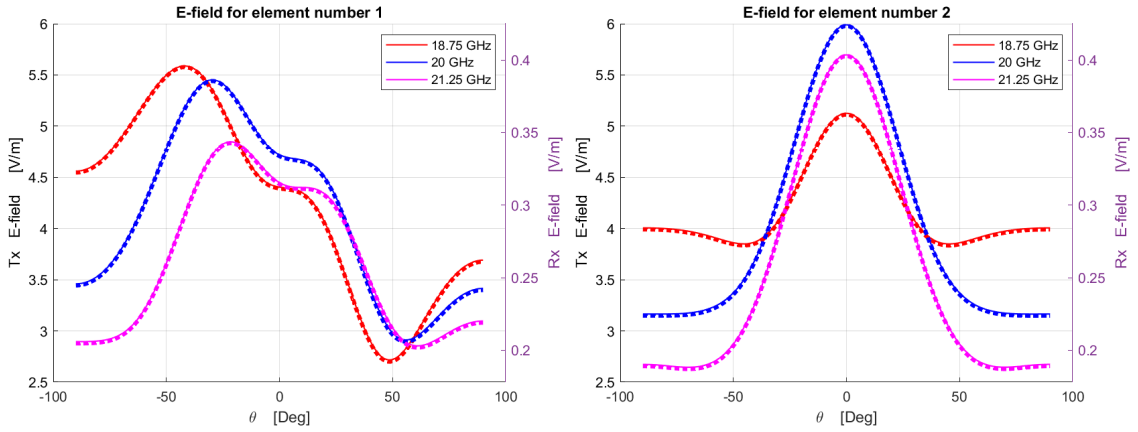


Figure 3.10: Embedded element E-field pattern results generated in MATLAB for the Tx subsystem (solid line) and the Rx subsystem (dotted line)

The shape of the two element patterns correspond exactly. It is worth noting that there is a scaling difference between the Tx and Rx case, however. This is because the patterns were generated with different excitation magnitudes. In the Tx case, a 1 W incident power wave is used, while the Rx case uses an incident plane wave of magnitude 1 V/m.

4

Beamformer Algorithms

This chapter provides an overview of the mathematical formulation of the beamformer coefficents for this system. First, a derivation of the maximum gain beamformers for an antenna as well as the maximum realised gain beamformers for an antenna-transceiver system are provided. Thereafter the algorithm for finding a desired beamformer that provides optimum gain while also providing a specified is discussed. Next, the process of designing a beamformer that ensures radiation mask adherence while also achieving optimum gain is described. Lastly, insight is given into how these three different beamforming goals are combined in a single algorithm.

4.1 Maximum Gain Transmit Beamformers

It is of interest to find a superdirective beamformer that provides the maximum output power or realised gain for the transceiver-antenna system. To do so, different beamformer formulations are considered.

For a given transmit antenna array, the maximum gain beamformer (MGB) excitations vector $\mathbf{w}$ (which is a more general representation than the voltage excitation $\mathbf{v}$ used in Section 2.2.3) can be determined by calculating the derivative of the gain equation with respect to $\mathbf{w}^H$ and setting it to zero. We first consider antenna voltage excitations, and correspondingly use the $\mathbf{EEP}^{\text{sc}}$. The resulting gain equation is given by equation 2.10.

To simplify the analysis, let $\mathbf{A}$ represent $\mathbf{EEP}^{\text{sc}}(\theta_0, \phi_0)^H \mathbf{EEP}^{\text{sc}}(\theta_0, \phi_0)$ and $\mathbf{B} = \text{Re}\{\mathbf{Y}_\text{A}\}$. Furthermore, the scaling constant $\frac{4\pi R^2}{\eta}$ is discarded (as it does not affect the final beamformer solution, which is normalised to achieve a ℓ_2-norm of 1). This results in a redefined gain equation as

$$G(\theta_0, \phi_0, \mathbf{w}, \mathbf{w}^H) = \frac{\mathbf{w}^H \mathbf{A} \mathbf{w}}{\mathbf{w}^H \mathbf{B} \mathbf{w}} \tag{4.1}$$

According to [34] if G is real valued function then setting $\nabla_{\mathbf{w}^H} G(\mathbf{w}, \mathbf{w}^H) = 0$ yields a $\mathbf{w}$ for which $\nabla_{\mathbf{w}} G(\mathbf{w}, \mathbf{w}^H)$ is also zero. Therefore, to calculate a $\mathbf{w}$ which maximises G, it suffices to consider $\nabla_{\mathbf{w}^H} G(\mathbf{w}, \mathbf{w}^H)$. From this, we differentiate the gain with

respect to $\mathbf{w}^H$ as follows:

$$\begin{aligned} \nabla_{\mathbf{w}^H} G(\theta_0, \phi_0, \mathbf{w}, \mathbf{w}^H) &= \frac{(\mathbf{w}^H\mathbf{B}\mathbf{w})(\mathbf{A}\mathbf{w}) - (\mathbf{w}^H\mathbf{A}\mathbf{w})(\mathbf{B}\mathbf{w})}{(\mathbf{w}^H\mathbf{B}\mathbf{w})^2} = 0 \\ \therefore (\mathbf{w}^H\mathbf{B}\mathbf{w})(\mathbf{A}\mathbf{w}) &= (\mathbf{w}^H\mathbf{A}\mathbf{w})(\mathbf{B}\mathbf{w}) \\ \mathbf{A}\mathbf{w} &= \frac{\mathbf{w}^H\mathbf{A}\mathbf{w}}{\mathbf{w}^H\mathbf{B}\mathbf{w}}\mathbf{B}\mathbf{w} = G(\theta_0, \phi_0, \mathbf{w})\mathbf{B}\mathbf{w} \end{aligned} \tag{4.2}$$

This result is of a generalised eigenvector form. Therefore, to maximise the gain we need to solve for the the maximum eigenvalue $G(\theta_0, \phi_0, \mathbf{w})$, and identify the corresponding eigenvector $\mathbf{w}$ as the optimal excitation for maximum gain. The general solution to this is given by

$$\mathbf{w_{MG}} = \mathbf{B}^{-1}\mathbf{EEP}^{\mathrm{sc}}(\theta_0, \phi_0) \tag{4.3}$$

This result shows that the maximum gain beamformer is a function of the **EEPs** as well as the pattern overlap integral. In fact, this result can be generalised to other beamformer excitations as well if the correct **EEP** and input power defintions are used. For example, to calculate the maximum gain beamformer current excitation for an open-circuited antenna system, the $\mathbf{EEP}^{\mathrm{oc}}$ should be used and $\mathbf{B}$ set to $\mathrm{Re}\{\mathbf{Z}_{\mathrm{A}}\}$.

In a similar way, the EIRP for an antenna-transceiver system can be maximised by calculating a beamformer that maximises the realised gain of the system (MRGB). First the reference plane of the $\mathbf{EEPs}^{\mathrm{Zg}}$ must be shifted from the antenna inputs to the Tx beamformers. Furthermore, instead of considering the input power (as in our gain calculation), we instead consider the incident power to calculate the maximum *realised* gain (see Section 2.1.1, Figure 2.3). The incident power is defined as

$$P_{\mathrm{inc}} = \frac{1}{2}\mathbf{w}^H\mathbf{w} \tag{4.4}$$

where $\mathbf{w}$ refers to the beamformed forward travelling voltage wave, as defined in Section 2.5. Therefore, to maximise EIRP, we consider the equation

$$\mathrm{EIRP}(\theta_0, \phi_0, \mathbf{w}) = \frac{\mathbf{w}^H\mathbf{A}\mathbf{w}}{\mathbf{w}^H\mathbf{w}} \tag{4.5}$$

In this case, $\mathbf{w}^H\mathbf{w}$ becomes independent of $\mathbf{w}$, assuming that its 2-norm value is constant (chosen to be unity, in this case). Therefore $\mathbf{w}$ is chosen to maximise $\mathbf{w}^H\mathbf{A}\mathbf{w}$ directly, which leads to the general solution

$$\mathbf{w_{MRG}} = \mathbf{EEP}^{\mathrm{Zg}}_{\hat{p}}(\theta_0, \phi_0)^* \tag{4.6}$$

This solution satisfies the conjugate field matching (CFM) conditions such that $\mathbf{w_{MRG}} = \mathbf{w_{CFM}}$.

Note that the maximum EIRP analysis for a realistically loaded antenna-transceiver system differs significantly from the maximum gain analysis for an antenna only

because the latter assumes ideal port terminations. It is interesting, however, to consider the analogous gain case for the loaded system, i.e. when the antenna ports are connected to the array transceiver with non-ideal loading. To do so, we consider $\mathbf{EEP}^{Zg}$ and the corresponding pattern overlap integral of the system, which is proportional to the the noise covariance matrix $\mathbf{C}$ (see Section 2.2.1). Substituting these values into equation 4.1 results in the equation that defines the SNR of the outgoing signal. This will be further elaborated on in Section 4.2.

4.1.1 System Implementation

For the transmit case, the beamformers are chosen to maximise EIRP. In the modelled transmitter subsystem, shown in Figure 3.3, the $\mathbf{EEPs}^{Zg}$ are calculated with reference to the transmitter beamformers, as described in Section 3.4.1. A normalized incident power of 1 W is applied at the Tx beamformer input. The optimum beamformer excitations are chosen according to equation 4.6.

4.2 Maximum Gain Receive Beamformers

For the receive antenna, two superdirective beamformer realizations are considered, namely the conjugate field matched (CFM) beamformer and the maximum signal to noise ratio (SNR) beamformer. To do so, we consider the entire transceiver system from its accessible ports (the Rx beamformer ports), as shown in Figure 4.1.

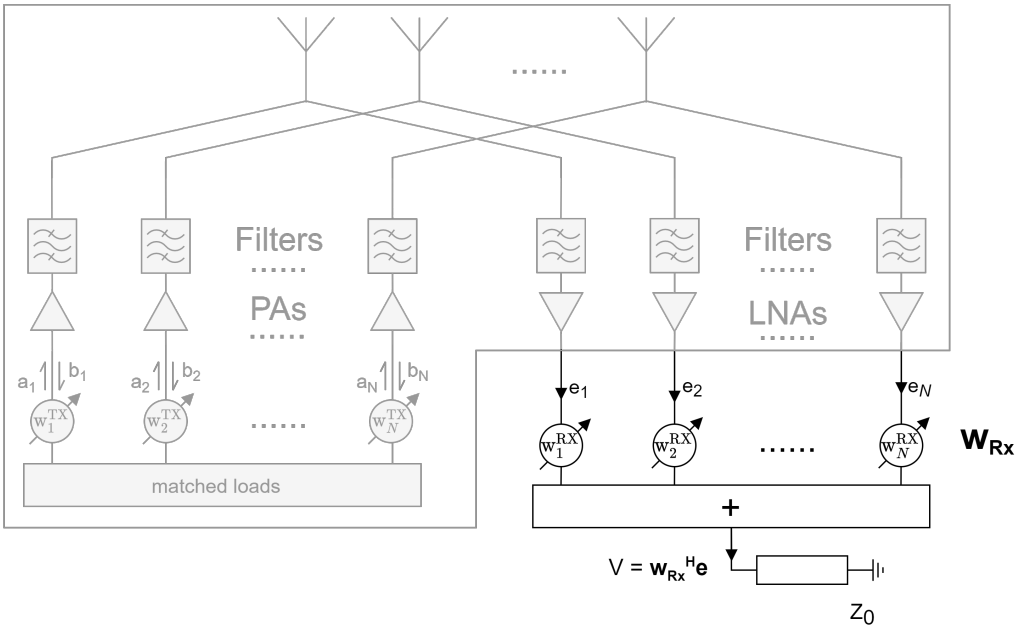


Figure 4.1: Generalised representation of the transceiver system from the receiver beamformer perspective

To maximise the sensitivity metric of a receive system, defined as the effective area of the antenna system divided by the system equivalent noise temperature ($A_{\text{eff}}/T_{\text{sys}}$),

should be maximised. This metric can be defined in terms of SNR as follows [13]:

$$\frac{A_{\mathrm{e}}}{T_{\mathrm{sys}}} = \frac{2k_B}{S_{\mathrm{planeWave}}}\mathrm{SNR} \tag{4.7}$$

where k_B is Boltzmann's constant and $S_{\mathrm{planeWave}}$ is the flux density of the source that the antenna is detecting. Furthermore, the SNR of the system is defined as

$$\mathrm{SNR} = \frac{\mathbf{w}^H\mathbf{P}\mathbf{w}}{\mathbf{w}^H\mathbf{C}\mathbf{w}} \tag{4.8}$$

Where $\mathbf{P}$ is the signal wave covariance matrix and $\mathbf{C}$ is the noise wave covariance matrix of the system. It is clear that, for a given source flux density and noise temperature, the sensitivity of the system can be maximised by maximising the SNR of the system. It is worth pointing out that equation 4.8 is of the same form as 4.1. Therefore, without any further derivation required, we can conclude that the SNR of the system will be maximised for

$$\mathbf{w_{MaxSNR}} = \mathbf{C}^{-1}\mathbf{e} \tag{4.9}$$

where $\mathbf{e}$ is the resulting signal waves at the Rx beamformer outputs from a plane wave incident from (θ_0, ϕ_0) such that $\mathbf{P} = \mathbf{e}\mathbf{e}^H$.

When $\mathbf{C}$ equals the identity matrix $\mathbf{I}$, equation 4.9 reduces to the well-known CFM beamformer

$$\mathbf{w_{CFM}} = \mathbf{e} \tag{4.10}$$

Therefore the CFM beamformer is a specific case of the maximum SNR beamformer for a system with equal and uncorrelated noise output powers. As discussed in Section 4.1, the CFM beamformer maximises the realised gain of the transceiver-antenna system.

4.2.1 System Implementation

As for the transmit case, the receive beamformer weights are chosen to maximise EIRP. To obtain the CFM beamformer a system simulation is set up which applies a plane wave from the desired scan direction, and the resulting wave excitations on the receiver ports are recorded. The normalised array of these excitations provide the CFM beamformer coefficients.

4.3 Tx-Rx Isolation Beamformer Algorithm

Beamformer functionality in a full duplex system extends beyond attaining superdirectivity. They are also useful to achieve other performance parameters, such as increased isolation between the Rx and Tx subsystems.

When considering the full transceiver system, as shown in Figure 3.2, it can be observed that some power will leak through (couple) from the Tx subsystem and be

detected by the Rx subsystem. This will happen due to both the scattering from the array antenna, i.e.over-the-air coupling, as well as non-ideal isolation of the power splitter that physically connects the two subsystems. The coupling between the Rx and Tx subsystems can be quantified as a ratio of the power provided to the system at the Tx beamformer ports to the resulting power at the Rx beamformer output. This definition of isolation takes the entire transceiver system, including the electronic components and the array antenna, into account.

Some of this coupled power is reduced by the filters on both the Rx and Tx subsystems. As discussed in Section 3.2.1, these filter parameters are chosen to prevent the LNAs from becoming saturated. However, on-chip filter design at K/Ka-band is challenging, especially with the strict area constraints required for beamformer chips. Therefore, instead of imposing more stringent performance requirements on the filter, both the Rx and Tx beamformers are investigated to analyse their usefullness in providing improved isolation.

4.3.1 Isolation Definition

The beamformer input power at the Tx ports can be defined as:

$$P_{\text{in}} = \mathbf{w}_{\mathbf{Tx}}^{H}\mathbf{w}_{\mathbf{Tx}} \tag{4.11}$$

The resulting output power at the Rx beamformer output can be defined in terms of incident interference waves ($\mathbf{c}$) as follows:

$$\begin{aligned} P_0 &= |\mathbf{w}_{\mathbf{Rx}}^{T}\mathbf{c}|^2 = (\mathbf{w}_{\mathbf{Rx}}^{T}\mathbf{c})(\mathbf{w}_{\mathbf{Rx}}^{T}\mathbf{c})^{H} \\ &= \mathbf{w}_{\mathbf{Rx}}^{T}\mathbf{c}\mathbf{c}^{H}\mathbf{w}_{\mathbf{Rx}}^{*} \\ &= \mathbf{w}_{\mathbf{Rx}}^{H}\mathbf{c}^{*}\mathbf{c}^{T}\mathbf{w}_{\mathbf{Rx}} \end{aligned} \tag{4.12}$$

Note that the output power is a purely real value, such that computing the conjugate P_0 in the last analysis step does not affect the final value. This isolation between the Rx and Tx subsystems is defined as

$$\text{Isolation} = 10\log\left(\frac{P_{\text{in}}}{P_0}\right) \tag{4.13}$$

4.3.2 Optimum Isolation Beamformer

The optimum isolation beamformer seeks to find the Rx beamformer that provides the highest possible isolation against the coupled power from the Tx subsystem, while still maintaining an acceptable sensitivity in the Rx band. This requires first that an Rx beamformer vector $\mathbf{w}_{\mathbf{Rx}}$ be calculated with an acceptable gain. Thereafter, the isolation can be improved by analysing the derivative of the output power to determine if it is increasing or decreasing for the given $\mathbf{w}_{\mathbf{Rx}}$.

$$\nabla_{\mathbf{w}_{\mathbf{Rx}}^{H}} P_0 = \mathbf{c}^{*}\mathbf{c}^{T}\mathbf{w}_{\mathbf{Rx}} \tag{4.14}$$

From this, a new $\mathbf{w_{Rx\_new}}$ can be calculated as follows:

$$\mathbf{w_{Rx\_new}} = \mathbf{w_{Rx}} - \gamma \nabla_{\mathbf{w}_{\mathbf{Rx}}^H} P_0 \tag{4.15}$$

where γ can be chosen for a desired trade-off between Rx gain and isolation. A solution for optimum isolation (with a non-optimum gain) can be calculated by combining equations 4.15 and 4.12 and minimizing P_0 with respect to γ. To simplify the analysis, we let $\mathbf{A} = \mathbf{c}^*\mathbf{c}^T = \mathbf{A}^H$, and assume that γ is a purely real value. The coupled output with the new beamformer weights from 4.15 is defined as:

$$\begin{aligned} P_{0\_\text{new}} &= \mathbf{w}_{\mathbf{Rx\_new}}^H \mathbf{A} \mathbf{w_{Rx\_new}} \\ &= (\mathbf{w_{Rx}} - \gamma \nabla_{\mathbf{w}_{\mathbf{Rx}}^H} P_0)^H \mathbf{A} (\mathbf{w_{Rx}} - \gamma \nabla_{\mathbf{w}_{\mathbf{Rx}}^H} P_0) \\ &= (\mathbf{w}_{\mathbf{Rx}}^H \mathbf{A} \mathbf{w_{Rx}} - 2\gamma \mathbf{w}_{\mathbf{Rx}}^H \mathbf{A}^2 \mathbf{w_{Rx}} + \gamma^2 \mathbf{w}_{\mathbf{Rx}}^H \mathbf{A}^3 \mathbf{w_{Rx}}) \end{aligned} \tag{4.16}$$

By differentiating with respect to γ and setting the result equal to zero, an optimum γ for minimum coupled output power can be calculated.

$$\gamma = \frac{\mathbf{w}_{\mathbf{Rx}}^H \mathbf{A}^2 \mathbf{w_{Rx}}}{\mathbf{w}_{\mathbf{Rx}}^H \mathbf{A}^3 \mathbf{w_{Rx}}} \tag{4.17}$$

This analysis considers only the Rx beamformer. However, if the interference waves $\mathbf{c}$ are defined in terms of the Tx beamformer, a similar analysis can be conducted to determine an optimum $\mathbf{w_{Tx}}$ for isolation.

To test the robustness of this beamformer over a frequency bandwidth, an arbitrary transceiver system with a 64-element array antenna and component specifications as in Table 3.1 was modelled in MATLAB. The antenna consisted of halfwave dipoles resonant at 27.2 GHz, with the Rx band at 26 GHz and the Tx band at 29 GHz. An input power of 1 W was provided at the Tx beamformers. The optimum isolation beamformer was applied as specified above, with the isolation results shown in Figure 4.2.

From this figure, it is clear that the optimum isolation beamformer only provides optimum isolation at a single frequency point. The isolation at this point is infinity, but is shown to be around 300 dB in the plot due to frequency resolution. It is worth noting that the isolation is improved over the entire bandwidth when the beamformer coefficients as optimised at a single frequency are fixed within this bandwidth.

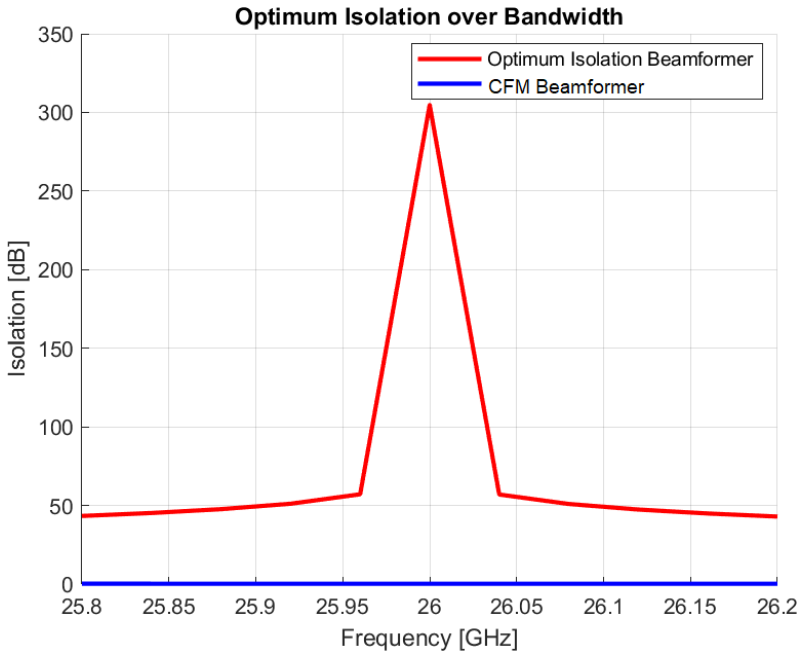


Figure 4.2: Achieved isolation over the Rx bandwidth with optimum isolation beamformer as compared to the maximum SNR beamformer

4.3.3 Isolation Over Bandwidth Beamformer

As discussed above, the method from Section 4.3.2 provides a very narrow-band optimum isolation solution. While it can be shown that the near perfect isolation achieved in this band, a more useful solution would be to achieve a specified isolation over a realistic bandwidth. In this section, an algorithm is described in which the target isolation over a bandwidth is achieved by altering the receive and transmit beamformer coefficients while also minimising the gain cost. This is done in a general convex problem statement as follows:

$$\begin{aligned} &\text{minimise}(\ell_2\text{-norm}(\mathbf{w_{isolation}} - \mathbf{w_{reference}})) \\ &\text{subject to} \\ &\quad |\mathbf{w}_{\mathbf{isolation}}^{H}\mathbf{c}| \geq \text{min isolation} \end{aligned} \tag{4.18}$$

where $\mathbf{c}$ refers to the interference waves present on the receiver ports due to the transmitter beamformer excitations, $\mathbf{w_{isolation}}$ is the calculated beamformer for increased isolation, and $\mathbf{w_{reference}}$ is the reference beamformer against which the system gain cost will be evaluated.

It is worth expanding on why the term $|\mathbf{w}_{\mathbf{isolation}}^{H}\mathbf{c}|$ is used in this statement. First, we consider the isolation as defined in equation 4.13. The input power P_{in} at the transmit beamformer is normalised, and therefore the isolation can be defined purely in terms of P_0. In linear units, the isolation is then $1/P_0$. However, the definition of P_0 requires a quadratic statement (given in equation 4.12), which violates DCP ruleset of the chosen convex solver. However, for a given load, it is known that

the power is proportional to the square of the voltage. Therefore the isolation is reformulated in terms of voltage, and given as $|\mathbf{w}_{\mathbf{isolation}}^{H}\mathbf{c}|$. Once $\mathbf{w}_{\mathbf{isolation}}$ has been determined from the algorithm, the resulting isolation in terms of power can be calculated and used to determine if the original target isolation is met.

4.3.4 Detailed Description and Method

A challenge of calculating a single beamformer solution that satisfies the isolation requirements over a frequency bandwidth is finding a way to include the frequency dependent characteristics of the system within the convex problem statement. First we will consider the case in which the receive beamformer is adjusted for increased isolation. In this case, the incident interference waves $\mathbf{c}$ at the receiver are predetermined as a two dimensional matrix containing the wave amplitudes per frequency. Multiplying these waves with the receiver beamformer will result in an isolation result per frequency, which can in turn be compared to an appropriately sized matrix containing the target isolation per frequency. This pseudo-code for this calculation is provided below.

```
rxInputs(frequencies, channels)
targetIsolation(frequencies)

rxInputs * beamformerRxIso <= targetIsolation
```

Next we consider how the transmit beamformer can be adjusted to increase isolation. This is a more involved process, as the interference waves $\mathbf{c}$ needs to be defined in terms of the transmit beamformer. This can be done by analysing the scattering parameters of the transceiver-antenna system. Since both the receive and transmit beamformer ports are defined to be ideally matched to the system (reflectionless), only the S_{21} parameter between each transmit and receive port is of interest.

A set of scattering parameters that characterize the frequency dependent behaviour of the antenna-transceiver system are extracted. These parameters are defined with the transmit ports as the input ports and receive ports as the output ports. Since the paramaters are frequency dependent, they are collected in a matrix with rows that correspond to different frequency points. Furthermore, to allow for correct matrix multiplication in the beamformer calculation, a receive beamformer matrix is created that contains the beamformer coefficients per frequency. Since the same beamformer is applied at each frequency point, this results in a matrix that contains copies of the receive beamformer corresponding to the number of frequency points. This process is described using pseudo-code in the following statements:

```
beamformerFreqExpanded(channels_x_frequencies, frequencies)
S_sys(channels, channels)
S_sysFreq(channels_x_frequencies, channels)

S_sysFreq = [S_sys(freq_1) ; S_sys(freq_2) ; ... ; S_sys(freq_n)]
for f = 1:nFreq
```

```
    beamformerFreqExpanded([(f-1) * nChannels + 1: f * nChannels], f)
        = beamformerRxRef;
end
```

With the correct manipulation applied to these matrices, a similar convex problem statement to the previous one can be formulated to solve for an increased isolation transmit beamformer. The statement is described using pseudo-code as follows:

```
abs((S_sysFreq * beamformerTxIso).' * beamformerFreqExp)).'
    <= targetIsolation
```

where the appended " .'" indicates a transpose of the matrix.

A further challenge to implementing the isolation beamformer algorithm is regulating the amplitude of both the receive and transmit beamformers. To understand this challenge, consider the fact that the isolation, as defined in equation 4.13, could be "improved" by simply reducing the amplitude of the transmit beamformer. Although this may seem like a logical solution to the convex solver, it does not improve the isolation for a given input power. To avoid this, the magnitudes of both the receive and transmit beamformers are kept at 1. This means that the convex problem must be placed in a loop and iterated, with the beamformer normalised at each iteration, until the requirements are met. To compensate for the inaccuracies in the isolation calculation due to this process, the target isolation is also adjusted accordingly, as illustrated by the following pseudo-code.

```
conditionsMet = false
while !conditionsMet
    perform convex problem

    beamformerIso = beamformerIso / l2-norm(beamformerIso)

    isolation = determine_isolation(beamformerIso)
    isolationIncrease = targetIsolation ./ isolation
    for point in isolationIncrease
        if point > target isolation
            point = 1

    if isolation == targetIsolation
        conditionsMet = true
    else
        targetIsolation = targetIsolation .* isolationIncrease
```

Since increasing the isolation requires an adjustment of the reference beamformer (i.e., moving away from the optimal gain point), it can only be achieved through a gain cost to the system. In fact, if too high of an isolation is specified in the algorithm, the resulting radiation pattern will not resemble the reference pattern at all. Therefore care should be taken to specify a target isolation that is reasonable. In

this work, the target isolation is capped at 100 dB. This is done not only to minimise the gain cost of the solution, but also to prohibit numerical error that occurs when extremely large and small numbers are used in convex problem statements with a limited numerical resolution.

4.4 Gain Mask Beamformer Algorithm

Both the transmit and receive radiation patterns have to adhere to a regulatory radiation mask, referred to as a gain mask for the receive side and an EIRP mask for the transmit side. For both cases the maximum gain in a scan direction is desired while conforming to the given mask. Thus the aperture size is chosen such that the main beam fits within the given main beam width of the mask and sufficient sidelobe suppression can be achieved. If the aperture size is insufficiently large the receive gain will not the able to adhere to the mask, and excessive back-off will have to be applied to the transmit side in order to lower the total EIRP in order to satisfy the mask.

The initial step follows the maximum gain beamformer algorithm, as in Section 4.1, and produces a beamformer weighting vector ($\mathbf{w_{gain}}$) with the desired vector that satisfies the mask as $\mathbf{w_{mask}}$. The mask must be presented in relative terms to the E-field and thus must be scaled according to the gain given by $\mathbf{w_{gain}}$. If a mask is given in terms of gain ($\mathbf{G_{mask}}$) the relative and scaled $\mathbf{E_{mask}}$ must be determined and used in the convex problem algorithm. The problem can thus be stated as a convex problem minimising the ℓ_2-norm of $\mathbf{w_{gain}} - \mathbf{w_{mask}}$ as shown below.

$$\begin{aligned} &\text{minimise}(\ell_2\text{-norm}(\mathbf{w_{gain}} - \mathbf{w_{mask}})) \\ &\text{subject to} \\ &\qquad \mathbf{w_{mask}} \cdot \mathbf{E} \leq \mathbf{E_{mask}} \end{aligned} \tag{4.19}$$

One catch with this method is that $\mathbf{w} \cdot \mathbf{E}$ scales with $|\mathbf{w}|$, whereas the gain is independent of $|\mathbf{w}|$. Thus by reducing $|\mathbf{w}|$ the E mask constraints can be met without meeting the gain mask constraints. Thus the convex problem statement 4.19 must be placed in a loop which normalises $\mathbf{w_{mask}}$ and checks the E mask constraint again.

$$\begin{aligned} &\mathbf{w_{comp}} = \mathbf{w_{gain}} \\ &\text{while } \mathbf{w_{comp}} \cdot \mathbf{E} \geq \mathbf{E_{mask}} \\ &\qquad \text{minimise loop} \\ &\qquad \mathbf{w_{comp}} = \mathbf{w_{mask}}/\ell_2\text{-norm}(\mathbf{w_{mask}}) \end{aligned} \tag{4.20}$$

4.4.1 Detailed Method and Description

The mask application algorithm only applies the mask correctly when providing a normalised beamformed E-field and a mask relative to this normalised field. The mask should also be satisfied in the specified bandwidth which requires manipulation of the matrices containing the EEPs. The dimensions of these matrices contain

the radiation pattern in the flattened theta and phi directions, per element, per frequency. Thus to have the correct dimensions and flattening, the radiation directions and frequency dimensions have to be flattened.

```
EEP(phi_theta, elements, frequencies)

for f = 1:nFrequencies
    EEP_beamformed = beamformerMaxGain * EEP(:, :, f)
    EEP_maxGain = max(abs(EEP_beamformed))
    EEP(:, :, f) = EEP(:, :, f) / EEP_maxGain

EEP = permute(EEP, [1, 3, 2])
EEP = reshape(EEP, [], size(EEP, 3))

EEP(phi_theta_frequency, elements)
```

A similar manipulation has to take place with the mask which can be accomplished by flattening the mask and appending itself the correct amount of times to correspond with the amount of frequency points. The gain mask must then be transformed to the relative E-field mask which is done by subtracting the peak gain from the gain mask per frequency.

```
gainMask(phi, theta)

gainMask = reshape(gainMask, [], 1)
oldGainMask = gainMask
for f = 2:nFrequencies
    gainMask(:, f) = oldGainMask

gainMask = reshape(gainMask, [], 1)

gainMask(phi_theta_frequency)

E_mask = gainMask - peakGain
```

Now that the array manipulation is such that normal matrix multiplication can take place the convex problem can be stated. As described in the previous section, the beamformer has to be normalised after a solution is found and the gain checked after normalisation. To combat the discrepancy the gain overshoot over the mask is determined and subtracted from the mask. This forces a faster convergence to the mask by correcting for the errors that are present from normalisation of the beamformer.

```
conditionsMet = false
while !conditionsMet
    perform convex problem

    beamformerMask = beamformerMask / l2-norm(beamformerMask)
```

```
gain = determine_gain_tx(beamformerMask)
overshoot = gain - gainMask
for point in overshoot
    if point < 0
        point = 0

if overshoot == 0
    conditionsMet = true
else
    E_mask = E_mask - overshoot
```

Upon meeting the conditions the latest iteration of the beamformer will satisfy the mask requirements. If a too stringent mask is applied to the radiation pattern the conditions will never be met and the loop will never end. For these conditions a set amount of iterations or other convergence criteria are specified. The resulting beamformer in that case will not satisfy the mask or provide an ideal radiation pattern. In these cases the mask must be adjusted and the beamformer disregarded.

4.5 Joint Isolation and Mask beamformers

The mask and isolation beamformers can be combined into a solution that satisfies both. The process of combining both is fairly trivial, but reduces the solution space for the beamformer significantly. This requires lowering of the maximum constraints of both the isolation and mask application to allow for an achievable solution. The methodologies of the mask and isolation beamformers are combined and the same optimisation goal provided to the convex problem but subject to both conditions of the mask and isolation.

5

Results

In Chapter 5 the most relevant results and outcomes are shown and discussed. In accordance with the previous chapter, the optimum beamformer results for the transmit and receive cases are provided first, followed by the isolation results, the mask adherence results, and lastly the combined beamformer results. After each results section, a brief summary is provided in which the unusual or unexpected results are highlighted. The chapter is concluded with a discussion that highlights the most important deductions that can be made from these results.

5.1 Transmit Subsystem Optimum Beamformers

For a given antenna-transceiver system, it is useful to analyse the system performance in terms of the maximum achievable gain. In this section, the transmit optimum gain beamformer results are shown for the transmit antenna as well as for the antenna-transceiver system. These results will be used as a reference to analyse the gain cost when other pattern restrictions (e.g., mask adherence) are applied.

5.1.1 Antenna Gain

The gain for the transmit antenna for seven scan angles (0°-60°) is shown in Figures 5.2 and 5.3. Note that the antenna gain is defined as the ratio of the output power to the accepted power of the antenna (see Figures 5.1 and 2.3 for reference).

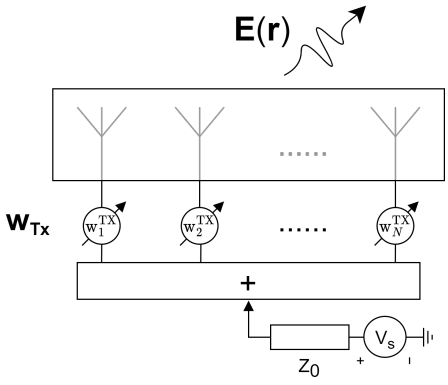


Figure 5.1: Transmit antenna gain

The graphs compare the gain performance of three different beamformers, namely the MGB, CFM beamformer, and the standard Uniform Excitation beamformer. Figure 5.2 shows the gain pattern for different scan angles, while Figure 5.3 compares the maximum achievable gain for the respective beamformer implementations over scan angle.

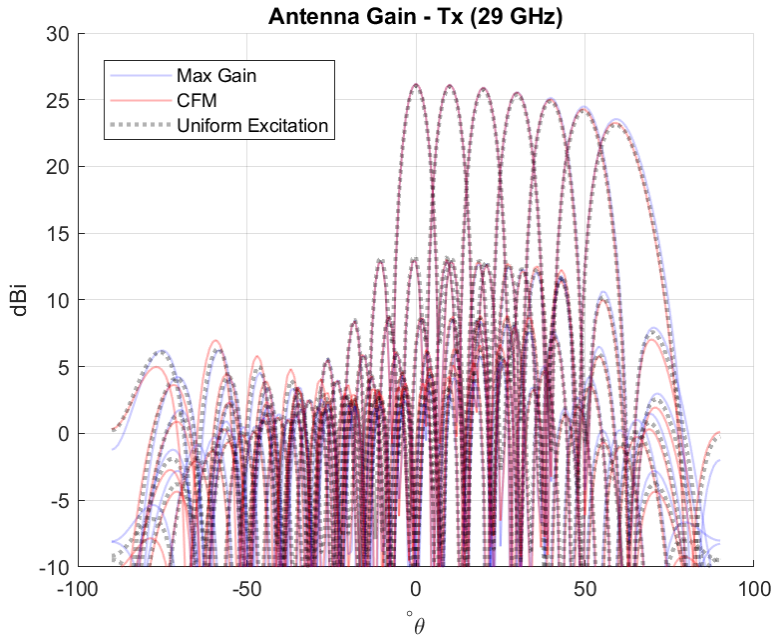


Figure 5.2: Transmit antenna gain over scan angle

From Figure 5.2, it is clear that the transmit gain pattern is degraded over scan angle. However, grating lobes are absent even at a scan angle of 60°.

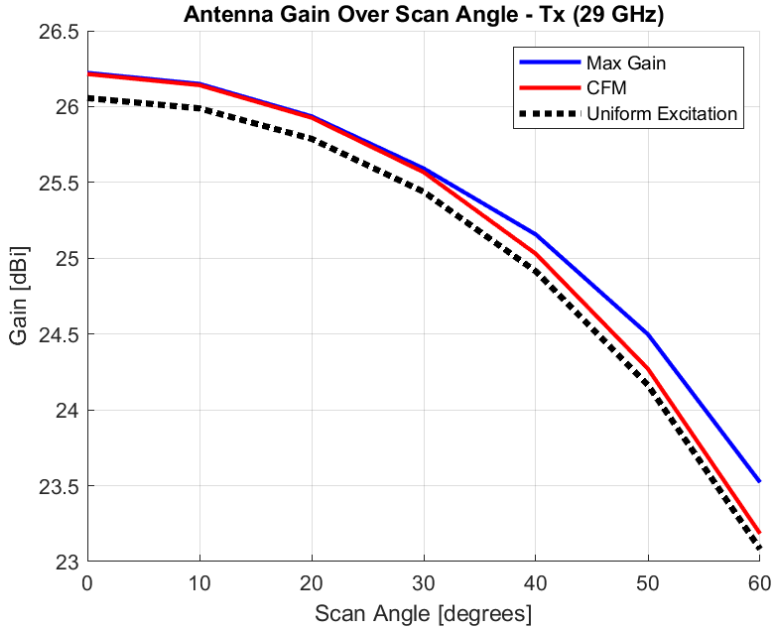


Figure 5.3: Transmit antenna gain over scan angle

Figure 5.3 shows that the MGB provides the best antenna gain performance over scan angle. However, the CFM beamformer also offers a gain improvement over the standard Uniform Excitation beamformer.

5.1.2 Realised System Gain

The realised system gain for the transmit subsystem for seven scan angles (0°-60°) is shown in Figures 5.5 and 5.6. Note that the realised system gain is defined as the ratio of the output power to the incident power of the system (see Figures 5.4 and 2.3 for reference). As the incident power is normalized to 1W, the realised system gain can also be described as the EIRP.

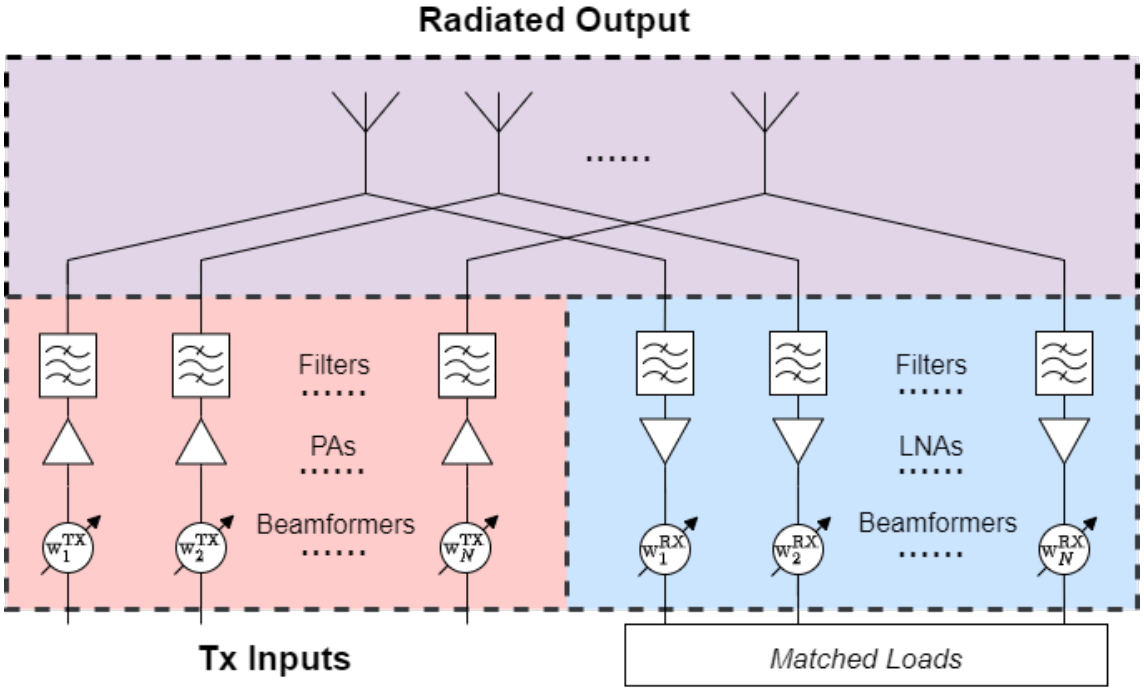


Figure 5.4: Input and output references for the realised transmit gain

As in the previous section, the graphs compare the gain performance of three different beamformers, namely the MRGB, CFM beamformer, and the standard Uniform Excitation beamformer.

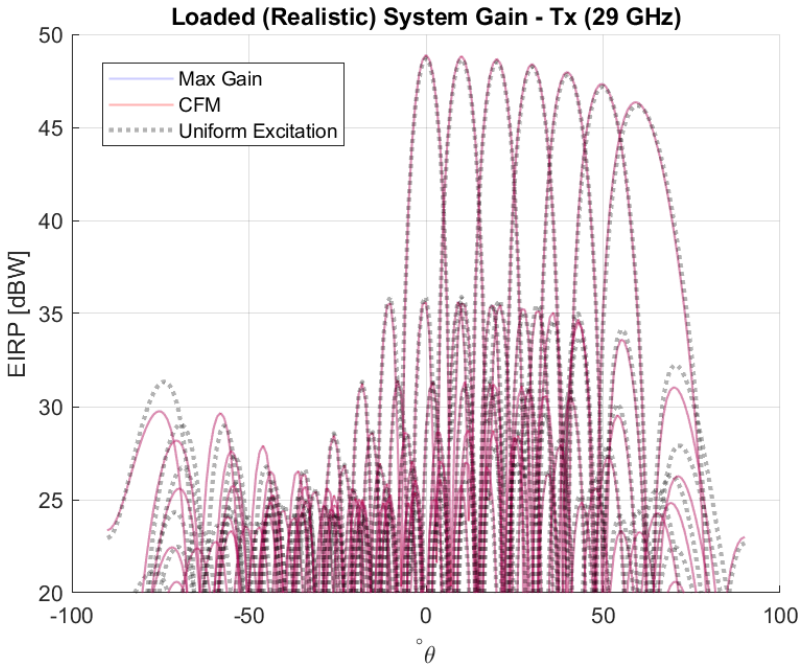


Figure 5.5: Realised transmit gain over scan angle

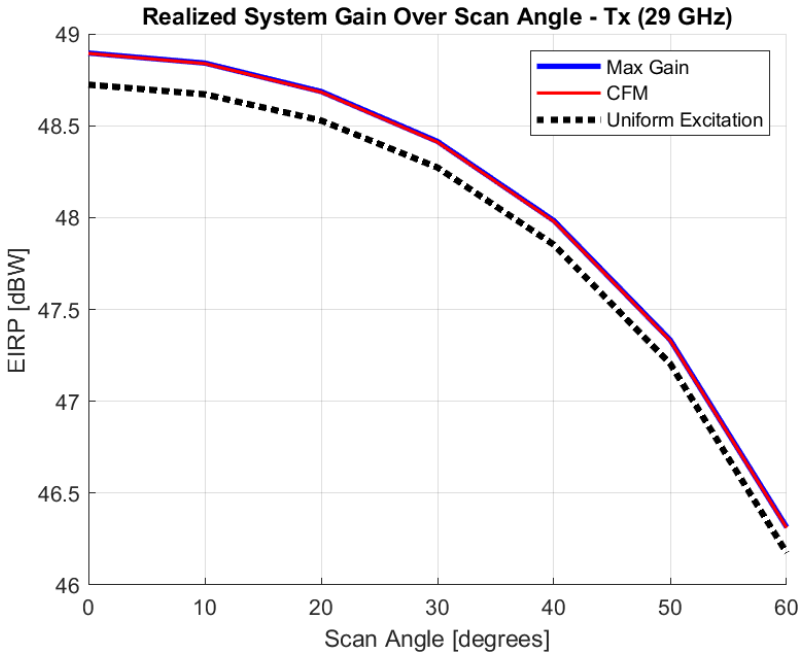


Figure 5.6: Realised transmit gain over scan angle

As seen in Figure 5.6, the realised system gain performance of the MRGB and CFM is identical. This confirms the analysis given in Section 4.1.

It is clear that the realised system gain is much higher than the antenna gain, due to the power amplifiers in the transmitter path. However, the full 28 dB gain of

from the PAs is not translated directly to the realised system gain. This is due to the loading of the antenna by the transceiver subsystem, which means that some power is dissipated in the electronic components and the receiver subsystem.

A further performance reduction is due to impedance mismatches between the transceiver components and the antenna. This effect is intensified over scan angle as the active impedance of the antenna changes.

5.1.3 Summary

It is clear that both the CFM and the MGB/MRGB beamformers offer a performance impovement over the Uniform Excitation beamformer. Furthermore, the CFM and MRGB beamformers give identical realised gain performance. This makes these superdirective beamformers a good starting point from which further pattern constraints can be applied. In this way, the maximum gain solution for a given pattern constraint can be found.

5.2 Receive Subsystem Optimum Beamformers

The same gain analysis is carried out for the receive subsystem. In this section, the receive optimum gain beamformer results are shown for the receive antenna and for the antenna-transceiver system.

5.2.1 Antenna Gain

The gain for the receive antenna for seven scan angles (0°-60°) is shown in Figures 5.8 and 5.9. The beamformer reference plane for the receive antenna gain is shown in Figure 5.7.

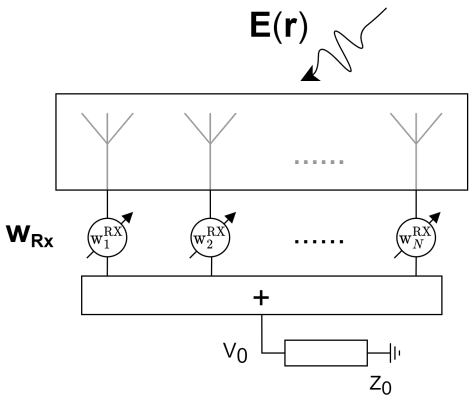


Figure 5.7: Receive antenna gain

The graphs compare the gain performance of three different beamformers, namely the MGB, CFM beamformer, and the standard Uniform Excitation beamformer.

Figure 5.8 shows the gain pattern for different scan angles, while Figure 5.9 compares the maximum achievable gain for the respective beamformer implementations over scan angle.

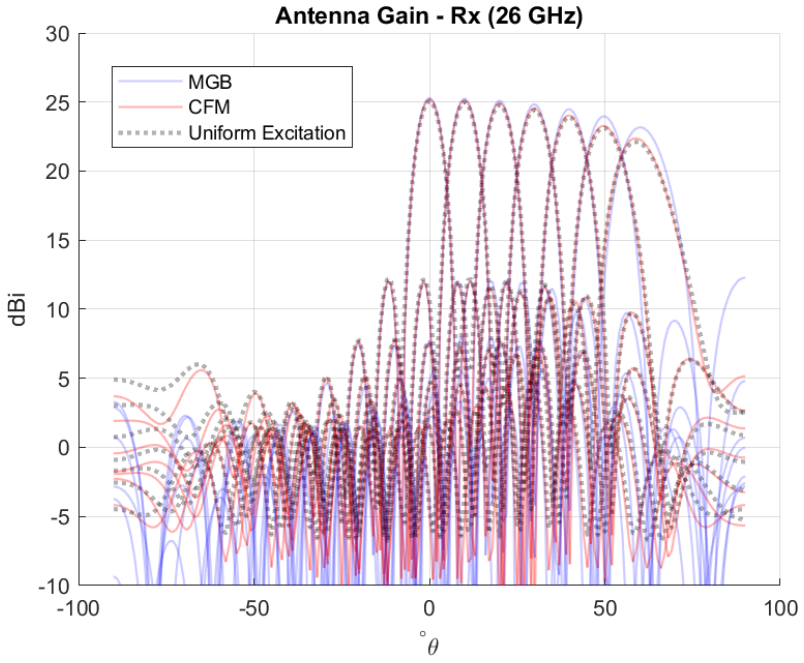


Figure 5.8: Receive antenna gain over scan angle

As with the transmit case, Figure 5.8 shows that the receive gain is degraded over scan angle. However, no grating lobes are present.

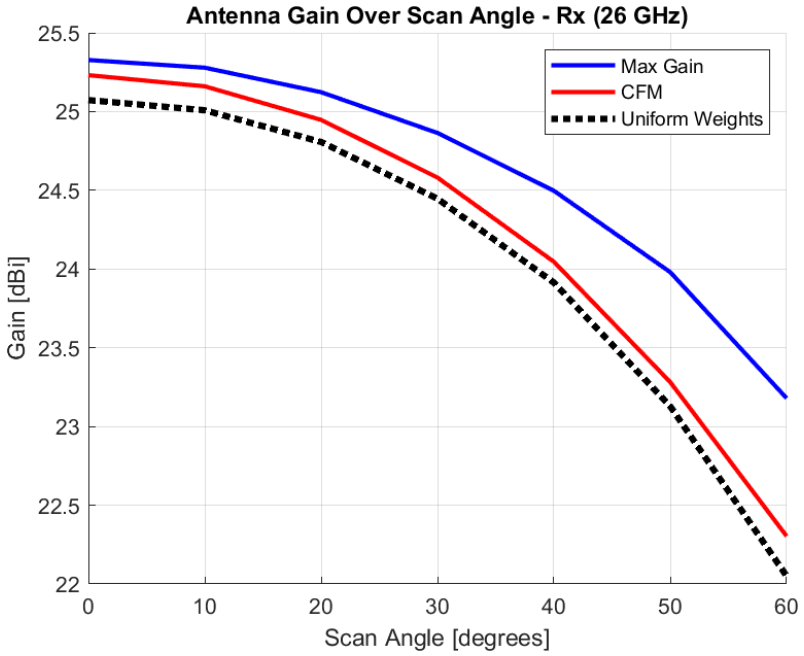


Figure 5.9: Receive antenna gain over scan angle

Figure 5.9 shows that both the CFM beamformer and MGB offer a performance increase over the Uniform Excitation beamformer, with the MGB offering the best performance.

It is interesting to note that the antenna gain for the receive case is less than that of the transmit case. This is expected, as the effective aperture for a given antenna (to which the gain is proportional to) increases with frequency.

5.2.2 Realised System Gain

The realised system gain for the receive subsystem for seven scan angles (0°-60°) is shown in Figures 5.11 and 5.12. The input power and output power references for the realised receive system gain is shown in Figure 5.10.

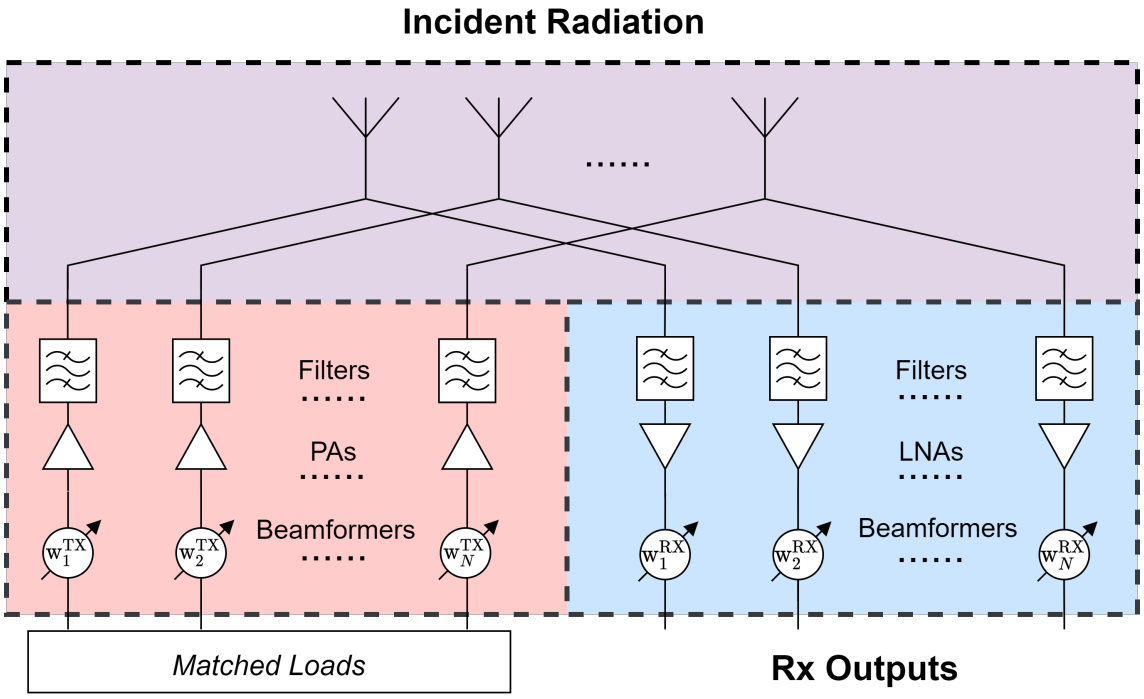


Figure 5.10: Input and output references for the realised receive gain

These results differ from the previous results in that the MRGB is omitted. However, as was clear from the analysis in Section 4.1 as well as the results in Section 5.1.2, the realised gain performance of the MRGB and that of the CFM beamformer is identical.

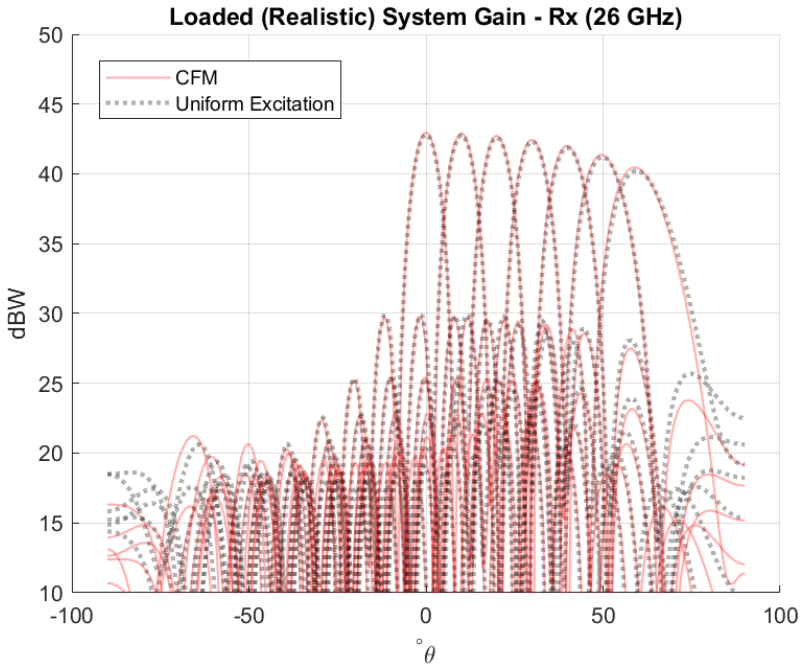


Figure 5.11: Realised receive gain over scan angle

Figure 5.12 shows the performance increase offered by the CFM beamformer over the Uniform Excitation beamformer.

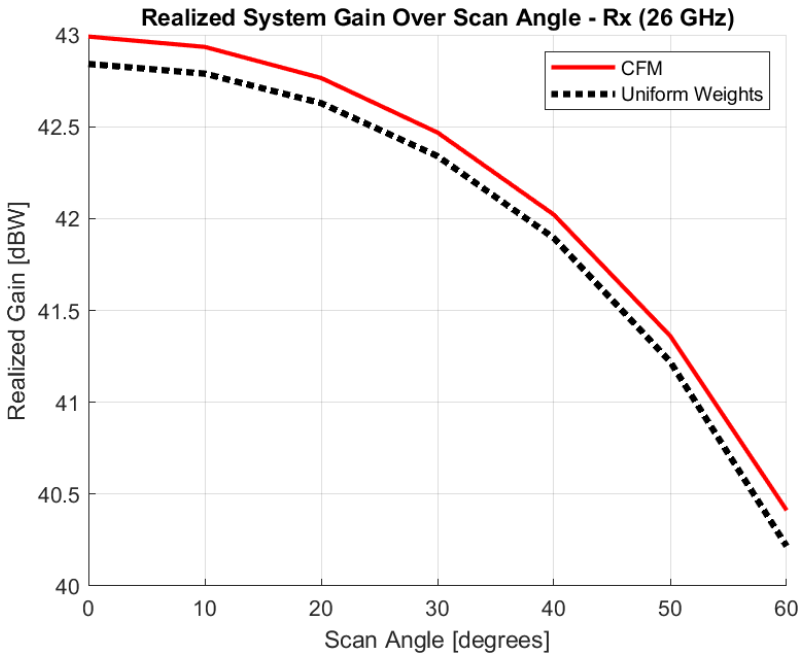


Figure 5.12: Realised receive gain over scan angle

As with the transmit case, it is clear that the full 20 dB of gain offered by the low-noise amplifiers is not transferred to the realised system gain. This is due to

mismatches in the antenna-transceiver system, and could be improved by designing dual band antenna elements that are well matched to the transceiver circuitry.

5.2.3 Summary

The above results show that the CFM beamformer offers better realised gain performance than the Uniform Excitation beamformer. This makes it a good reference from which further pattern constraints can be applied.

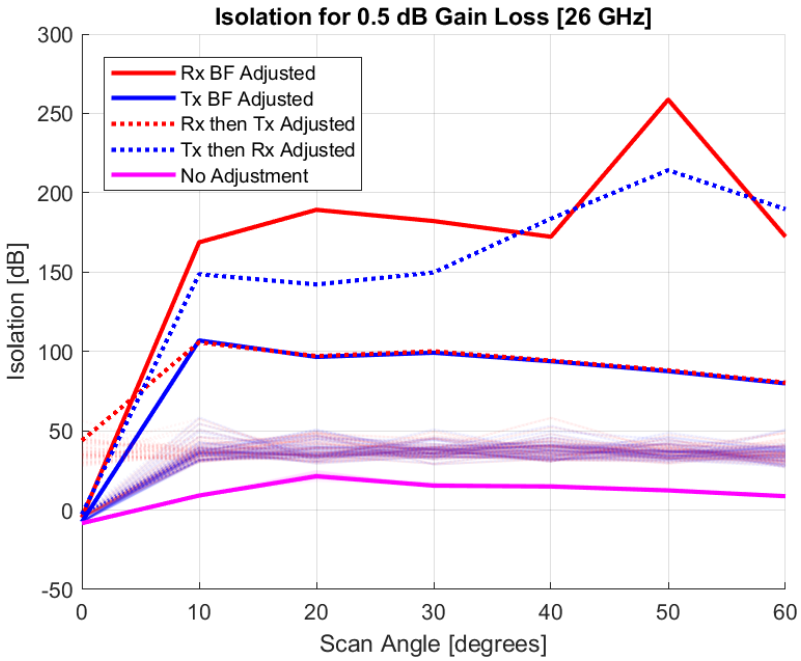
5.3 Isolation Beamformer Results

The following section shows the results of adjusting the Tx and Rx CFM beamformers in such a way as to provide higher isolation between the Tx and Rx ports. The isolation beamformers are calculated according to the algorithm described in Section 4.3.3. The isolation definition takes into account the entire transceiver-antenna system (see Figure 3.2 for reference). Four different beamformers are calculated to improve the isolation: first the Rx and Tx beamformers are adjusted independently in such a way as to increase the isolation when their CFM counterparts are present respectively. Thereafter the algorithm is incremented, and new Rx and Tx beamformers are calculated based on the increased isolation beamformers in the previous step.

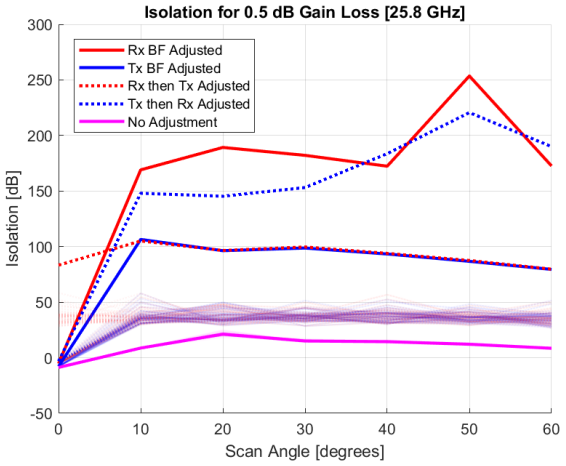
The beamformers are calculated per scan angle. However, the beamformer algorithms are set up in such a way that a single beamformer is calculated to satisfy the gain and isolation requirements over a given bandwidth (400 MHz, in this case). This is done by specifying specific frequency points within the bandwidth at which the algorithm checks that the isolation requirement is met.

5.3.1 Achievable Isolation for a Set Gain Cost

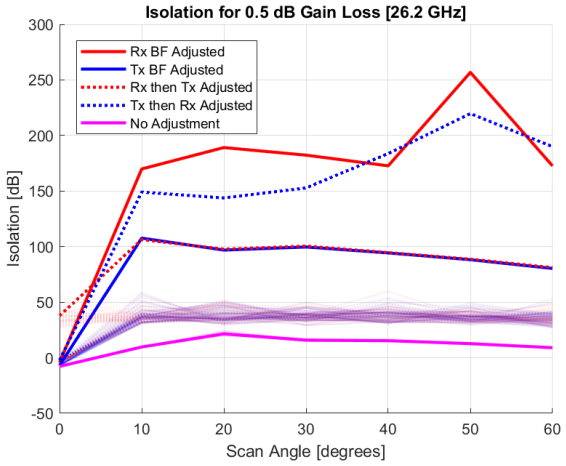
Figure 5.13 shows the maximum achievable isolation between the Rx and Tx ports if up to a 0.5 dB drop in gain is permitted (with reference to the optimal gain achieved with the CFM beamformers). Three frequency points are shown over the bandwidth, although the isolation algorithm can be used to calculate a beamformer that takes into account an arbitrary number of frequency points (for an increase in computational power/time).



(a) Centre frequency



(b) Lower frequency bound



(c) Upper frequency bound

Figure 5.13: Achievable isolation for a 0.5 dB drop in optimal gain over scan angle and frequency

The dark lines show the isolation result provided by the optimal solution as calculated by the beamformer algorithm. The opaque lines show the results of a Monte Carlo simulation run on the beamformer with an amplitude perturbation of 0.5 dB and a phase perturbation of 6°.

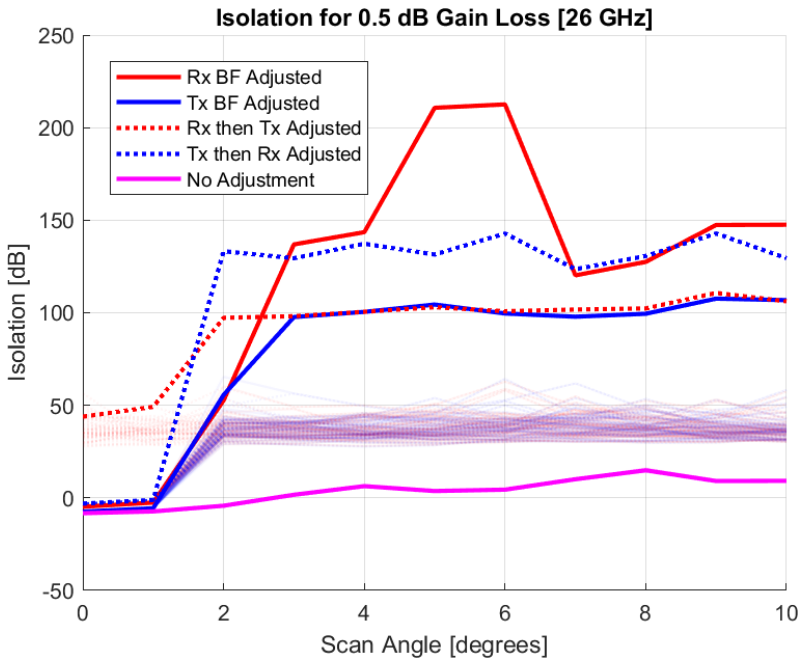
One peculiarity to note present in all figures in this section is that iterated beamformers (e.g., Rx beamformer adjusted after the Tx beamformer has been adjusted) are expected to provide improved isolation over the initial beamformers. In other

words, it is expected that the dotted red lines in Figure 5.13 should always be above the solid lines. However, as is clear from the figure, this is not always the case. The reason for this is that even small adjustments in the beamformers from the optimum isolation point can lead to a large drop in gain. This is evident in the Monte Carlo simulations, where a small perturbation to the beamformer weights will reduce the isolation from the optimum point to around 35 dB. Furthermore, the isolation algorithm resolution is limited by the specified stop criteria. Therefore the decrease in isolation in the iterated beamformers is due a very narrow solution space and limited numerical resolution.

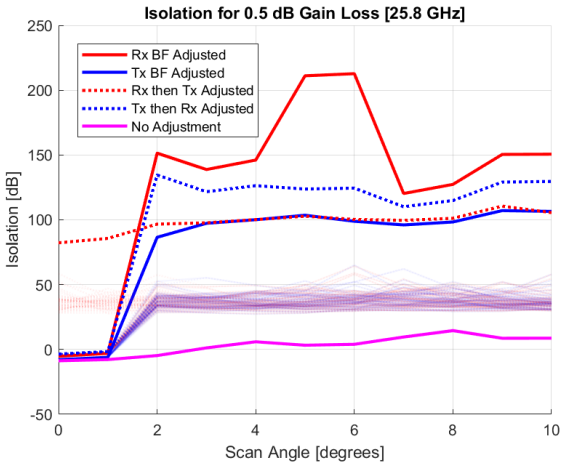
Several stop criteria are set for the isolation algorithm. A convergence criteria is set stating that, during iterations of determining the beamformer, if the weights individually differ less than 0.1% from each other in terms of real and imaginary components the solver should halt and choose the latest results. We reason that the resulting beamformers will not be practically implementable with a precision higher than this.

Furthermore, a maximum target isolation of 200 dB was specified in order to aid with the simulation. While the achieved isolation can be higher than the limit of 200 dB, the resulting numerical computation reduces greatly in accuracy due to the size of the numbers, which causes the convex toolbox used as part of the solver to break.

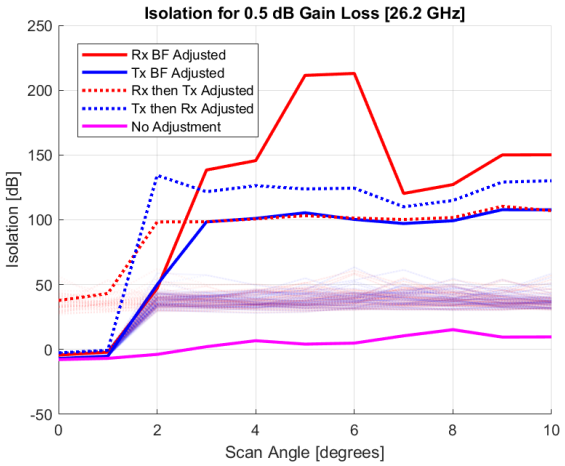
From Figure 5.13, it is interesting to note that the achievable isolation is very low at broadside - significantly lower than at higher scan angles. To investigate this further, an isolation simulation is set up with a finer scan angle resolution between 1°-10°. The results are shown in Figure 5.14.



(a) Centre frequency



(b) Lower frequency bound



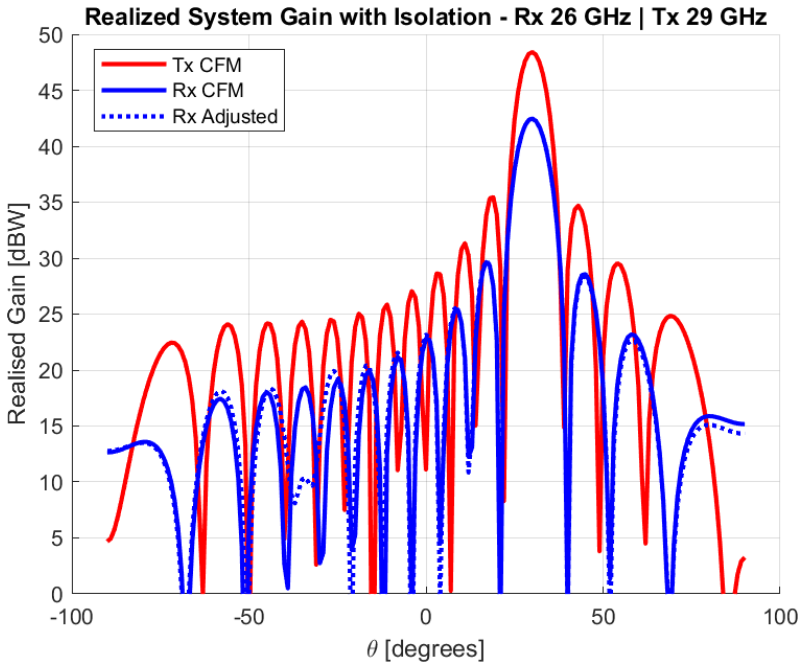
(c) Upper frequency bound

Figure 5.14: Achievable isolation for a 0.5 dB drop in optimal gain over scan angle and frequency

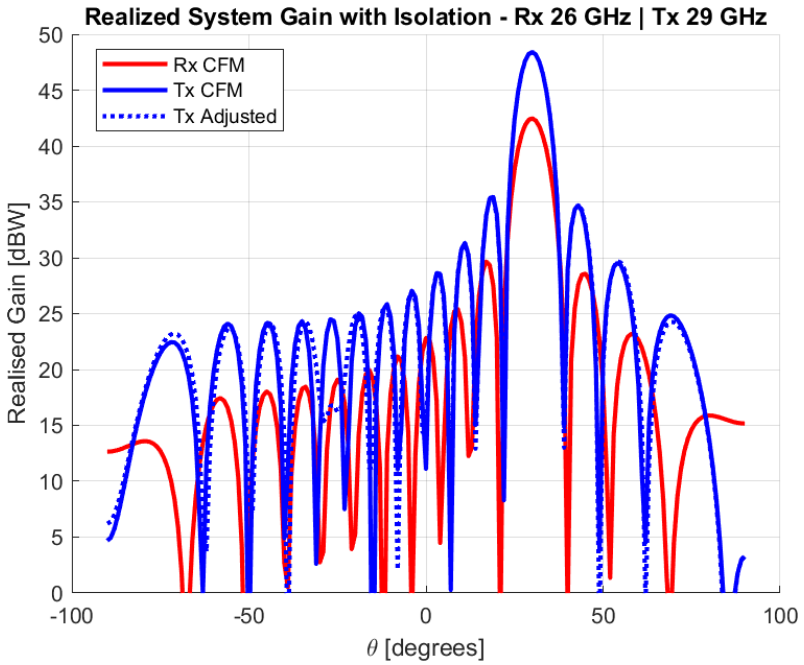
From Figure 5.14, it is clear that the achievable isolation drops at a scan angle of approximately 2°. This corresponds roughly with the half-power beamwidth of the broadside beam. The achievable isolation for scan angles 2°-10° corresponds well with that seen in higher scan angles in Figure 5.13.

5.3.2 Gain Cut

The following figures show the resulting realised gain patterns in the principle plane with the isolation beamformers applied. The receive and transmit patterns from the CFM beamformers are shown as reference, together with the adjusted isolation beamformers. For brevity's sake only the scan to 30° is included, although this is representative of the patterns at other scan angles as well.



(a) Rx beamformer adjusted



(b) Tx beamformer adjusted

Figure 5.15: Realised gain resulting from the CFM beamformers compared to the isolation improved beamformers

From these figures, it can be seen that the isolation beamformers rely on sidelobe redistribution to improve the isolation. It is useful to think of this in terms of the

pattern overlap integral - if the receive and transmit patterns can be adjusted to overlap less, the isolation between them can be increased.

5.3.3 Summary

The above results clearly indicate that an improvement in isolation can be achieved for a small gain cost by adjusting the transmitter and receiver beamformers. The resulting beamfomer is very sensitive to small perturbations in phase and amplitude. However, even the perturbed beamformer offers a significant improvement in isolation, as shown by the Monte Carlo analyses in Figure 5.13 and 5.14.

The results also indicate that there exists solutions that improve isolation at broadside, especially if the algorithm is incremented. Further incrementation should be investigated for increased broadside isolation and improved robustness against perturbations.

5.4 Mask Application Beamformer Results

The following section shows the results of constraining the gain pattern to adhere to a given mask by adjusting the Tx and Rx CFM beamformers. First the transmit mask adherence results are shown, followed by the receive mask adherence results, and lastly an analysis of the gain reduction due to the mask adherence. Note that these results consider the realised system gain in both the receive and transmit cases (see Figures 5.4 and 5.10). As for the isolation algorithm, the beamformers are calculated per scan angle, but must work over the given bandwidth.

Although only the radiation patterns in the principle planes are shown here, it is important to note that the mask is applied in all phi planes, and not just in the shown principle plane.

5.4.1 Transmit Mask Adherence

Figure 5.16 shows the transmit mask adherence results for a scan angle of 60°. The black lines indicate the mask constraints, while the blue line shows the gain pattern resulting from the CFM beamformer excitations. It is clear that the CFM gain pattern violates the mask at multiple points. The red lines show the results of applying the mask adherence algorithm to the CFM beamformer. The multiple red line represent the results of a Monte Carlo simulation run on the Applied Mask beamformer, with a 0.5 dB perturbance in beamformer amplitude and a 6° perturbance in beamformer phase.

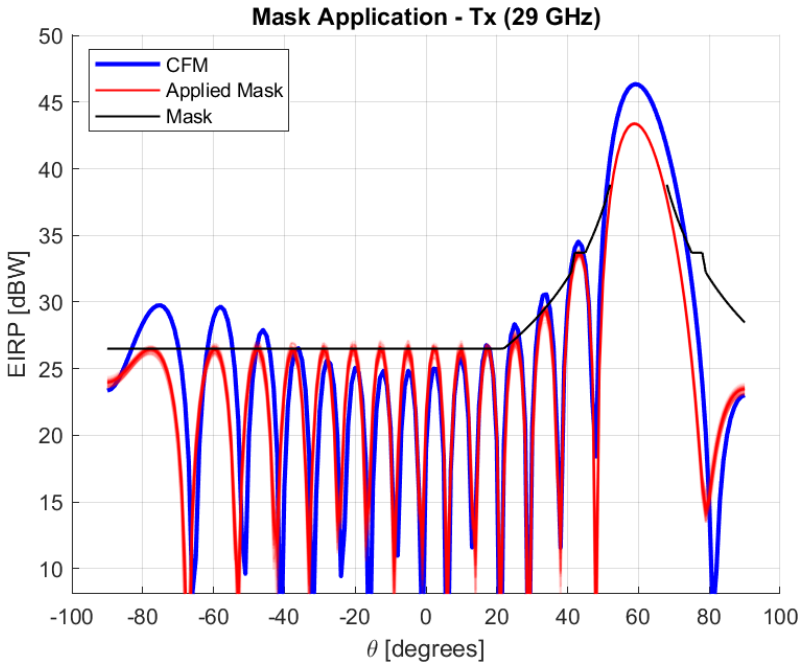
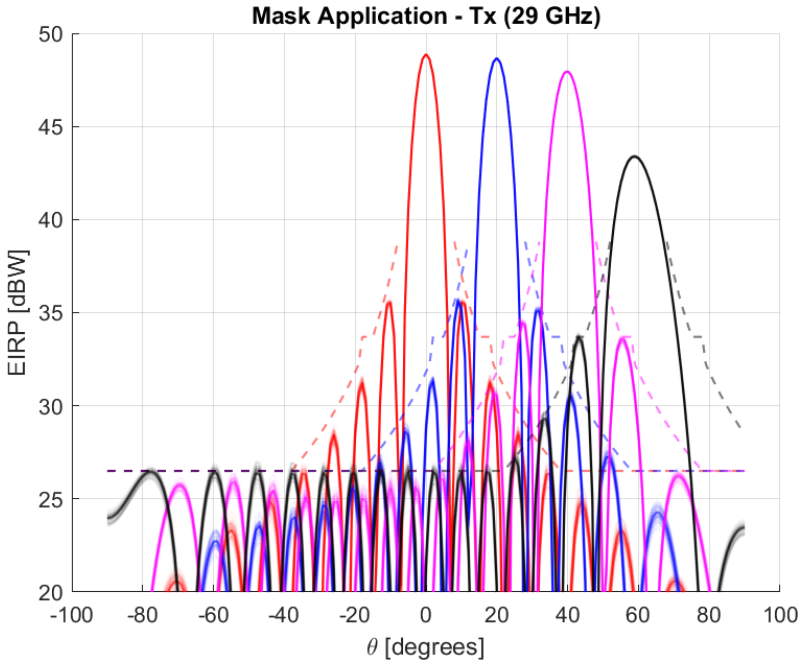
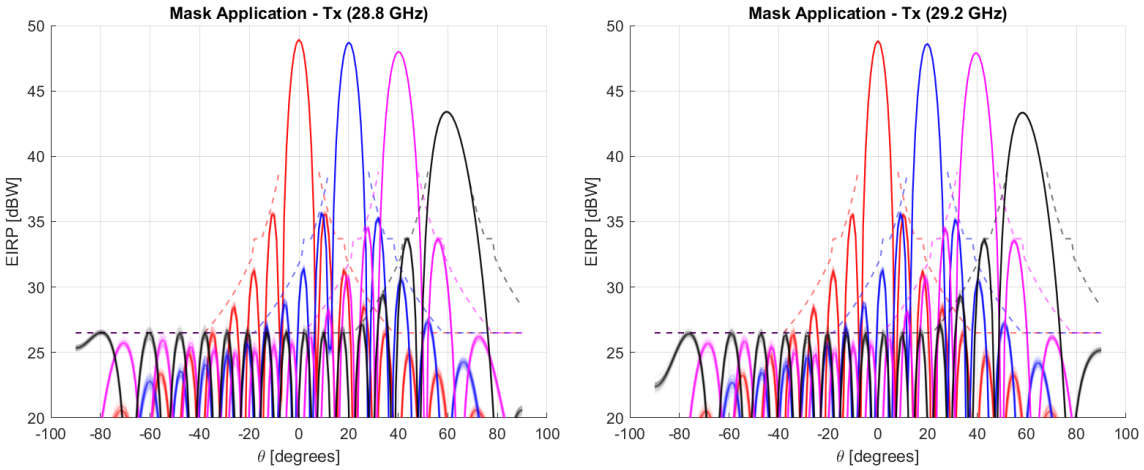


Figure 5.16: Transmit mask adherence at a 60° scan angle

Figure 5.17 shows the transmit mask adherence results for different scan angles at the centre frequency, lower frequency bound, and the upper frequency bound respectively. Each colour represents a different scan angle, while the multiple opaque lines per colour show the results of a Monte Carlo simulation per scan angle. The mask per scan angle is shown as dotted lines.



(a) Centre frequency



(b) Lower frequency bound **(c)** Upper frequency bound

Figure 5.17: Transmit mask adherence over scan angle

5.4.2 Receive Mask Adherence

Figure 5.18 shows the receive mask adherence results for a scan angle of 60°. The same Monte Carlo simulation was run on the receive mask adherence beamformer as on the transmit case, with a 0.5 dB perturbance in beamformer amplitude and a 6° perturbance in beamformer phase.

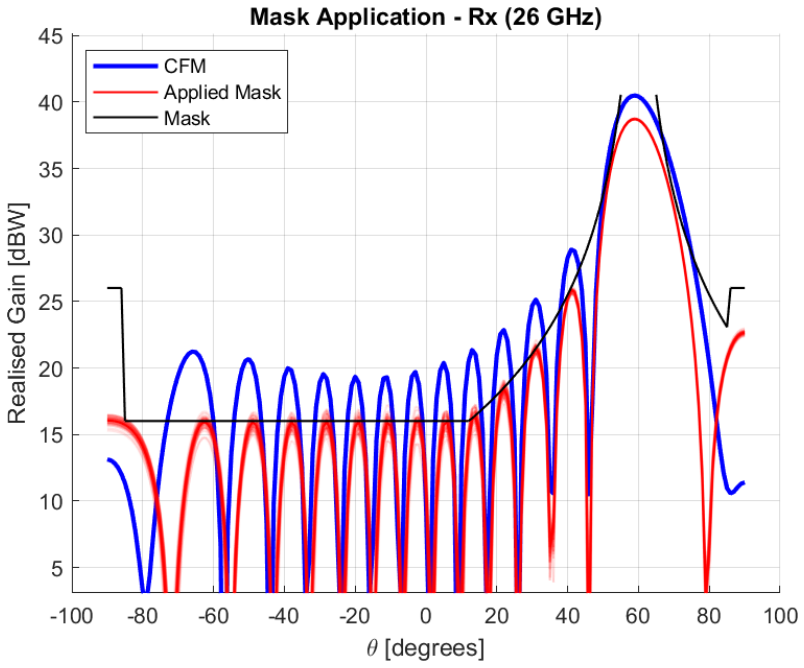
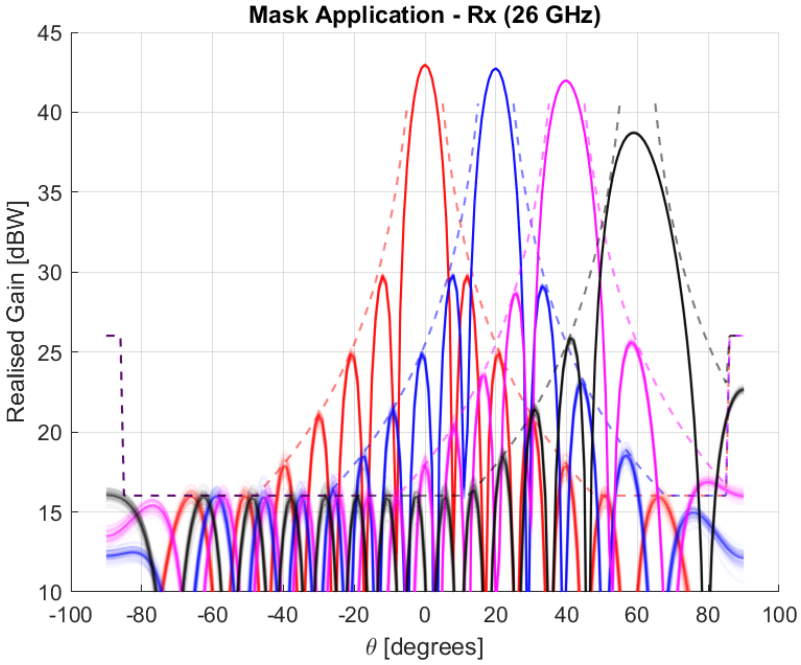


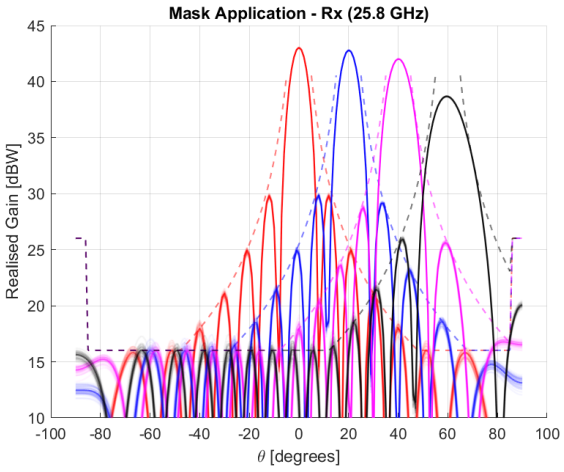
Figure 5.18: Receive mask adherence at a 60° scan angle

Figure 5.19 show the receive mask adherence results for different scan angles at the centre frequency, lower frequency bound, and the upper frequency bound respec-

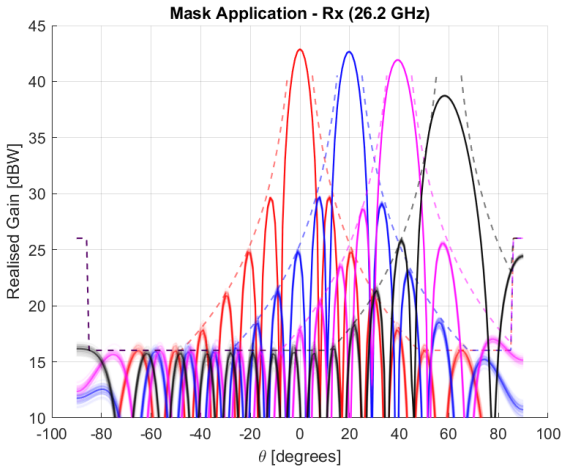
tively. As with the transmit case, each colour represents a different scan angle, with the dotted lines indicating the mask per scan angle. The results of a Monte Carlo simulation per scan angle are shown as opaque lines.



(a) Centre frequency



(b) Lower frequency bound



(c) Upper frequency bound

Figure 5.19: Receive mask adherence over scan angle

5.4.3 Gain Reduction over Scan Angle

Figure 5.20 summarised the gain cost of mask adherence for both the transmit and receive case. Note that only the original mask application beamformer solution is taken into account in this graph, and not the Monte Carlo results. The magenta

lines, with reference to the right y-axis, show the gain difference between the CFM and mask application beamformers for the receive and transmit case respectively.

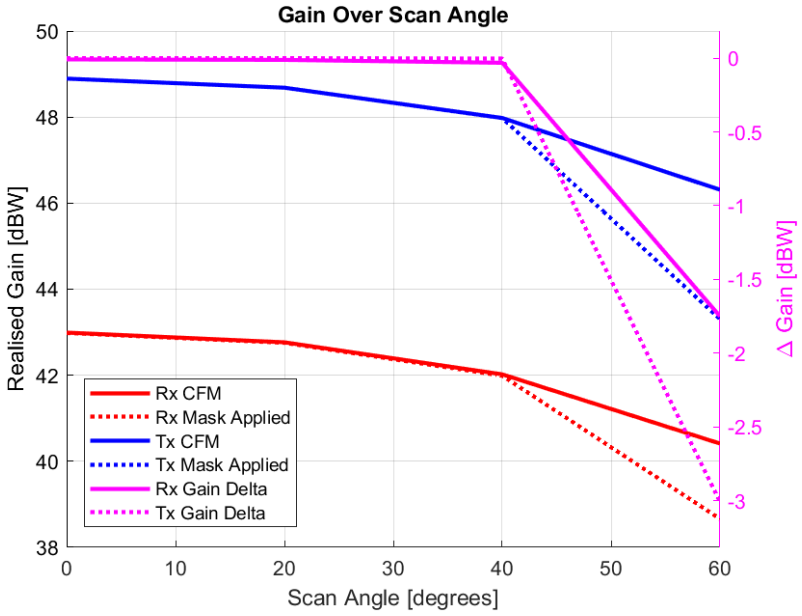


Figure 5.20: Gain cost for mask adherence over scan angle

Both the receive and transmit mask application cases show a very small gain cost for scan angles up to 40°. However, as discussed in the previous sections, a significant gain cost is present for large scan angles. Especially the transmit gain degrades at large scan angles due to beam-widening.

5.4.4 Summary

It is clear that the application of the mask adherence beamformer for both the receive and transmit case is successful. The Monte Carlo results show that some perturbations may result in a mask violation, but this violation is very slight and can be compensated for with a slight offset applied to the mask in the sidelobes. Overall, the beamformer is robust against amplitude and phase variation.

A large gain cost can be seen in both receive and transmit for mask adherence at a scan angle 60°. This is due to beam-widening at large scan angles - as can be clearly seen from Figures 5.17 and 5.19, the beam is much broader at 60° than at broadside. To force the broadened beam to adhere to the mask, the algorithm must reduce the gain across the entire gain pattern.

It is important to note in the transmit case that the mask adherence was achieved without applying back-off on the PAs. Throughout all simulations the PAs are assumed to be operating at the same gain with the only adjustment being available on the beamformers. Furthermore, the beamformers are constrained to have a magnitude of 1, which does not allow the overall lowering of EIRP.

5.5 Joint Mask and Isolation Beamformer Results

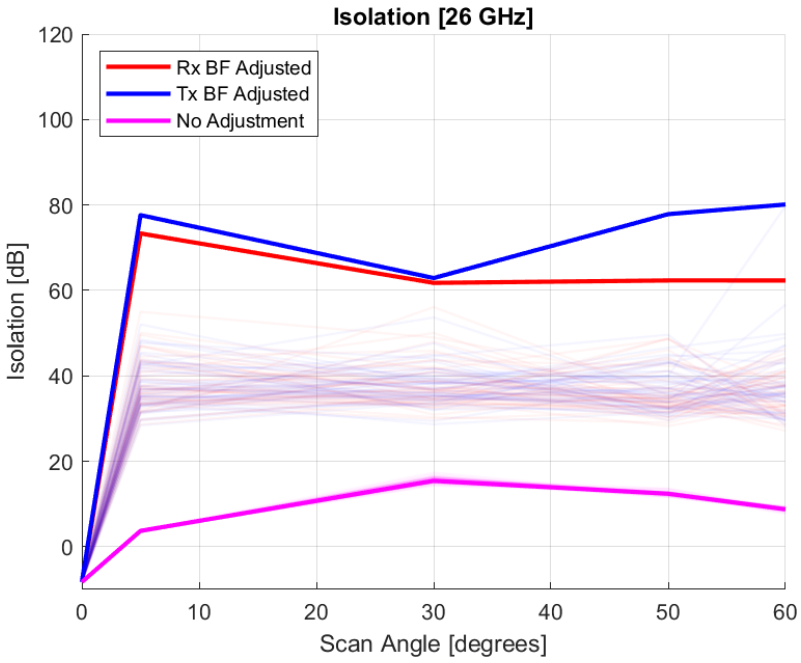
In this section, the results of applying both the mask adherence algorithm as well as the isolation algorithm are shown. The Rx and Tx CFM beamformers are used as a starting point to ensure that the final solution is as close to optimum gain as possible. Thereafter a target isolation and a radiation mask is specified. From these, receive and transmit beamformers are calculated that simultaneously meet the isolation and mask adherence requirements. First the achieved isolation results are shown, followed by the final gain patterns showing mask adherence. Lastly, the gain cost for the combination of mask adherence and isolation improvement is analysed. Note that these results consider the realised system gain in both the receive and transmit cases (see Figures 5.4 and 5.10).

The isolation and mask adherence specifications were considered very carefully during the simulation setup for the final system, and chosen to be achievable for the specified system. In Appendix C, some extra results are provided that illustrate the result when the performance specifications for the simulations are poorly chosen or not achievable.

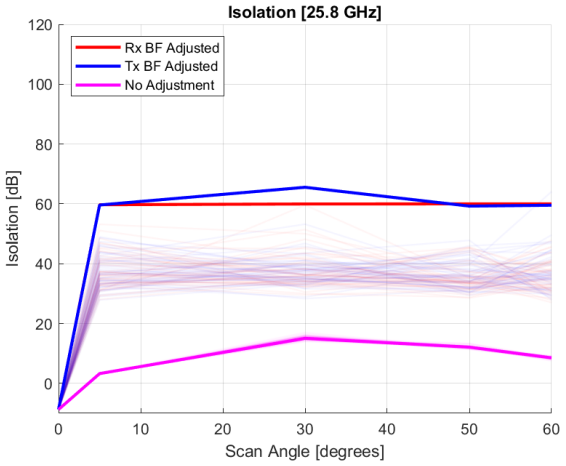
5.5.1 Specified Isolation

Figure 5.21 shows the achieved isolation of the final system (including mask adherence). Only the results of the independently adjusted Rx and Tx beamformers are presented. It is expected that the iterated beamformers will provide a better performance, but they are omitted here due to limitations in computational power and time. The dark lines show the isolation result of the calculated solution, while the faint lines show the results of a Monte Carlo simulation run on the beamformers with a 0.5 dB perturbance in beamformer amplitude and a 6° perturbance in beamformer phase.

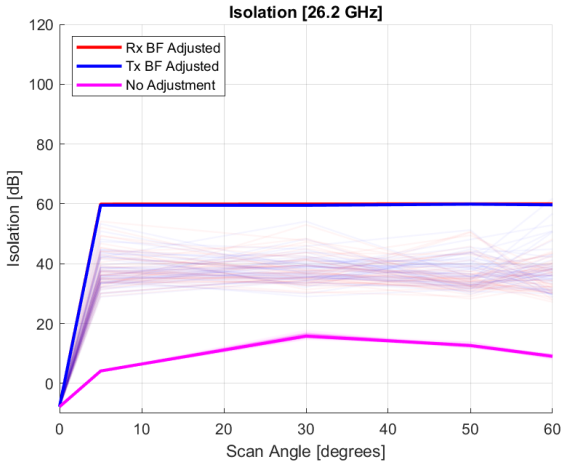
The target isolation for the system was set to 60 dB. Note that the target isolation at broadside relaxed to a 1 dB improvement. This was done because a combination of good mask adherence and high isolation is difficult to achieve at broadside.



(a) Centre frequency



(b) Lower frequency bound

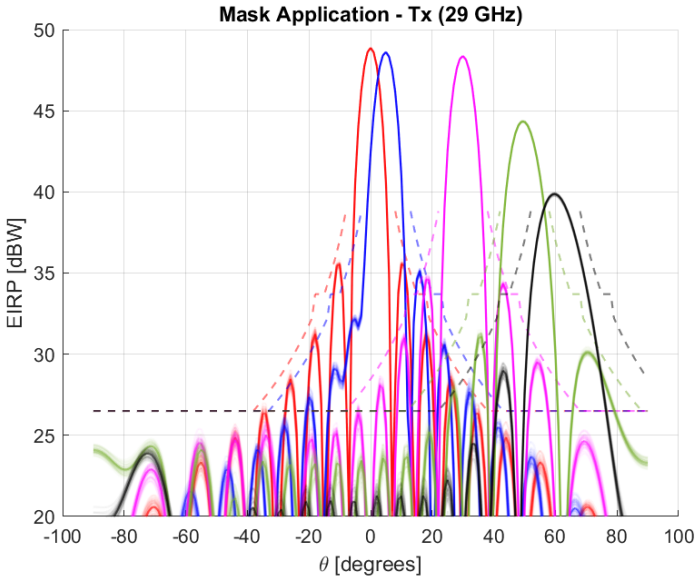


(c) Upper frequency bound

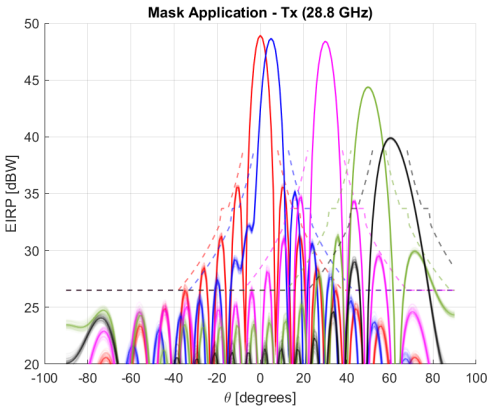
Figure 5.21: Achieved isolation when specifying isolation over bandwidth per scan angle

5.5.2 Mask Adherence Tx

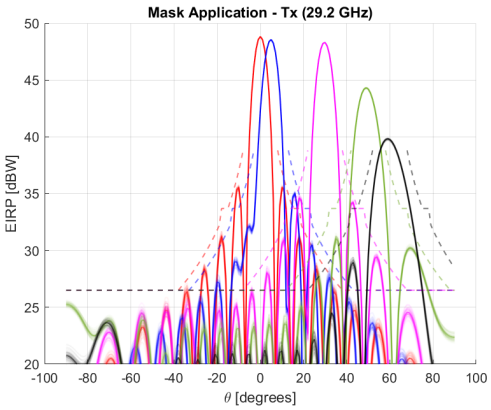
The radiation pattern and transmit mask adherence of the system are shown in Figure 5.22. The dark lines show the isolation result of the calculated solution, while the faint lines show the results of a Monte Carlo simulation run on the beamformer. The dotted lines show the radiation mask for each scan angle.



(a) Centre frequency



(b) Lower frequency bound

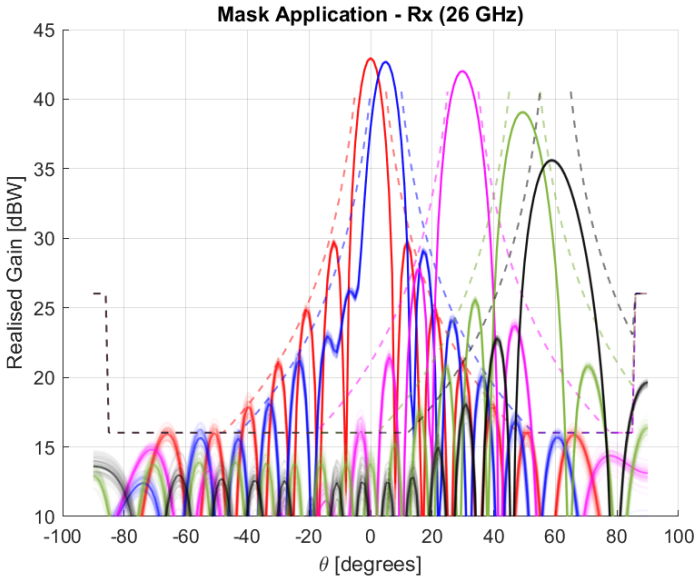


(c) Upper frequency bound

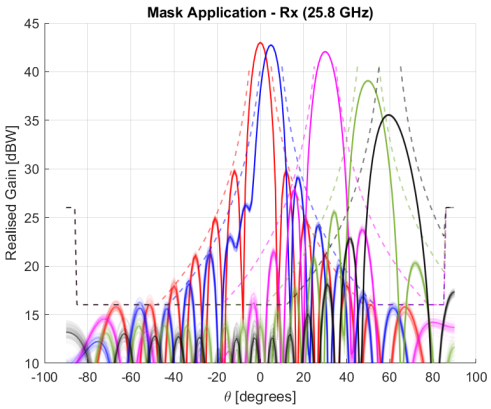
Figure 5.22: Transmit mask adherence with specified isolation applied

5.5.3 Mask Adherence Rx

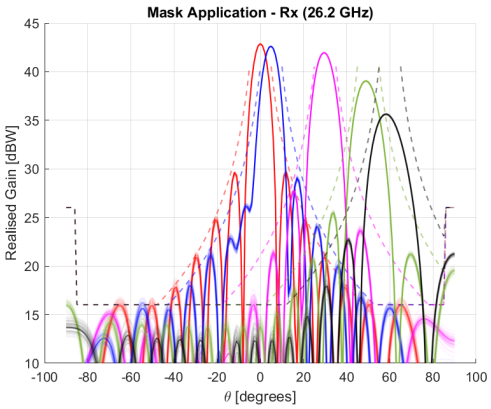
The received radiation pattern and receive mask adherence of the system are shown in Figure 5.23. The dark lines show the isolation result of the calculated solution, while the faint lines show the results of a Monte Carlo simulation run on the beamformer. The dotted lines show the radiation mask for each scan angle.



(a) Centre frequency



(b) Lower frequency bound



(c) Upper frequency bound

Figure 5.23: Receive mask adherence with specified isolation applied

5.5.4 Gain reduction

Figure 5.24 summarises the gain cost of the combination of mask adherence and improved isolation for both the transmit and receive case. Note that only the calculated solution is taken into account in this graph, not the Monte Carlo results. The magenta lines, with reference to the right y-axis, show the gain difference between the CFM beamformer and the joint isolation and mask application beamformer for the receive and transmit case respectively.

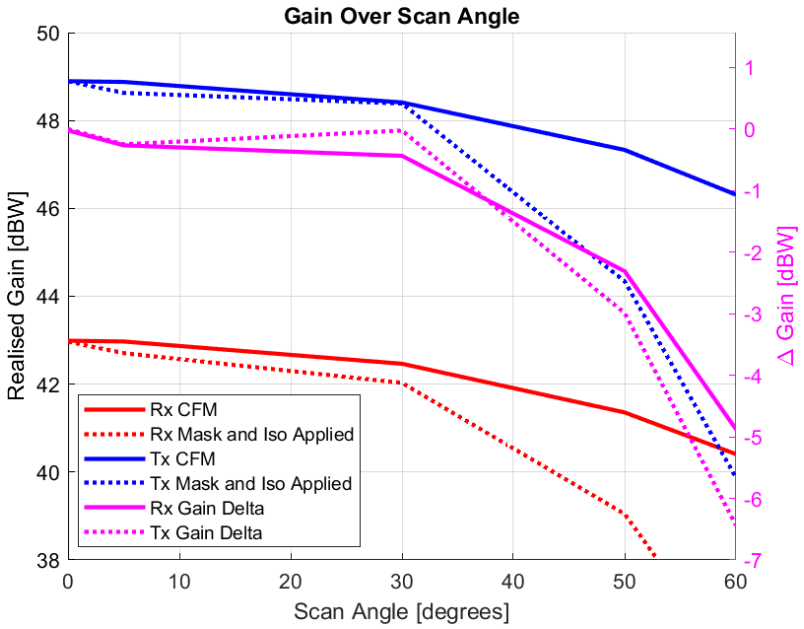


Figure 5.24: Gain reduction over scan angle

Both the receive and transmit results show a small gain cost for scan angles up to 40°. However, a significant gain cost is present for large scan angles.

5.5.5 Summary

These results illustrate that a specified mask and isolation can be met simultaneously with a slight cost to gain. However, care must be taken in specifying the mask and isolation requirements of the system. For example, achieving good isolation at broadside requires a large gain cost. This is elaborated on in Appendix C.

In some instances the specified target isolation is either overshot or not met. This is either due to an excessive isolation constraint, or a numerically narrow solution space which requires a numerical resolution that is finer than the convergence criteria of the solver.

When scanning to 60° a marked gain decrease is observed in both the transmit and the receive case, which is due to the excessive constraints of the mask and isolation simultaneously.

5.6 Discussion of Results

This section elaborates on the results given in the previous section. Specifically, the most important deductions that can be made from these results are highlighted and fundamental performance trade-offs are identified.

5.6.1 Optimum Beamformers

From the results in Section 5.1, some interesting observations on superdirective beamformers are worth pointing out. First, it is useful to consider the array antenna gain over scan angle, as shown in Figure 5.3 for the transmit case and Figure 5.9 for the receive case. For the uniformly weighted beamformer, a 3 dB loss in gain is observed at a scan angle of 60°. This is expected, as antenna gain is proportional to the effective area of the antenna, which is reduced by a factor of $1/\cos\theta = 2$ at $\theta = 60°$. However, due to the nature of superdirective beamformers and the fact that the gain of an array antenna is not only proportional to its effective area but also to the chosen beamformer excitation, this limit is overcome in both the CFM beamformer and the MGB case.

Furthermore, it is interesting to note that, for the full antenna-transceiver system (Figures 5.6 and 5.12), the realised gain at 60° is reduced by more than the expected 3 dB, even in the superdirective beamformer case. This is due to impedance mismatches between the array antenna and the transceiver system. These impedances can be matched for a specific antenna impedance which is valid for one scan angle at one frequency. However, the performance at other scan angles and frequencies will be degraded. The loading conditions implemented to produce the shown results were chosen to provide a balanced performance over scan angle and frequency. These results can be improved by designing dual band elements that are matched at both Tx and Rx frequencies, or by implementing a frequency dependent matching network.

Nevertheless, the results show conclusively that the optimum beamformer weights and excitations can be calculated for a given array antenna-transceiver system. These results offer increased gain performance over the conventional uniform amplitude beamformer. This justifies their use as a starting point for applying further radiation pattern constraints as well as a reference beamformer against which the gain cost for implemented pattern constraints can be evaluated.

5.6.2 Mask Application Beamformers

The results presented in Section 5.4 show that the radiation pattern can be altered to meet the specifications of a given mask for a small gain cost. However, this assumes that the specified mask is not too strict and physically possible to meet for the given system. When given an overly restrictive mask, the resulting radiation will either violate the mask at some points, or reduce the overall gain drastically in an attempt to adhere to the mask requirements.

5.6.3 Isolation Beamformers

From the figures presented in Section 5.3, it is clear that the isolation between Rx and Tx ports can be improved greatly for a small gain cost. However, it is important to note that this isolation is only valuable for signal improvement after the beamformers. The isolation improvement does not account for the amount of

power present on the LNAs in the receiver beamformer ICs and hence does not aid in preventing LNA saturation. However, assuming enough filtering is present to prevent saturation of the LNAs, either through spacial separation of the Tx and Rx elements or physical filters present on the RF chain, this method of increasing isolation can improve SINR or prevent saturation of the rest of the receiver chain.

The Monte Carlo analyses on the isolation beamformer performance show that the achieved isolation is very sensitive to perturbations in beamformer amplitude and phase. However, within the resolution of the Monte Carlo analysis (0.5 dB in amplitude and 6° in phase), an additional isolation of 30-60 dB is typically achieved. Future work should focus on identifying the fundamental reason behind this sensitivity and mitigating it within the isolation beamformer algorithm.

It is also interesting to consider the isolation performance at broadside, which is greatly reduced over a scan angle similar to the half-power beamwidth. This is most likely due to a lack of flexibility in possible values for the phase of the beamformer weights. When scanning to broadside, all the antenna elements have approximately the same phase, which results in a narrower solution space. Another way to explain this result is to consider the similarities in beam and sidelobe shape and location between the receive and transmit gain patterns at broadside. When scanning to further than the half-power beamwidth, the sidelobes can be better redistributed to improve isolation while scanning to the correct location.

That being said, some isolation improvement at broadside can be seen in the case where the algorithm is iterated and both receive and transmit beamformers are adjusted. This is due to the additional flexibility of changing both the Rx and Tx beamformer weights, indicating that if a solver were to be created which iteratively steps between the two beamformers to reach the target isolation, improved broadside isolation performance is achievable.

5.6.4 Joint Mask and Isolation Beamformers

The joint isolation and mask results require a specified target isolation per scanning point along with the adjusted mask. This differs from the isolation results in which the target isolation is stepped in order to provide an isolation given a reduction in gain. Hence, generation of these results posed quite a challenge in choosing suitable isolation values as the additional mask constraints not only places additional strain on the beamformer but changes the initial isolation. Ideally the target isolation would be determined by looping the solver and allowing a certain reduction in gain. However, the time required for such a simulation far exceeds what is available for this work.

From Figure 5.21 it appears that the target isolation was overshot in some cases. However, the isolation is specified to be met over all frequency points, thus, the results given depict satisfying the minimum isolation over all frequency points with the lowest gain reduction.

The mask for both transmit and receive appear to be applied incorrectly when scanning to 60° in that the main beam seems to be attenuated more than is required. There are two reasons for this: firstly, this is due to the combination of strict isolation and mask adherence criteria as well as relaxed convergence criteria. A more optimal solution could potentially be obtained with stricter convergence criteria and a longer simulation time. The second reason for the large gain loss at 60° is that the mask is applied in a 2D plane. Although not visible from these plots, there are sidelobes in other ϕ-θ planes that the algorithm takes into account. Therefore the large gain cost observed may be necessary to ensure complete mask adherence.

6

Conclusions and Recommendations

In this section we summarise the results of the previous section and provide some general conclusions. The achieved performance is related back to the initial specifications stated in Chapter 1. Thereafter we give recommendations for future work that needs to be completed before this work can be applied to practical SatCom systems. We conclude with a few closing thoughts.

6.1 General Conclusions

In this work a mathematical model of a complete FDD K/Ka-band SatCom phased array communication system was implemented. This model can be used to derive optimal beamformer coefficients for the system, and to evaluate the performance of the system if those beamformer coefficiencts are applied. The model takes into account the mutual coupling effects as well as the interactions between the array antenna and the transceiver system. Noise is omitted from the model in this work, although CAESAR simulation tool provides the structures to include internal or external noise in the analyses.

Beamformer coefficients were derived to provide optimal realised system gain in both the transmit and the receive scenarios. These were used as reference beamformers, against which the gain cost of adding further pattern constraints could be measured. Furthermore, beamformer algorithms were implemented to calculate coefficients that offered increased isolation between the transmit and receive ports, as well as coefficients that would ensure adherence to a set mask. As a last step, these three performance metrics were combined to produce beamformer coefficients with good isolation and mask adherence that are as close as possible to the optimum realised gain solution.

It was found that the isolation between the receive and transmit ports could be vastly improved for a very small gain cost. However, with only a single iteration of the algorithm, the resulting beamformer was found to be very sensitive to small amplitude and phase variations. Furthermore, the achievable isolation at broadside

(specifically within the broadside half power beamwidth) for a given gain cost is significantly lower than at other scan angles. However, with a second iteration, both of these challenges were somewhat mitigated. It is believed that the isolation algorithm can be further optimised and the beamformers incrementally iterated to provide better performance and robustness.

Furthermore, although the isolation beamformer algorithm offers a significant improvement in Tx-Rx isolation, it is not useful in preventing the saturation of the receive LNAs due to interference from the transmitter subsystem. This means that the need for filtering cannot be completely eliminated by the implementation of this beamformer. However, filtering requirements can be relaxed such that they only need to prevent LNA saturation, while further signal-to-interference ratio improvement can be accounted for by the beamformer.

Applying a radiation mask to a system with an aperture size sufficient to fulfil the main lobe restrictions proved to be possible for a (mostly) small gain cost. Notably, the gain cost increased at high scan angles due to widening of the main beam. This could be addressed by designing the elements and array such that a sufficiently narrow beam is present at high scan angles. Notably, the mask adherence beamformer is remarkably robust against amplitude and phase perturbations.

Lastly, the combinations of the optimum gain, isolation and mask adherence beamformers showed that good performance metrics were attainable for a slight gain cost. The gain cost was dominated by the mask application beamformer, especially at high scan angles. The single iteration isolation beamformers were calculated for an isolation of only 60 dB, which proved to be more robust against perturbance than the case where stricter isolation was required. However, the isolation performance remains very sensitive to even small changes in beamformer amplitude and phase.

With reference to the performance specifications listed in Tables 1.1 and 1.2, it is worth noting that the adjusted system specifications are all met with this model and the calculated beamformers. It is therefore not unreasonable to conclude that the same model could be used to design an expanded system that could meet the SatCom specifications, if adequate time and computation resources are available. Furthermore, with the performance gain seen in this work due to careful beamformer design, it is possible that such a system could be realised while adhering to strict size, weight and cost constraints.

Two notable aspects in Table 1.1 that were not considered in this work are the axial ratio of the electromagnetic radiation and the gain flatness over scan angle. These performance requirements can be addressed through careful antenna element and array design, as well as evaluating the matching of the antenna to the transceiver system over scan angle.

6.2 Recommendations for Future Work

This work shows that the careful selection of beamformer coefficients for a given system can provide improved performance. However, certain considerations need to be taken into account before the methodology can be applied to a practical system.

6.2.1 Choice of Solver

The first aspect is concerned with the tools used in determining the beamformers. At the heart of the beamformer solutions is a convex problem solver. Therefore the beamformer criteria need to be stated in a way that adheres to the DCP ruleset of the chosen convex solver. This approach was based on initial simulations that were set up for a simple antenna system which did not include any interactions between the antenna and the system. However, for a more complex system, complex statements for gain and isolation are present which do not conform this ruleset. These had to be approximated and adjusted to be useful within the given solver. Investigation into another solver would prove useful not only to improve accuracy but also to decrease simulation time, as many iterations are required to achieve the desired accuracy in the current methodology.

6.2.2 Practical Beamformer Resolution

The methodology implemented in this work assumes ideal beamformers with infinite resolution. Typical beamformers have a fixed amplitude and phase resolution of 0.5 dB and 5.6° respectively. From the Monte Carlo analyses conducted with these ranges, it is clear that some results are very sensitive to beamformer perturbations. By considering a practical beamformer resolution, a solution space of discrete values can be identified. Furthermore, within this solution space, it is expected that the beamformer accuracy is limited by half a step size. Therefore, the resulting variation in results will be similarly limited. Future work should focus on limiting the solver to consider only this solution space and identifying beamformers that are robust against variations within practical beamformer resolutions.

6.2.3 Isolation Beamformer Algorithm

With regard to the methodology used in calculating the isolation beamformers, there are two major areas for improvement, namely the DCP ruleset, and method used to iterate the algorithm. The first challenge is that the isolation power definition is a quadratic statement, which leads to a quadratic criteria, does not conform to the solver statement ruleset. To circumvent this problem, it is stated that the magnitude of the voltage at the receiver beamformer output is proportional to the output power given at the same point, given a fixed load. Therefore the isolation voltage definition, which is not based on a quadratic statement, is used in the algorithm instead. Even though this is a valid solution to the problem, it involves extra computational steps that cost time and computational power. This highlights the need to investigate alternatives to the convex toolbox solver.

The second challenge with the isolation beamformer algorithm is in how to implement iteration between the Rx and Tx beamformer adjustment. The current limitation is once again the convex toolbox, which does not allow multiple subjects within one solver, and therefore limits the algorithm to consider the transmit and receive beamformers independently. One solution to this would be to identify a different solver. Alternatively, some improvements can be made within the given method. Currently one beamformer (for example, the Rx beamformer) is optimised and fixed within the given framework, and is then used as a reference in the framework when its counterpart (the Tx beamformer) is calculated. Thus the isolation beamformers are determined in two steps. However, this solution is sub-optimal, as the reference beamformer that was used in the optimisation criteria in the first step has now changed. Therefore the first beamformer must be re-calculated to take its new counterpart into account, which in turn means that the beamformer calculated in the second step is now sub-optimal. A suggestion to overcome this challenge is to introduce multiple incremental steps between beamformer adjustments, which would allow one to approach the optimal solution of both beamformers through an iterative process.

6.2.4 Antenna Element Design

Outside of the beamformer methodologies and solvers, improvements can be made in the choice of antenna elements. Specifically, dual-band elements that are well-matched in both the Rx and Tx bands are required. With these elements, the system model could be extended to consider the original target frequency bands stated in Table 1.1. While the current work is useful in identifying fundamental performance trade-offs and building a general understanding of the system capabilities, a complete model at the target frequencies that includes the interactions between the two bands of interest is required. Furthermore, the limitations in the beamformer solutions for both mask adherence and isolation can be addressed by the antenna element design - for example, a physical separation of the Tx and Rx apertures will result in increased isolation between the bands.

6.2.5 Electromagnetic Solvers

Currently the system is set up to work with a MoM solver within the MATLAB environment. However, this solver is limited to fully metallic structures that have to be drawn by defining polygons. Other electromagnetc simulation software, such as Ansys HFSS or CST Studio Suite, exist that can provide the same information as the internal MoM solver. Thus, if integration between the results from these software packages and MATLAB can be achieved, they can be used to calculated optimum beamformers. This allows future work to be expanded to radiating element designs and antenna geometries that are only limited by the capabilities of current commercial electromagnetic software packages.

6.2.6 System Modelling

The framework for including noise in the current system model exists in the CAESAR simulation tool, but has not been used in this implementation. By including noise in future analyses, the design of optimum beamformer can be expanded to consider an optimum SINR criteria in the receive scenario. In fact, both internal and external noise sources can be specified within this framework and compensated for by beamformer design. Furthermore, the internal noise generated by the system would influence not only the optimum SINR beamformer, but also the design of the isolation beamformers. For example, increased isolation could be specified in the frequencies in which higher noise from the system is present, such as spurious noise from the PAs.

6.3 Closing Thoughts

Our work has provided a basis on modelling the antenna and system to determine optimum beamformers as well as beamformers designed to improve performance or meet requirements and specifications. While the current setup is still fairly simplistic, the knowledge gained in phased arrays and excitations is monumental. Our understanding from the basic principles of a phased array to the more complex and involved processes of optimum beamformers and implementations has improved drastically.

The basis we provide also points out clear directions for improvement, further work, and development, as well as showing the usefulness in practical implementation. We are grateful for the opportunity provided to us by both Satcube and Chalmers which enabled us to pursue this project. We are particularly indebted to Lukas Nyström for his support, guidance, and understanding throughout, and Rob Maaskant and Marianna Ivashina for their endless support, guidance, and patience.

Bibliography

[1] K. F. Warnick, D. B. Davidson, and D. Buck, "Embedded Element Pattern Loading Condition Transformations for Phased Array Modeling," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 3, pp. 1769–1774, 2021.

[2] R. Maaskant, "Analysis of Large Antenna Systems," Ph.D. dissertation, Technische Universiteit Eindhoven, 2010.

[3] Bayer, Hendrik and Krauss, Alexander and Zaiczek, Tobias and Stephan, Ralf and Enge-Rosenblatt, Olaf and Hein, Matthias A., "Ka-Band User Terminal Antennas for Satellite Communications [Antenna Applications Corner]," *IEEE Antennas and Propagation Magazine*, vol. 58, no. 1, pp. 76–88, 2016.

[4] Y. Kohda, K. Takano, D. Nakano, N. Ohba, T. Yamane, and Y. Katayama, "Single-channel Full-duplex mmWave Fink using Phased-array for Ethernet," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015, pp. 400–405.

[5] Aryafar, Ehsan and Keshavarz-Haddad, Alireza, "PAFD: Phased Array Full-Duplex," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 261–269.

[6] Yin, Bei and Wu, Michael and Studer, Christoph and Cavallaro, Joseph R. and Lilleberg, Jorma, "Full-duplex in Large-scale Wireless Systems," in *2013 Asilomar Conference on Signals, Systems and Computers*, 2013, pp. 1623–1627.

[7] Tabarani, Filipe and Boccia, Luigi and Calzona, Domenico and Amendola, G. and Schumacher, Hermann, "Power Efficient Full-duplex K/Ka-band Phased Array Front-End," *IET Microwaves, Antennas & Propagation*, vol. 14, 12 2019.

[8] O. Iupikov, "Digital beamforming focal plane arrays for radio astronomy and space-borne passive remote sensing," Ph.D. dissertation, Chalmers Tekniska Hogskola (Sweden), 2017.

[9] R. Maaskant and B. Yang, "A Combined Electromagnetic and Microwave Antenna System Simulator for Radio Astronomy," in *2006 First European Con-*

ference on Antennas and Propagation. IEEE, 2006, pp. 1–4.

[10] M. V. Ivashina, R. Maaskant, and B. Woestenburg, "Equivalent System Representation to Model the Beam Sensitivity of Receiving Antenna Arrays," *IEEE Antennas and Wireless Propagation Letters*, vol. 7, pp. 733–737, 2008.

[11] C. Fager, K. Hausmair, K. Buisman, K. Andersson, E. Sienkiewicz, and D. Gustafsson, "Analysis of Nonlinear Distortion in Phased Array Transmitters," in *2017 Integrated Nonlinear Microwave and Millimetre-wave Circuits Workshop (INMMiC)*, 04 2017, pp. 1–4.

[12] A. R. Vilenskiy, W.-C. Liao, R. Maaskant, V. Vassilev, O. A. Iupikov, T. Emanuelsson, and M. V. Ivashina, "Co-design and validation approach for beam-steerable phased arrays of active antenna elements with integrated power amplifiers," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 11, p. 7497–7507, 2021.

[13] M. Ivashina, O. Iupikov, R. Maaskant, W. Cappellen, and T. Oosterloo, "An optimal beamforming strategy for wide-field surveys with phased-array-fed reflector antennas," *Antennas and Propagation, IEEE Transactions on*, vol. 59, pp. 1864 – 1875, 07 2011.

[14] W.-C. Liao, A. R. Vilenskiy, R. Maaskant, T. Emanuelsson, V. Vassilev, and M. V. Ivashina, "mmwave metal bowtie slot array element integrating power amplifier mmic via on-chip probe to enhance efficiency and bandwidth," *IEEE Transactions on Antennas and Propagation*, 2022.

[15] Yin, Yusheng and Zihir, Samet and Kanar, Tumay and Ma, Qian and Chung, Hyunchul and Gao, Li and Rebeiz, Gabriel M., "A 37–42-GHz 8 × 8 Phased-Array With 48–51-dBm EIRP, 64–QAM 30-Gb/s Data Rates, and EVM Analysis Versus Channel RMS Errors," *IEEE Transactions on Microwave Theory and Techniques*, vol. 68, no. 11, pp. 4753–4764, 2020.

[16] C. D. Nwankwo, L. Zhang, A. Quddus, M. A. Imran, and R. Tafazolli, "A survey of self-interference management techniques for single frequency full duplex systems," *IEEE Access*, vol. 6, pp. 30 242–30 268, 2017.

[17] K. F. Warnick, R. Maaskant, M. V. Ivashina, D. B. Davidson, and B. D. Jeffs, *Phased Arrays for Radio Astronomy, Remote Sensing, and Satellite Communications*, ser. EuMA High Frequency Technologies Series. Cambridge University Press, 2018.

[18] Xiao, Zhenyu and Xia, Pengfei and Xia, Xiang-Gen, "Full-Duplex Millimeter-Wave Communication," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 136–143, 2017.

[19] Singh, Hema and H L, Sneha and Jha, Rm, "Mutual Coupling in Phased Arrays: A Review," *International Journal of Antennas and Propagation*, vol. 2013, 01 2013.

[20] IEEE, "IEEE Standard Definitions of Terms for Antennas," *IEEE Std 145-1993*, pp. 1–32, 1993.

[21] S. Blanch, J. Romeu, and I. Corbella, "Exact Presentation of Antenna System Diversity Performance from Input Parameter Description," *Electronics Letters*, vol. 39, pp. 705 – 707, 06 2003.

[22] I. Salonen and P. Vainikainen, "Estimation of Signal Correlation in Antenna Arrays," in *JINA 2002 International Symposium on Antennas, Nice, France, November 12-14, 2002*, 2002, pp. 383–386.

[23] D. M. Pozar, *Microwave engineering*. John wiley & sons, 2011.

[24] K. Kurokawa, "Power waves and the scattering matrix," *IEEE transactions on microwave theory and techniques*, vol. 13, no. 2, pp. 194–202, 1965.

[25] L. Liberti and N. Maculan, *Global optimization: from theory to implementation*. Springer Science & Business Media, 2006, vol. 84.

[26] S. Boyd and M. C. Grant, "The cvx users' guide-release 2.2," Technical report, 2020.(Cited on page 60.), Tech. Rep., 2020.

[27] A. J. Van Katwijk, A. Neto, G. Toso, and D. Cavallo, "Design of wideband wide-scanning dual-polarized phased array covering simultaneously both the ku-and the ka-satcom bands," in *2020 14th European Conference on Antennas and Propagation (EuCAP)*. IEEE, 2020, pp. 1–3.

[28] A. I. Sandhu, E. Arnieri, G. Amendola, L. Boccia, E. Meniconi, and V. Ziegler, "Radiating elements for shared aperture tx/rx phased arrays at k/ka band," *IEEE Transactions on Antennas and Propagation*, vol. 64, no. 6, pp. 2270–2282, 2016.

[29] T. Chaloun, C. Hillebrand, C. Waldschmidt, and W. Menzel, "Active transmitarray submodule for k/ka band satcom applications," in *2015 German Microwave Conference*. IEEE, 2015, pp. 198–201.

[30] M. Ferrando-Rocher, J. I. Herranz-Herruzo, A. Valero-Nogueira, and B. Bernardo-Clemente, "Full-metal k-ka dual-band shared-aperture array antenna fed by combined ridge-groove gap waveguide," *IEEE Antennas and Wireless Propagation Letters*, vol. 18, no. 7, pp. 1463–1467, 2019.

[31] D. Sánchez-Escuderos, J. I. Herranz-Herruzo, M. Ferrando-Rocher, and

A. Valero-Nogueira, "True-time-delay mechanical phase shifter in gap waveguide technology for slotted waveguide arrays in ka-band," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 5, pp. 2727–2740, 2020.

[32] H. DE, "The n-port receiving antenna and its equivalent electrical network." 1975.

[33] A. De Hoop and G. De Jong, "Power reciprocity in antenna theory," in *Proceedings of the Institution of Electrical Engineers*, vol. 121, no. 10. IET, 1974, pp. 1051–1056.

[34] R. Hunger, "An introduction to complex differentials and complex differentiability," 2007.

A

Definitions

Ansys	Refers to Ansys HFSS, a 3D electromagnetic simulation software tool.
Antenna Gain	The ratio of radiated power to accepted power (P_R/P_O) as defined by IEEE in Figure 2.3.
Convex Problem	A convex problem, here referring to a convex optimisation problem, is a problem where all of the constraints are convex functions. Additionally the objective must also be a convex function if minimising, or a concave function if maximising (i.e., only one optimal solution, the global optimum, should exist).
Embedded Element Pattern	The radiation pattern of a given element including the coupling effects of the surrounding elements.
Mutual Coupling	Refers to the radiation and scattering effects of adjacent elements in an array.
Realised Gain	The ratio of radiated power to power available from the generator (P_R/P_A) as defined by IEEE in Figure 2.3.
System Gain	The ratio of power accepted by the antenna to power available from the generator (P_O/P_A) using the definitions given in Figure 2.3.

B

Acronyms

CFM	Conjugate Field Matched
CST	Computer Simulation Technology
DCP	Disciplined Convex Programming
EEP	Embedded Element Pattern
EIRP	Effective Isotropic Radiated Power
FEM	Finite Element Method
HFSS	High Frequency Structural Simulator
IC	Integrated Circuit
LEO	Low Eart Orbit
LNA	Low Noise Amplifier
MGB	Maximum Gain Beamformer
MoM	Method of Moments
MRGB	Maximum Realized Gain Beamformer
PA	Power Amplifier
RAM	Random Access Memory
SatCom	Satellite Communication
SINR	Signal to Interference and Noise Ratio
SLL	Side-lobe levels
SNR	Signal to Noise Ratio

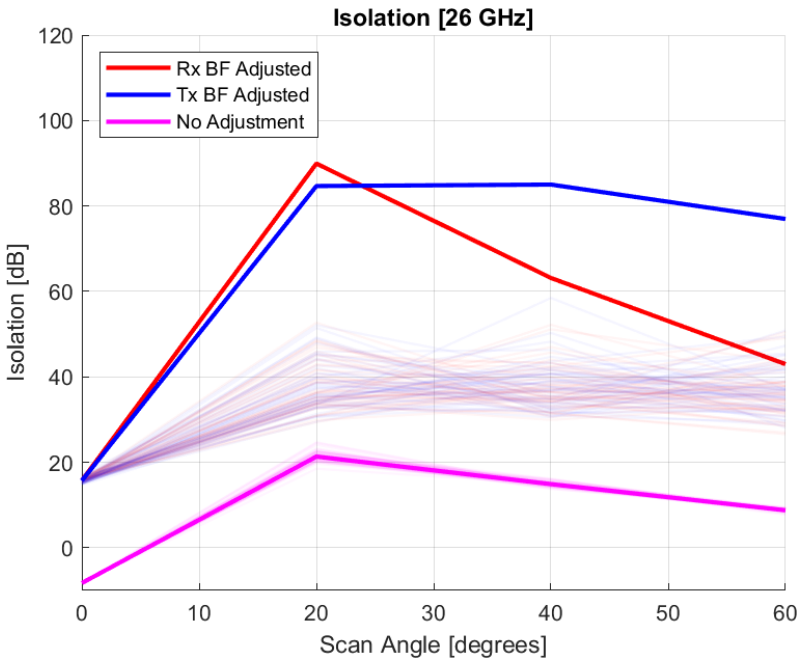
C

Extra Results

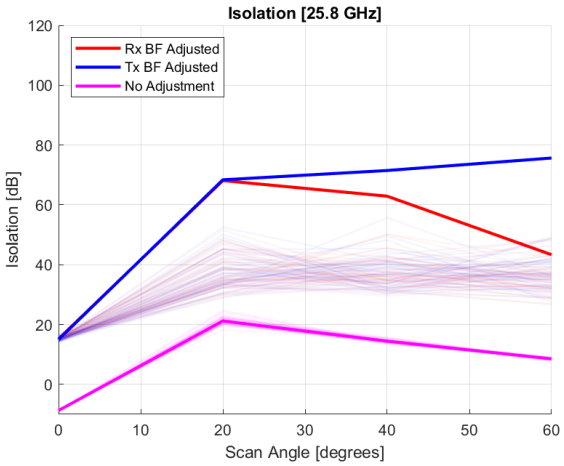
C.1 Joint Mask and Isolation Beamformer Results with Strict Isolation Requirements

C.1.1 Initial Simulation Setup

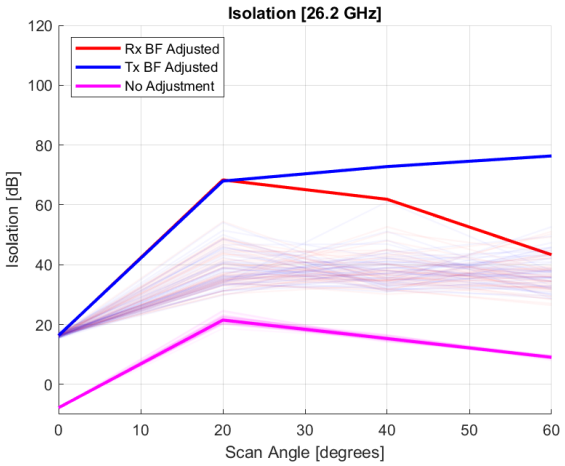
In the initial simulation setup, the achievable isolation performance of the system was not properly taken into account. Instead, a strict isolation requirement of at least 75 dB was specified over all scan angles for both the receive and transmit cases. The isolation results are shown in C.1.



(a) Centre frequency



(b) Lower frequency bound

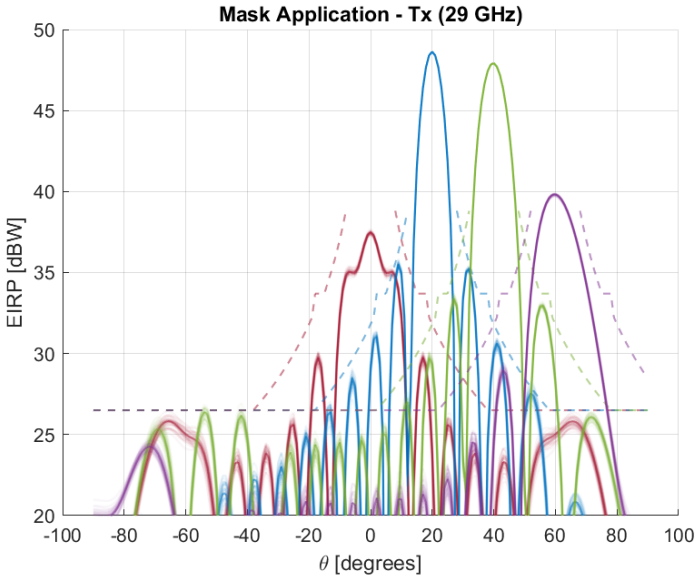


(c) Upper frequency bound

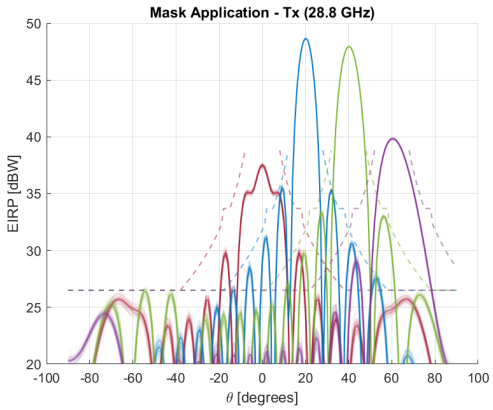
Figure C.1: Achieved isolation when specifying isolation over bandwidth per scan angle

The isolation requirement is met with the Tx beamformer for scan angles of $> 20°$. For the Rx beamformer, the isolation requirement is only met at a scan angle of $20°$. In the cases where the isolation is not met, the algorithm reached one of it's specified convergence criteria without achieving its goal. Notably, the isolation performance is very poor at broadside.

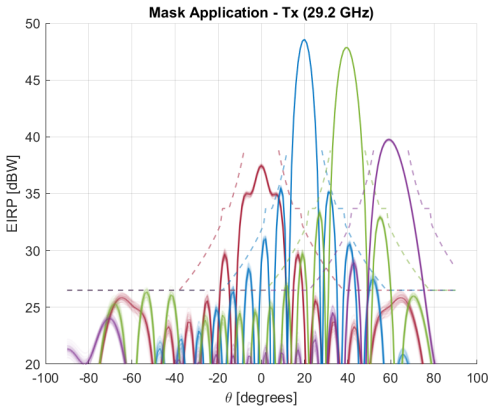
The corresponding receive and transmit gain patterns over scan angle are shown in Figure C.2 - C.3. From these figures, the gain cost of achieving the specified isolation is clear. At broadside, the shape of the beam has been completely warped by the algorithm to try and meet the specifications.



(a) Centre frequency

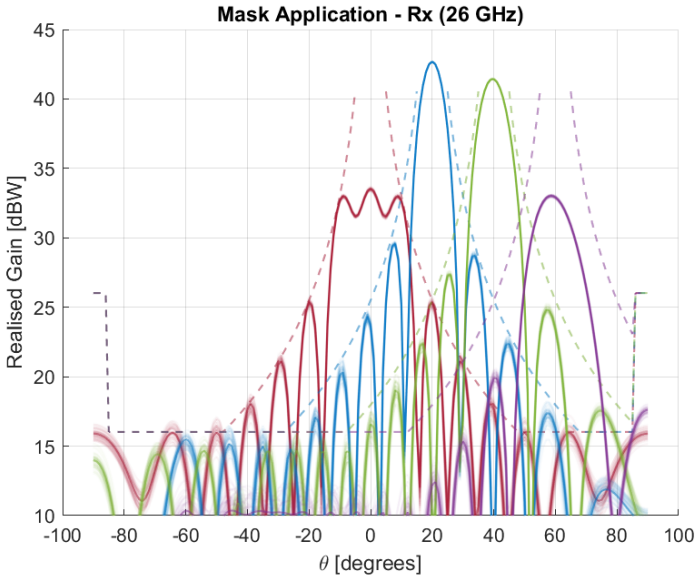


(b) Lower frequency bound

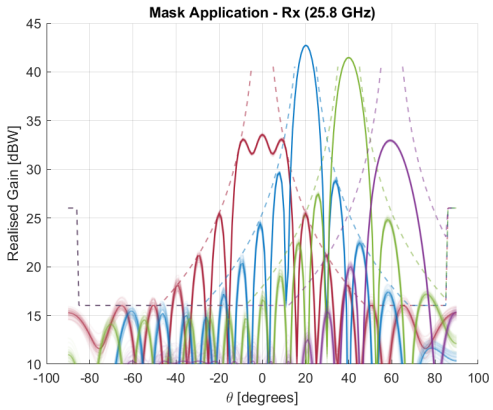


(c) Upper frequency bound

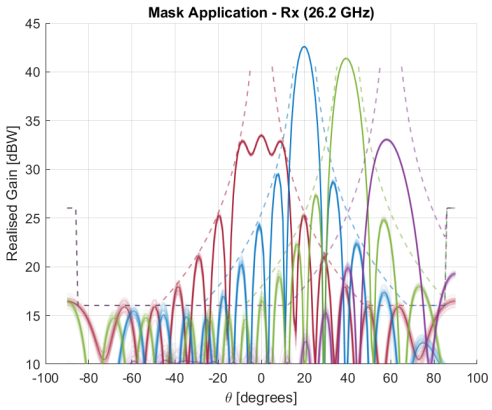
Figure C.2: Transmit mask adherence with specified isolation applied



(a) Centre frequency



(b) Lower frequency bound



(c) Upper frequency bound

Figure C.3: Receive mask adherence with specified isolation applied

D

Code

D.1 Projects

D.1.1 Main.m

```
%                      ____________________
%                     ||                    ||
%                     ||      A R R A Y     ||
%                     ||____________________||
%                        |      |      |
%                      3N+1   3N+2 .. 4*N
%                       _|_    _|_    _|_
%                       | |    | |    | |
%                       | |    | |    | |
%                      N+1    N+2 .. 2*N
%                     ___|_____|_____|____
%                     ||     ARRAY PORTS    ||
% Zo--4N+1--|--1--|| ___                   _     ||--2N+1--|--5N+1--Zo
% Zo--4N+2--|--2--||  | \/       |_| \/ ||--2N+2--|--5N+2--Zo
%       :       :   ||  | /\       | \ /\ ||     :       :
% Zo--5N----|--N--||                       ||--3N----|--6N----Zo
%                     ||____________________||

clc
clear
close all

%% User defined constants

% Settings
determineEandY          = false;
determineLoadedEEPs     = false;

objSaveName = '256square10phi3f';

% Rx and Tx frequencies
F_TX             = 29e9;
F_RX             = 26e9;
BW               = 0.4e9;
nFreqPoints      = 3;
dipoleResonance = 29e9;


% set array geometry
nElem            = 256;
arrayShape      = 'square';
% array geometry for rectangular arrays
nElemX = 2;
nElemY = 2;

xElemLambdaSpacing = 0.52;
yElemLambdaSpacing = 0.50;

% scanning and resolution
```

```
thetaStep       = 1;
phiStep         = 10;
thetaScanDeg    = (0:10:60);
phiScanDeg      = 90;

% microwave component parameters
gainTxPathdBRxFreq  = 20;
gainTxPathdBTxFreq  = 28;
gainRxPathdBRxFreq  = 20;
gainRxPathdBTxFreq  = 18;
delayTxPathRad      = 0;
delayRxPathRad      = 0;
ZoRxSubSys          = 450;
ZoTxSubSys          = 450;
ZoAnt               = 400;

% set filter constraints
rxBPFpass = 0.5;
rxBPFreject = 20;
rxBPFdelay = 0.3;  % rad

txNotchPass = 0.5;
txNotchReject = 20;
txNotchDelay = 0.3;  % rad

%% Functional constants

% Microwave items
gainTxPathRxFreq   = 10^(gainTxPathdBRxFreq/20);
gainTxPathTxFreq   = 10^(gainTxPathdBTxFreq/20);
gainRxPathRxFreq   = 10^(gainRxPathdBRxFreq/20);
gainRxPathTxFreq   = 10^(gainRxPathdBTxFreq/20);

PA_S21RxFreq  = gainTxPathRxFreq * ...
                (cos(delayTxPathRad) + 1j * sin(delayTxPathRad));
PA_S21TxFreq  = gainTxPathTxFreq * ...
                (cos(delayTxPathRad) + 1j * sin(delayTxPathRad));
LNA_S21RxFreq = gainRxPathRxFreq * ...
                (cos(delayRxPathRad) + 1j * sin(delayRxPathRad));
LNA_S21TxFreq = gainRxPathTxFreq * ...
                (cos(delayRxPathRad) + 1j * sin(delayRxPathRad));

% Frequency parameters
lambda          = 3e8 / dipoleResonance;

if nFreqPoints == 1
    F_RX_ARRAY = F_RX;
    F_TX_ARRAY = F_TX;
else
    F_RX_ARRAY      = linspace(F_RX-BW/2, F_RX+BW/2, nFreqPoints).';
    F_TX_ARRAY      = linspace(F_TX-BW/2, F_TX+BW/2, nFreqPoints).';
end

FREQ            = [F_TX_ARRAY; F_RX_ARRAY];

% Microwave network constants
if strcmp(arrayShape, 'square')
    nChannels           = determine_nelem_square(nElem);
elseif strcmp(arrayShape, 'hex')
    [nChannels, ~]      = determine_nelem_ncircle_hex_array(nElem);
elseif strcmp(arrayShape, 'rect')
    nChannels           = nElemX * nElemY;
else
    nChannels           = nElem;
end

portsArrayElement   = (1:1:nChannels);
portsTx             = (nChannels+1:1:2*nChannels);
portsAnt            = (2*nChannels+1:1:3*nChannels);
portsRx             = (3*nChannels+1:1:4*nChannels);
portsLoadTx         = (4*nChannels+1:1:5*nChannels);
portsLoadRx         = (5*nChannels+1:1:6*nChannels);

% Create useful arrays and variables
```

```
123 thetaScanRad     = thetaScanDeg * pi / 180;
124 phiScanRad       = phiScanDeg * pi / 180;
125
126 [thetaMesh, phiMesh]     = meshgrid( (0 : thetaStep : 90)/180*pi, ...
127                                       (0 : phiStep : 360-phiStep)/180*pi );
128
129 % determine the phi and theta scan closest to existing number in the array
130 [~, index]            = min(abs(phiMesh - phiScanRad));
131 [phiScanRad, ~]       = max(phiMesh(index));
132
133 tempThetaMesh         = thetaMesh.';
134 for a = 1:length(thetaScanRad)
135     [~, index]             = min(abs(tempThetaMesh - thetaScanRad(a)));
136     [thetaScanRad(a), ~]     = max(tempThetaMesh(index));
137 end
138
139 % create ease-of-use arrays from meshes and cutting planes
140 thetaCutRange    = [-1 .* flip(thetaMesh(1, 2:end))'; thetaMesh(1, :)'];
141 thetaRangeDeg    = thetaCutRange * 180 / pi;
142 thetaArray       = thetaMesh(1, :)';
143 phiArray         = phiMesh(:, 1);
144
145 % create nice degree arrays for plotting
146 thetaArrayDeg        = thetaArray * 180 / pi;
147 phiArrayDeg          = phiArray * 180 / pi;
148
149 % filters
150 sParamRxBPF = create_filter( ...
151      'FREQ', FREQ ...
152     ,'F_PASS_ARRAY', F_RX_ARRAY ...
153     ,'aPass', 10 ^ (-1 * rxBPFpass / 20) ...
154     ,'phase', rxBPFdelay ...
155     ,'aStop', 10 ^ (-1 * rxBPFreject / 20)...
156     );
157
158 sParamTxNotch = create_filter( ...
159      'FREQ', FREQ ...
160     ,'F_PASS_ARRAY', F_RX_ARRAY ...
161     ,'aPass', 10 ^ (-1 * txNotchReject / 20) ...
162     ,'phase', txNotchDelay ...
163     ,'aStop', 10 ^ (-1 * txNotchPass / 20)...
164     );
165
166 %% Array geometry
167 elementPos = define_array_geometry( ...
168     'lambda', lambda, ...
169     'arrayShape', arrayShape, ...
170     'xElemLambdaSpacing', xElemLambdaSpacing, ...
171     'yElemLambdaSpacing', yElemLambdaSpacing, ...
172     'nElemX', nElemX, ...
173     'nElemY',nElemY,...
174     'nElem', nElem...
175     );
176
177 % Define element geometry
178 element = define_element_geometry( ...
179     'lambda', lambda, ...
180     'elementType', 'hwl_dipole'...
181     );
182
183 % Mesh the array
184 parfor f = 1:length(FREQ)
185     initialConst.Freq = FREQ(f);
186     [initialConst, initialGeom, initialMesh] = LoadConstants(initialConst);
187     lambda = 3e8/FREQ(f);
188     [Const(:, f), Mesh(:, f), Geom(:, f), ...
189         elementPatternParameters(:, f), ...
190         x(:, f), y(:, f), h(:, f)] = ...
191         mesh_array(initialConst, ...
192             initialMesh, initialGeom, element, elementPos, lambda, ...
193             'boundaryNodeResolution', 20);
194 end
195
196 %% Find SC EEP ref plane antenna
```

```
197
198 % Build excitation vector, voltage excitation at the port
199 % Excite element by 1 Volt, others short-circuited (columns are excitation
200 % vectors), for determining Yant and (voltage) embedded element patterns
201 observationDistance = 100*3e8/Const(:, FREQ == min(F_RX, F_TX)).Freq;
202
203 if determineEandY
204     yAnt    = zeros(nChannels, nChannels, 2 * nFreqPoints);
205     EEPs_SC = zeros((size(phiMesh, 1) * size(phiMesh, 2)), ...
206                     3, nChannels, 2 * nFreqPoints);
207     parfor f = 1:2*nFreqPoints
208         elementPatternParameters(:, f).vExcitation  = eye(...
209             length(Geom(:, f).RWGPortIndex));
210         [yAnt(:, :, f), EEPs_SC(:, :, :, f)] = ...
211             calculate_Yant_and_element_pattern(...
212                 'Const', Const(:, f), ...
213                 'Mesh', Mesh(:, f), ...
214                 'Geom', Geom(:, f), ...
215                 'thetaMesh', thetaMesh, ...
216                 'phiMesh', phiMesh, ...
217                 'elementPatternParameters', ...
218                     elementPatternParameters(:, f), ...
219                 'observationDistance', observationDistance...
220                 );
221     end
222     parfor f = 1:2*nFreqPoints
223         for p = 1:3
224         % Calculate incident excitations wave due to plane waves
225             if p == 1
226                 incidentPlaneWavePolarization = [1, 0, 0];
227             elseif p == 2
228                 incidentPlaneWavePolarization = [0, 1, 0];
229             elseif p == 3
230                 incidentPlaneWavePolarization = [0, 0, 1];
231             end
232
233             planeWaveExcitations(:, p, :, f) = ...
234                 incident_wave_antenna_voltages(...
235                     'incidentEWave', incidentPlaneWavePolarization, ...
236                     'freqIndex', f, ...
237                     'Const', Const, ...
238                     'Zo', ZoAnt, ...
239                     'EEPs_SC', EEPs_SC, ...
240                     'yAnt', yAnt, ...
241                     'observationDistance', observationDistance, ...
242                     'Mesh', Mesh, ...
243                     'Geom', Geom, ...
244                     'thetaArray', thetaArray, ...
245                     'phiArray', phiArray);
246         end
247     end
248     name = objSaveName;
249     if isempty(name)
250         save('previousMoM', "EEPs_SC", "yAnt", "planeWaveExcitations");
251     else
252         save([name, 'MoM'], "EEPs_SC", "yAnt", "planeWaveExcitations");
253         save('previousMoM', "EEPs_SC", "yAnt", "planeWaveExcitations");
254     end
255
256 else
257     name = objSaveName;
258     if isempty(name)
259         load('previousMoM');
260     else
261         load([name, 'MoM']);
262     end
263 end
264
265 %% Build Microwave System
266
267 % Const.OutputDirName='MW_Output';
268
269 % ============================================
270 % === Start of input file for MW-Simulator ===
```

```
271 % ==========================================
272
273 MW.FreqArray                          = FREQ;
274 MW.IncidentPlaneWavePolarization     = [];
275 MW.Tsky                              = [];
276 MW.ExternalPorts                     = [];
277 MW.ComputeOverallSparMatrix          = false;
278 MW.ComputeOverallNoiseWaveMatrix     = true;
279
280 % Assign independent excitations to all Tx ports
281 MW.CircuitSourcePorts                = zeros(nChannels, nChannels+1);
282 MW.CircuitSourcePorts(:, 1)          = portsLoadTx;
283 for n = 1:nChannels
284     MW.CircuitSourcePorts(n, 1+n) = 1;
285 end
286
287
288 % --- Build active devices
289 for f = 1:nFreqPoints
290     % Rx Frequencies
291     LNA.S(:, :, nFreqPoints + f) = ...
292             [10^(-20/20),   10^(-50/20);...
293             LNA_S21RxFreq,   10^(-15/20)];
294
295     % Tx Frequencies
296     LNA.S(:, :, f) = ...
297             [10^(-10/20),   10^(-50/20);...
298             LNA_S21TxFreq,   10^(-10/20)];
299
300     % --- Build PA device
301     % Rx Frequencies
302     PA.S(:, :, nFreqPoints + f) = ...
303             [10^(-3/20),   10^(-50/20);...
304             PA_S21RxFreq,   10^(-3/20)];
305
306     % Tx Frequencies
307     PA.S(:, :, f) = ...
308             [10^(-20/20),  10^(-50/20);...
309             PA_S21TxFreq,   10^(-10/20)];
310 end
311
312 LNA.GammaOpt = 0;
313 LNA.Tmin     = 0;
314 LNA.rn       = 4 / 100;
315 LNA.Znorm    = ZoRxSubSys;
316
317 PA.GammaOpt = 0;
318 PA.Tmin     = 0;
319 PA.rn       = 4 / 100;
320 PA.Znorm    = ZoTxSubSys;
321
322 % --- Build Load Terminations
323 LoadRx.Z      = ZoRxSubSys;
324 LoadRx.Ta     = 0;
325 LoadRx.Znorm  = ZoRxSubSys;
326
327 LoadTx.Z      = ZoTxSubSys;
328 LoadTx.Ta     = 0;
329 LoadTx.Znorm  = ZoTxSubSys;
330
331 % Generate antenna => 1 : nChannels
332 [antenna, index] = AntennaSParams(MW, yAnt, ZoAnt, ...
333                                   nChannels, 1);
334
335 % Generate transceiver system => Ports nChannels + 1 : 4*nChannels
336 % [Tx Ports; Ant Ports; Rx Ports]
337 [transceiverSystem, index] = generate_transceiver_system( ...
338     'nChannels', nChannels, ...
339     'freqArray', MW.FreqArray,...
340     'LNA', LNA,...
341     'PA', PA,...
342     'rxBPF', sParamRxBPF,...
343     'txNotchFilter', sParamTxNotch,...
344     'ZoAnt', ZoAnt, ...
```

```
345     'index', index...
346     );
347
348 % Load Ports => 4nChannels+1:6nChannels
349 for n = 1:nChannels
350     [loadArray(n), index] = LoadImpedanceDevice( ...
351         MW, LoadTx.Z, LoadTx.Ta, LoadTx.Znorm, index);
352 end
353
354 for n = 1:nChannels
355     [loadArray(nChannels + n), index] = LoadImpedanceDevice( ...
356         MW, LoadRx.Z, LoadRx.Ta, LoadRx.Znorm, index);
357 end
358
359 % Terminate all Tx and Rx ports in loads
360 MW.PortConnectionTopology(1:nChannels, 1) = portsTx;
361 MW.PortConnectionTopology(1:nChannels, 2) = portsLoadTx;
362 MW.PortConnectionTopology(nChannels+1:2*nChannels, 1) = portsRx;
363 MW.PortConnectionTopology(nChannels+1:2*nChannels, 2) = portsLoadRx;
364 % Connect antenna to antenna ports
365 MW.PortConnectionTopology(2*nChannels+1:3*nChannels, 1) = portsAnt;
366 MW.PortConnectionTopology(2*nChannels+1:3*nChannels, 2) =portsArrayElement;
367
368 % === START of MW-Engine ===
369 resultsMW = RunMWSimulator([antenna, transceiverSystem, loadArray], MW);
370 % === End of MW-Engine ===
371
372
373 %% Compute Loaded Element Patterns
374
375 zSysArray = calculate_system_impedance( ...
376         'MW', MW, ...
377         'MW_system', [transceiverSystem, loadArray], ...
378         'observationPorts', portsArrayElement, ...
379         'nChannels', nChannels, ...
380         'Load', LoadRx...
381         );
382
383 if determineLoadedEEPs
384     rxVoltages          = zeros( ...
385                           (size(phiMesh, 1) * size(phiMesh, 2)), ...
386                           3, nChannels, 2 * nFreqPoints);
387     rxWaves             = zeros( ...
388                           (size(phiMesh, 1) * size(phiMesh, 2)), ...
389                           3, nChannels, 2 * nFreqPoints);
390     EEPs_Tx_Loaded      = zeros(size(EEPs_SC));
391     vAnt                = zeros(nChannels, nChannels, 2*nFreqPoints);
392
393     parfor f = 1:length(FREQ)
394         if any(FREQ(f) == F_TX_ARRAY)
395             for n = 1:nChannels
396                 vAnt(:, n, f) = abToVIP(...
397                     resultsMW.a(portsArrayElement, n, f), ...
398                     resultsMW.b(portsArrayElement, n, f), ...
399                     ZoAnt);
400                 for p = 1:3
401                     EEPs_Tx_Loaded(:, p, n, f) = ...
402                         squeeze(EEPs_SC(:, p, :, f)) * vAnt(:, n, f);
403                 end
404             end
405         end
406
407         if any(FREQ(f) == F_RX_ARRAY)
408
409              [rxVoltages(:, :, :, f),  rxWaves(:, :, :, f)] = ...
410                  calculate_rx_voltages( ...
411                     'nChannels', nChannels, ...
412                     'freqIndex', f, ...
413                     'yAnt', yAnt, ...
414                     'MW', MW, ...
415                     'arrayElementPorts', portsArrayElement, ...
416                     'LoadRx', LoadRx, ...
417                     'LoadTx', LoadTx, ...
418                     'transceiverSystem', transceiverSystem, ...
```

```
                    'RxPorts', portsRx, ...
                    'thetaArray', thetaArray, ...
                    'phiArray', phiArray, ...
                    'loadPorts', [portsLoadTx, portsLoadRx], ...
                    'ZoAnt', ZoAnt, ...
                    'planeWaveExcitations', planeWaveExcitations, ...
                    'sAnt', antenna.S, ...
                    'solverMethod', 's_params'...
                    );
        end
    end
    name = objSaveName;
    if isempty(name)
        save('previousEEPs', "rxVoltages", "rxWaves", "EEPs_Tx_Loaded");
    else
        save([name, 'EEPs'], "rxVoltages", "rxWaves", "EEPs_Tx_Loaded");
        save('previousEEPs', "rxVoltages", "rxWaves", "EEPs_Tx_Loaded");
    end


else
    name = objSaveName;
    if isempty(name)
        load('previousEEPs');
    else
        load([name, 'EEPs']);
    end

end

%% Per Scan Angle Calculation

gainTxMGBAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxSNRAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxConvAnt= zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxCFMAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));

gainTxMGBSys = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxSNRSys = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxConvSys= zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainTxCFMSys = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));

gainRxMGBAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainRxSNRAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainRxConvAnt= zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainRxCFMAnt = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));

gainRxSNRSys = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainRxConvSys= zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
gainRxCFMSys = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));

beamformerTxMG      = zeros(nChannels, length(thetaScanRad));
beamformerTxSNR     = zeros(nChannels, length(thetaScanRad));
beamformerTxCFMAnt  = zeros(nChannels, length(thetaScanRad));
beamformerTxConv    = zeros(nChannels, length(thetaScanRad));

beamformerTxMGSys   = zeros(nChannels, length(thetaScanRad));
beamformerTxSNRSys  = zeros(nChannels, length(thetaScanRad));
beamformerTxCFMSys  = zeros(nChannels, length(thetaScanRad));


beamformerRxMG      = zeros(nChannels, length(thetaScanRad));
beamformerRxSNR     = zeros(nChannels, length(thetaScanRad));
beamformerRxCFMAnt  = zeros(nChannels, length(thetaScanRad));
beamformerRxConv    = zeros(nChannels, length(thetaScanRad));

beamformerRxSNRSys  = zeros(nChannels, length(thetaScanRad));
beamformerRxCFMSys  = zeros(nChannels, length(thetaScanRad));

beamformerTxIso         = zeros(nChannels, length(thetaScanRad));
beamformerRxIso         = zeros(nChannels, length(thetaScanRad));
beamformerTxIso_RxTx    = zeros(nChannels, length(thetaScanRad));
beamformerRxIso_TxRx    = zeros(nChannels, length(thetaScanRad));

```

```
493 gainTxSysIso = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
494 gainRxSysIso = zeros(size(EEPs_SC, 1), length(FREQ), length(thetaScanRad));
495 gainTxSysIso_RxTx = zeros(size(EEPs_SC, 1), length(FREQ), ...
496     length(thetaScanRad));
497 gainRxSysIso_TxRx = zeros(size(EEPs_SC, 1), length(FREQ), ...
498     length(thetaScanRad));
499 isoTxScan    = zeros(length(FREQ), length(thetaScanRad));
500 isoRxScan    = zeros(length(FREQ), length(thetaScanRad));
501 isoTxScan_RxTx = zeros(length(FREQ), length(thetaScanRad));
502 isoRxScan_TxRx = zeros(length(FREQ), length(thetaScanRad));
503 isoMGBScan   = zeros(length(FREQ), length(thetaScanRad));
504
505 gainTxSysMask = zeros(size(EEPs_SC, 1), ...
506                       length(FREQ), ...
507                       length(thetaScanRad));
508 gainRxSysMask = zeros(size(EEPs_SC, 1), ...
509                       length(FREQ), ...
510                       length(thetaScanRad));
511
512 beamformerTxMask = zeros(nChannels, length(thetaScanRad));
513 beamformerRxMask = zeros(nChannels, length(thetaScanRad));
514
515 rxMask = zeros(size(phiMesh, 1), size(phiMesh, 2), length(thetaScanRad));
516 txMask = zeros(size(phiMesh, 1), size(phiMesh, 2), length(thetaScanRad));
517
518 lambda = 3e8 ./ FREQ;
519
520 zSysTxPorts = calculate_system_impedance( ...
521         'MW', MW, ...
522         'MW_system', [transceiverSystem, antenna, ...
523             loadArray(nChannels+1:2*nChannels)], ...
524         'observationPorts', portsLoadTx, ...
525         'nChannels', nChannels, ...
526         'Load', LoadTx...
527         );
528 ySysTxPorts = zeros(size(zSysTxPorts));
529
530 for f = 1:2*nFreqPoints
531     ySysTxPorts(:, :, f) = inv(zSysTxPorts(:, :, f));
532 end
533
534 for a = 1:length(thetaScanRad)
535     tic
536     %% Calculate MGB beamformers at Antenna
537     tic
538     beamformerTxMG(:, a) = calculate_max_gain_beamformer_weights( ...
539         'yAnt', yAnt, ...
540         'EEPs_SC', EEPs_SC, ...
541         'thetaMesh', thetaMesh, ...
542         'phiMesh', phiMesh, ...
543         'thetaScanRad', thetaScanRad(a), ...
544         'phiScanRad', phiScanRad, ...
545         'observationDistance', observationDistance, ...
546         'freqIndex', find(FREQ == F_TX), ...
547         'polarisation', '' ...
548         );
549
550     beamformerRxMG(:, a) = calculate_max_gain_beamformer_weights( ...
551         'yAnt', yAnt, ...
552         'EEPs_SC', EEPs_SC, ...
553         'thetaMesh', thetaMesh, ...
554         'phiMesh', phiMesh, ...
555         'thetaScanRad', thetaScanRad(a), ...
556         'phiScanRad', phiScanRad, ...
557         'observationDistance', observationDistance, ...
558         'freqIndex', find(FREQ == F_RX), ...
559         'polarisation', '' ...
560         );
561
562     %% Calculate CFM beamformers at Antenna
563
564     beamformerTxCFMAnt(:, a) = calculate_cfm_beamformer_weights(...
565         'planeWaveExcitations', planeWaveExcitations, ...
566         'thetaMesh', thetaMesh, ...
```

```
567         'phiMesh', phiMesh, ...
568         'thetaScanRad', thetaScanRad(a), ...
569         'phiScanRad', phiScanRad, ...
570         'freqIndex', find(FREQ == F_TX),...
571         'polarisation', ''...
572         );
573     beamformerTxCFMAnt(:, a) = conj(beamformerTxCFMAnt(:, a));
574 
575     beamformerRxCFMAnt(:, a) = calculate_cfm_beamformer_weights(...
576         'planeWaveExcitations', planeWaveExcitations, ...
577         'thetaMesh', thetaMesh, ...
578         'phiMesh', phiMesh, ...
579         'thetaScanRad', thetaScanRad(a), ...
580         'phiScanRad', phiScanRad, ...
581         'freqIndex', find(FREQ == F_RX),...
582         'polarisation', ''...
583         );
584 
585     %% Calculate Max SNR beamformers at Antenna
586     beamformerTxSNR(:, a) = calculate_max_sens_beamformer_weights( ...
587         'rxWaves', planeWaveExcitations, ...
588         'thetaMesh', thetaMesh, ...
589         'phiMesh', phiMesh, ...
590         'thetaScanRad', thetaScanRad(a), ...
591         'phiScanRad', phiScanRad, ...
592         'freqIndex', FREQ(:) == F_TX, ...
593         'noiseWaveCorrMatrix', antenna.C, ...
594         'polarisation', 'x', ...
595         'sensType', 'SNR'...
596         );
597     beamformerTxSNR(:, a) = conj(beamformerTxSNR(:, a));
598 
599 
600     beamformerRxSNR(:, a) =  calculate_max_sens_beamformer_weights( ...
601         'rxWaves', planeWaveExcitations, ...
602         'thetaMesh', thetaMesh, ...
603         'phiMesh', phiMesh, ...
604         'thetaScanRad', thetaScanRad(a), ...
605         'phiScanRad', phiScanRad, ...
606         'freqIndex', FREQ(:) == F_RX, ...
607         'noiseWaveCorrMatrix', antenna.C, ...
608         'polarisation', 'x', ...
609         'sensType', 'SNR'...
610         );
611     beamformerRxSNR(:, a) = conj(beamformerRxSNR(:, a));
612 
613     %% Calculate conventional beamformers (Antenna and sys)
614 
615     beamformerTxConv(:, a) = calculate_conventional_beamformer( ...
616         'elementPos', elementPos ...
617         ,'thetaScanRad', thetaScanRad(a) ...
618         ,'phiScanRad', phiScanRad ...
619         ,'lambda', lambda(FREQ(:) == F_TX)...
620         );
621 
622     beamformerRxConv(:, a) = calculate_conventional_beamformer( ...
623         'elementPos', elementPos ...
624         ,'thetaScanRad', thetaScanRad(a) ...
625         ,'phiScanRad', phiScanRad ...
626         ,'lambda', lambda(FREQ(:) == F_RX)...
627         );
628 
629     beamformerRxConv(:, a) = conj(beamformerRxConv(:, a));
630 
631     %% Calculate MGB / MSNR for System
632     I = zeros(size(yAnt));
633     for f = 1:2*nFreqPoints
634         I(:, :, f) = eye(nChannels);
635     end
636 
637     beamformerTxMGSys(:, a) = calculate_max_gain_beamformer_weights( ...
638         'yAnt', I, ...
639         'EEPs_SC', EEPs_Tx_Loaded, ...
640         'thetaMesh', thetaMesh, ...
```

```
641         'phiMesh', phiMesh, ...
642         'thetaScanRad', thetaScanRad(a), ...
643         'phiScanRad', phiScanRad, ...
644         'observationDistance', observationDistance, ...
645         'freqIndex', find(FREQ == F_TX), ...
646         'polarisation', 'x' ...
647         );
648
649     beamformerTxSNRSys(:, a) = calculate_max_gain_beamformer_weights( ...
650         'yAnt', antenna.C, ...
651         'EEPs_SC', EEPs_Tx_Loaded, ...
652         'thetaMesh', thetaMesh, ...
653         'phiMesh', phiMesh, ...
654         'thetaScanRad', thetaScanRad(a), ...
655         'phiScanRad', phiScanRad, ...
656         'observationDistance', observationDistance, ...
657         'freqIndex', find(FREQ == F_TX), ...
658         'polarisation', 'x' ...
659         );
660
661     beamformerRxSNRSys(:, a) = calculate_max_sens_beamformer_weights( ...
662         'rxWaves', rxWaves, ...
663         'thetaMesh', thetaMesh, ...
664         'phiMesh', phiMesh, ...
665         'thetaScanRad', thetaScanRad(a), ...
666         'phiScanRad', phiScanRad, ...
667         'freqIndex', FREQ(:) == F_RX, ...
668         'noiseWaveCorrMatrix', resultsMW.Cnet...
669             (portsLoadRx, portsLoadRx, :), ...
670         'polarisation', '', ...
671         'sensType', 'SNR'...
672         );
673      %% Calculate CFM for System
674
675     beamformerTxCFMSys(:, a) = calculate_cfm_beamformer_weights(...
676         'planeWaveExcitations', EEPs_Tx_Loaded, ...
677         'thetaMesh', thetaMesh, ...
678         'phiMesh', phiMesh, ...
679         'thetaScanRad', thetaScanRad(a), ...
680         'phiScanRad', phiScanRad, ...
681         'freqIndex', FREQ(:) == F_TX,...
682         'polarisation', ''...
683         );
684     beamformerTxCFMSys(:, a) = conj(beamformerTxCFMSys(:, a));
685
686     beamformerRxCFMSys(:, a) = calculate_max_sens_beamformer_weights( ...
687         'rxWaves', rxWaves, ...
688         'thetaMesh', thetaMesh, ...
689         'phiMesh', phiMesh, ...
690         'thetaScanRad', thetaScanRad(a), ...
691         'phiScanRad', phiScanRad, ...
692         'freqIndex', FREQ(:) == F_RX, ...
693         'noiseWaveCorrMatrix', antenna.C, ...
694         'polarisation', 'x', ...
695         'sensType', 'CFM'...
696         );
697
698     %% Microwave System with MGBs to scale excitation vectors
699
700     [MW, loadMGBArray] = define_load_source_ports( ...
701         'MW', MW, ...
702         'sourcePortArray', portsLoadTx, ...
703         'excitationArray', beamformerTxMGSys(:, a), ...
704         'Load', LoadTx ...
705         );
706
707     % === START of MW-Engine ===
708     resultsMW_MGB = RunMWSimulator(...
709         [transceiverSystem, ...
710             antenna, ...
711             loadMGBArray, ...
712             loadArray(nChannels+1:2*nChannels)], ...
713         MW);
714     % === End of MW-Engine ===
```

```
715
716     [MW, loadConvArray] = define_load_source_ports( ...
717         'MW', MW, ...
718         'sourcePortArray', portsLoadTx, ...
719         'excitationArray', beamformerTxConv(:, a), ...
720         'Load', LoadTx ...
721         );
722
723     % === START of MW-Engine ===
724     resultsMW_Conv = RunMWSimulator(...
725         [transceiverSystem, ...
726             antenna, ...
727             loadConvArray, ...
728             loadArray(nChannels+1:2*nChannels)], ...
729         MW);
730     % === End of MW-Engine ===
731
732     [MW, loadCFMArray] = define_load_source_ports( ...
733         'MW', MW, ...
734         'sourcePortArray', portsLoadTx, ...
735         'excitationArray', beamformerTxCFMSys(:, a), ...
736         'Load', LoadTx ...
737         );
738
739     % === START of MW-Engine ===
740     resultsMW_CFM = RunMWSimulator(...
741         [transceiverSystem, ...
742             antenna, ...
743             loadCFMArray, ...
744             loadArray(nChannels+1:2*nChannels)], ...
745         MW);
746     % === End of MW-Engine ===
747
748     [MW, loadSNRArray] = define_load_source_ports( ...
749         'MW', MW, ...
750         'sourcePortArray', portsLoadTx, ...
751         'excitationArray', beamformerTxSNRSys(:, a), ...
752         'Load', LoadTx ...
753         );
754
755     % === START of MW-Engine ===
756     resultsMW_SNR = RunMWSimulator(...
757         [transceiverSystem, ...
758             antenna, ...
759             loadSNRArray, ...
760             loadArray(nChannels+1:2*nChannels)], ...
761         MW);
762     % === End of MW-Engine ===
763
764
765     %% Calculate Tx and Rx gain
766
767     parfor f = 1:length(FREQ)
768         if any(FREQ(f) == F_TX_ARRAY)
769             gainTxMGBAnt(:, f, a) = calculate_array_gain( ...
770                 'EEPs_SC', EEPs_SC, ...
771                 'observationDistance', observationDistance, ...
772                 'Const', Const, ...
773                 'beamformer', beamformerTxMG(:, a), ...
774                 'yAnt', yAnt, ...
775                 'freqIndex', f ...
776                 );
777
778             gainTxSNRAnt(:, f, a) = calculate_array_gain( ...
779                 'EEPs_SC', EEPs_SC, ...
780                 'observationDistance', observationDistance, ...
781                 'Const', Const, ...
782                 'beamformer', beamformerTxSNR(:, a), ...
783                 'yAnt', yAnt, ...
784                 'freqIndex', f ...
785                 );
786
787             gainTxConvAnt(:, f, a) = calculate_array_gain( ...
788                 'EEPs_SC', EEPs_SC, ...
```

```
789                 'observationDistance', observationDistance, ...
790                 'Const', Const, ...
791                 'beamformer', beamformerTxConv(:, a), ...
792                 'yAnt', yAnt, ...
793                 'freqIndex', f ...
794                 );
795 
796             gainTxCFMAnt(:, f, a) = calculate_array_gain( ...
797                 'EEPs_SC', EEPs_SC, ...
798                 'observationDistance', observationDistance, ...
799                 'Const', Const, ...
800                 'beamformer', beamformerTxCFMAnt(:, a), ...
801                 'yAnt', yAnt, ...
802                 'freqIndex', f ...
803                 );
804 
805               gainTxMGBSys(:, f, a) = calculate_gain_tx( ...
806                 'EEPs', EEPs_SC, ...
807                 'observationDistance', observationDistance, ...
808                 'Const', Const, ...
809                 'beamformer', beamformerTxMGSys(:, a), ...
810                 'resultsMW', resultsMW_MGB, ...
811                 'freqIndex', f, ...
812                 'ZoAnt', ZoAnt, ...
813                 'ZoTxSubSys', ZoTxSubSys, ...
814                 'inputPorts', portsTx, ...
815                 'outputPorts', portsArrayElement, ...
816                 'gainType', 'radiation' ...
817                 );
818 
819               gainTxConvSys(:, f, a) = calculate_gain_tx( ...
820                 'EEPs', EEPs_SC, ...
821                 'observationDistance', observationDistance, ...
822                 'Const', Const, ...
823                 'beamformer', beamformerTxConv(:, a), ...
824                 'resultsMW', resultsMW_Conv, ...
825                 'freqIndex', f, ...
826                 'ZoAnt', ZoAnt, ...
827                 'ZoTxSubSys', ZoTxSubSys, ...
828                 'inputPorts', portsTx, ...
829                 'outputPorts', portsArrayElement, ...
830                 'gainType', 'radiation' ...
831                 );
832 
833               gainTxCFMSys(:, f, a) = calculate_gain_tx( ...
834                 'EEPs', EEPs_SC, ...
835                 'observationDistance', observationDistance, ...
836                 'Const', Const, ...
837                 'beamformer', beamformerTxCFMSys(:, a), ...
838                 'resultsMW', resultsMW_CFM, ...
839                 'freqIndex', f, ...
840                 'ZoAnt', ZoAnt, ...
841                 'ZoTxSubSys', ZoTxSubSys, ...
842                 'inputPorts', portsTx, ...
843                 'outputPorts', portsArrayElement, ...
844                 'gainType', 'radiation' ...
845                 );
846 
847               gainTxSNRSys(:, f, a) = calculate_gain_tx( ...
848                 'EEPs', EEPs_SC, ...
849                 'observationDistance', observationDistance, ...
850                 'Const', Const, ...
851                 'beamformer', beamformerTxSNRSys(:, a), ...
852                 'resultsMW', resultsMW_SNR, ...
853                 'freqIndex', f, ...
854                 'ZoAnt', ZoAnt, ...
855                 'ZoTxSubSys', ZoTxSubSys, ...
856                 'inputPorts', portsTx, ...
857                 'outputPorts', portsArrayElement, ...
858                 'gainType', 'radiation' ...
859                 );
860         end
861 
862         if any(FREQ(f) == F_RX_ARRAY)
```

```
863
864             gainRxMGBAnt(:, f, a) = calculate_array_gain( ...
865                 'EEPs_SC', EEPs_SC, ...
866                 'observationDistance', observationDistance, ...
867                 'Const', Const, ...
868                 'beamformer', beamformerRxMG(:, a), ...
869                 'yAnt', yAnt, ...
870                 'freqIndex', f ...
871                 );
872
873             gainRxSNRAnt(:, f, a) = calculate_array_gain( ...
874                 'EEPs_SC', EEPs_SC, ...
875                 'observationDistance', observationDistance, ...
876                 'Const', Const, ...
877                 'beamformer', beamformerRxSNR(:, a), ...
878                 'yAnt', yAnt, ...
879                 'freqIndex', f ...
880                 );
881
882             gainRxConvAnt(:, f, a) = calculate_array_gain( ...
883                 'EEPs_SC', EEPs_SC, ...
884                 'observationDistance', observationDistance, ...
885                 'Const', Const, ...
886                 'beamformer', conj(beamformerRxConv(:, a)), ...
887                 'yAnt', yAnt, ...
888                 'freqIndex', f ...
889                 );
890
891             gainRxCFMAnt(:, f, a) = calculate_array_gain( ...
892                 'EEPs_SC', EEPs_SC, ...
893                 'observationDistance', observationDistance, ...
894                 'Const', Const, ...
895                 'beamformer', conj(beamformerRxCFMAnt(:, a)), ...
896                 'yAnt', yAnt, ...
897                 'freqIndex', f ...
898                 );
899
900             gainRxSNRSys(:, f, a) = calculate_system_gain_rx( ...
901                 'rxWaves', rxWaves, ...
902                 'beamformer', beamformerRxSNRSys(:, a), ...
903                 'lambda', lambda, ....
904                 'freqIndex', f, ...
905                 'yAnt', yAnt, ...
906                 'zSys', zSysArray  ...
907                 );
908
909             gainRxConvSys(:, f, a) = calculate_system_gain_rx( ...
910                 'rxWaves', rxWaves, ...
911                 'beamformer', beamformerRxConv(:, a), ...
912                 'lambda', lambda, ...
913                 'freqIndex', f, ...
914                 'yAnt', yAnt, ...
915                 'zSys', zSysArray ...
916                 );
917
918             gainRxCFMSys(:, f, a) = calculate_system_gain_rx( ...
919                 'rxWaves', rxWaves, ...
920                 'beamformer', beamformerRxCFMSys(:, a), ...
921                 'lambda', lambda, ...
922                 'freqIndex', f, ...
923                 'yAnt', yAnt, ...
924                 'zSys', zSysArray ...
925                 );
926         end
927     end
928
929     %% Iso application
930     IsoBandwidthForGain;
931
932     gainTxSysIso(:, :, a) = gainTxIso;
933     gainRxSysIso(:, :, a) = gainRxIso;
934     gainTxSysIso_RxTx(:, :, a) = gainTxIso_RxTx;
935     gainRxSysIso_TxRx(:, :, a) = gainRxIso_TxRx;
936     isoTxScan(:, a) = isoTxBF;
```

```
937     isoRxScan(:, a) = isoRxBF;
938     isoTxScan_RxTx(:, a) = isoTxBF_RxTx;
939     isoRxScan_TxRx(:, a) = isoRxBF_TxRx;
940     isoMGBScan(:, a) = isoMGB;
941
942     %% Mask application
943
944     MaskApplication;
945
946     gainTxSysMask(:, :, a) = gainTxMaskPoint;
947     gainRxSysMask(:, :, a) = gainRxMaskPoint;
948
949     beamformerTxMask(:, a) = beamformerTxMaskPoint;
950     beamformerRxMask(:, a) = beamformerRxMaskPoint;
951
952     rxMask(:, :, a) = rxMaskPoint;
953     txMask(:, :, a) = txMaskPoint;
954
955     time = toc;
956     determine_eta(time, a, length(thetaScanRad));
957 end
958
959 %% Monte Carlo analysis
960
961
962 nVariations = 50;
963 magnitudeVariation = 0.5;
964 phaseVariation = 6;
965
966 %% Isolation
967 isoTxScanVar = zeros(nVariations, length(FREQ), length(thetaScanRad));
968 isoRxScanVar = zeros(nVariations, length(FREQ), length(thetaScanRad));
969 isoTxScan_RxTxVar = zeros(nVariations, length(FREQ), length(thetaScanRad));
970 isoRxScan_TxRxVar = zeros(nVariations, length(FREQ), length(thetaScanRad));
971 isoMGBScanVar = zeros(nVariations, length(FREQ), length(thetaScanRad));
972
973 for scanIndex = 1:length(thetaScanRad)
974     tic
975     parfor freqIndex = 1:length(FREQ)
976         isoTxScanVar(:, freqIndex, scanIndex) = monte_carlo_isolation( ...
977              'beamformerRx', beamformerRxCFMSys(:, scanIndex) ...
978             ,'beamformerTx', beamformerTxIso(:, scanIndex) ...
979             ,'freqIndex', freqIndex ...
980             ,'MW', MW ...
981             ,'portsLoadTx', portsLoadTx ...
982             ,'portsLoadRx', portsLoadRx ...
983             ,'Load', LoadTx ...
984             ,'transceiverSystem', transceiverSystem ...
985             ,'antenna', antenna ...
986             ,'loadArray', loadArray ...
987             ,'magnitudeVariation', magnitudeVariation ...
988             ,'phaseVariation', phaseVariation ...
989             ,'nVariations', nVariations ...
990            );
991
992         isoRxScanVar(:, freqIndex, scanIndex) = monte_carlo_isolation( ...
993              'beamformerRx', beamformerRxIso(:, scanIndex) ...
994             ,'beamformerTx', beamformerTxCFMSys(:, scanIndex) ...
995             ,'freqIndex', freqIndex ...
996             ,'MW', MW ...
997             ,'portsLoadTx', portsLoadTx ...
998             ,'portsLoadRx', portsLoadRx ...
999             ,'Load', LoadTx ...
1000            ,'transceiverSystem', transceiverSystem ...
1001            ,'antenna', antenna ...
1002            ,'loadArray', loadArray ...
1003            ,'magnitudeVariation', magnitudeVariation ...
1004            ,'phaseVariation', phaseVariation ...
1005            ,'nVariations', nVariations ...
1006           );
1007
1008        isoTxScan_RxTxVar(:, freqIndex, scanIndex) = ...
1009            monte_carlo_isolation( ...
1010             'beamformerRx', beamformerRxIso(:, scanIndex) ...
```

```
1011                ,'beamformerTx', beamformerTxIso_RxTx(:, scanIndex) ...
1012                ,'freqIndex', freqIndex ...
1013                ,'MW', MW ...
1014                ,'portsLoadTx', portsLoadTx ...
1015                ,'portsLoadRx', portsLoadRx ...
1016                ,'Load', LoadTx ...
1017                ,'transceiverSystem', transceiverSystem ...
1018                ,'antenna', antenna ...
1019                ,'loadArray', loadArray ...
1020                ,'magnitudeVariation', magnitudeVariation ...
1021                ,'phaseVariation', phaseVariation ...
1022                ,'nVariations', nVariations ...
1023               );
1024
1025           isoRxScan_TxRxVar(:, freqIndex, scanIndex) = ...
1026               monte_carlo_isolation( ...
1027                'beamformerRx', beamformerRxIso_TxRx(:, scanIndex) ...
1028                ,'beamformerTx', beamformerTxIso(:, scanIndex) ...
1029                ,'freqIndex', freqIndex ...
1030                ,'MW', MW ...
1031                ,'portsLoadTx', portsLoadTx ...
1032                ,'portsLoadRx', portsLoadRx ...
1033                ,'Load', LoadTx ...
1034                ,'transceiverSystem', transceiverSystem ...
1035                ,'antenna', antenna ...
1036                ,'loadArray', loadArray ...
1037                ,'magnitudeVariation', magnitudeVariation ...
1038                ,'phaseVariation', phaseVariation ...
1039                ,'nVariations', nVariations ...
1040               );
1041
1042           isoMGBScanVar(:, freqIndex, scanIndex) = monte_carlo_isolation( ...
1043                'beamformerRx', beamformerRxCFMSys(:, scanIndex) ...
1044                ,'beamformerTx', beamformerTxCFMSys(:, scanIndex) ...
1045                ,'freqIndex', freqIndex ...
1046                ,'MW', MW ...
1047                ,'portsLoadTx', portsLoadTx ...
1048                ,'portsLoadRx', portsLoadRx ...
1049                ,'Load', LoadTx ...
1050                ,'transceiverSystem', transceiverSystem ...
1051                ,'antenna', antenna ...
1052                ,'loadArray', loadArray ...
1053                ,'magnitudeVariation', magnitudeVariation ...
1054                ,'phaseVariation', phaseVariation ...
1055                ,'nVariations', nVariations ...
1056               );
1057       end
1058       runTime = toc;
1059       determine_eta(runTime, scanIndex, length(thetaScanRad));
1060   end
1061
1062   %% Mask Application
1063
1064   gainTxSysMaskVar = zeros(size(EEPs_SC, 1), ...
1065                            nVariations, ...
1066                            length(FREQ), ...
1067                            length(thetaScanRad));
1068   gainRxSysMaskVar = zeros(size(EEPs_SC, 1), ...
1069                            nVariations, ...
1070                            length(FREQ), ...
1071                            length(thetaScanRad));
1072
1073   for scanIndex = 1:length(thetaScanRad)
1074       tic
1075       parfor freqIndex = 1:length(FREQ)
1076           if any(FREQ(freqIndex) == F_TX_ARRAY)
1077               gainTxSysMaskVar(:, :, freqIndex, scanIndex) = ...
1078                   monte_carlo_gain_patterns( ...
1079                   'EEPs', EEPs_SC ...
1080                   ,'observationDistance', observationDistance ...
1081                   ,'Const', Const ...
1082                   ,'beamformer', beamformerTxMask(:, scanIndex) ...
1083                   ,'freqIndex', freqIndex ...
1084                   ,'ZoAnt', ZoAnt ...
```

```
                ,'ZoTxSubSys', ZoTxSubSys ...
                ,'inputPorts', portsTx ...
                ,'outputPorts', portsArrayElement ...
                ,'MW', MW ...
                ,'portsLoadTx', portsLoadTx ...
                ,'LoadTx', LoadTx ...
                ,'transceiverSystem', transceiverSystem ...
                ,'antenna', antenna ...
                ,'loadArray', loadArray ...
                ,'magnitudeVariation', magnitudeVariation ...
                ,'phaseVariation', phaseVariation ...
                ,'gainType', 'tx_sys' ...
                ,'nVariations', nVariations ...
                );
        elseif any(FREQ(freqIndex) == F_RX_ARRAY)
            gainRxSysMaskVar(:, :, freqIndex, scanIndex) = ...
                monte_carlo_gain_patterns( ...
                'EEPs', rxWaves ...
                ,'observationDistance', observationDistance ...
                ,'Const', Const ...
                ,'beamformer', beamformerRxMask(:, scanIndex) ...
                ,'freqIndex', freqIndex ...
                ,'lambda', lambda ...
                ,'yAnt', yAnt ...
                ,'zSysArray', zSysArray ...
                ,'magnitudeVariation', magnitudeVariation ...
                ,'phaseVariation', phaseVariation ...
                ,'gainType', 'rx_sys' ...
                ,'nVariations', nVariations ...
                );
        end

    end
    runTime = toc;
    determine_eta(runTime, scanIndex, length(thetaScanRad));
end

gainTxSysMaskPeakVar = ...
    zeros(nVariations, length(FREQ), length(thetaScanRad));
gainRxSysMaskPeakVar = ...
    zeros(nVariations, length(FREQ), length(thetaScanRad));

for scanIndex = 1:length(thetaScanRad)
    scanLogic = (phiMesh(:) == phiScanRad & ...
        thetaMesh(:) == thetaScanRad(scanIndex));
    for freqIndex = 1:length(FREQ)
        gainTxSysMaskPeakVar(:, freqIndex, scanIndex) = ...
            squeeze(gainTxSysMaskVar(scanLogic, :, freqIndex, scanIndex));

        gainRxSysMaskPeakVar(:, freqIndex, scanIndex) = ...
            squeeze(gainRxSysMaskVar(scanLogic, :, freqIndex, scanIndex));
    end
end

% It is important to note that this Main.m version does not contain the
% setup and code for the isoaltion and mask dual application, since the
% isoaltion and mask results are mostly overwritten. Thus a different
% version is created but the differences are trivial.
```

D.1.2 IsoBandwidthForGain.m

```
% Main.m must have run first!

beamformerRxRef = beamformerRxCFMSys;
beamformerTxRef = beamformerTxCFMSys;
resultsMW_ref = resultsMW_MGB;

accGainLoss = 0.5; % dB
isoStepTx   = 20; % dB
isoStepRx   = 20; % dB
convergenceDelta = 1e-3;
isoStepMin  = 0.5; % dB
maxIso = 100;
```

```
13
14 isoStepTx_RxTx    = isoStepTx; % dB
15 isoStepRx_TxRx    = isoStepRx; % dB
16
17 scanLogic = (phiMesh(:) == phiScanRad & thetaMesh(:) == thetaScanRad(a));
18
19 gainTxInit = gainTxCFMSys(scanLogic, :, a);
20 gainRxInit = gainRxCFMSys(scanLogic, :, a);
21
22 isoMGB = zeros(length(FREQ), 1);
23 for f = 1:length(FREQ)
24     rxInputRef = resultsMW_ref.a(portsLoadRx, 1, f);
25     iso = beamformerRxRef(:, a)' * (rxInputRef) * ...
26           (rxInputRef)' * beamformerRxRef(:, a);
27     isoMGB(f) = 10 * log10(abs(iso));
28 end
29 isoInit = isoMGB(FREQ(:) == F_RX);
30
31 targetIsoTx = -isoInit + isoStepTx / 2;
32 targetIsoRx = -isoInit + isoStepRx / 2;
33
34
35 %% Build SParams
36
37 sParamsWithAntenna = calculate_system_s_params( ...
38     'FREQ', FREQ ...
39    ,'deviceArray', [antenna, transceiverSystem] ...
40    ,'portConnectionTopology', [portsArrayElement; portsAnt].' ...
41    ,'externalPorts', [portsTx, portsRx]...
42     );
43
44 %% Looping the isolation calculations
45 loop = true;
46 while loop
47     %% Isolation Tx beamformer calculation
48
49     beamformerTxIso(:, a) = calculate_isolation_tx_beamformer( ...
50         'beamformerRxRef', beamformerRxRef(:, a), ...
51         'beamformerTxRef', beamformerTxRef(:, a), ...
52         'targetIso', targetIsoTx, ...
53         'sSystem', sParamsWithAntenna, ...
54         'sParamPortsRx', [nChannels+1:2*nChannels], ...
55         'sParamPortsTx', [1:nChannels], ...
56         'FREQ', FREQ, ...
57         'F_RX_ARRAY', F_RX_ARRAY, ...
58         'MW', MW, ...
59         'portsLoadTx', portsLoadTx, ...
60         'portsLoadRx', portsLoadRx, ...
61         'LoadTx', LoadTx, ...
62         'transceiverSystem', transceiverSystem, ...
63         'antenna', antenna, ...
64         'loadArray', loadArray, ...
65         'convergenceDelta', convergenceDelta ...
66         );
67
68     %% Microwave System with isolation beamformer to scale excitation
69      % vectors
70
71     [MW, loadMGBArray] = define_load_source_ports( ...
72         'MW', MW, ...
73         'sourcePortArray', portsLoadTx, ...
74         'excitationArray', beamformerTxIso(:, a), ...
75         'Load', LoadTx ...
76         );
77
78
79     % === START of MW-Engine ===
80     resultsMW_iso = RunMWSimulator(...
81         [transceiverSystem, ...
82             antenna, ...
83             loadMGBArray, ...
84             loadArray(nChannels+1:2*nChannels)], ...
85         MW);
86     % === End of MW-Engine ===
```

```
87
88      %% Isolation Rx beamformer calculation
89      beamformerRxIso(:, a) = calculate_isolation_rx_beamformer( ...
90          'beamformerRxRef', beamformerRxRef(:, a), ...
91          'targetIso', targetIsoRx, ...
92          'resultsMW_MGB', resultsMW_ref, ...
93          'portsLoadRx', portsLoadRx, ...
94          'FREQ', FREQ, ...
95          'F_RX_ARRAY', F_RX_ARRAY, ...
96          'convergenceDelta', convergenceDelta ...
97          );
98
99      %% Calculate Tx gain and Tx-Rx isolation
100
101     lambda = 3e8 ./ FREQ;
102     gainTxIso = zeros(size(EEPs_SC, 1), length(FREQ));
103     gainRxIso = zeros(size(EEPs_SC, 1), length(FREQ));
104     isoTxBF = zeros(length(FREQ), 1);
105     isoRxBF = zeros(length(FREQ), 1);
106     parfor f = 1:length(FREQ)
107         if any(FREQ(f) == F_TX_ARRAY)
108
109             gainTxIso(:, f) = calculate_gain_tx( ...
110                 'EEPs', EEPs_SC, ...
111                 'observationDistance', observationDistance, ...
112                 'Const', Const, ...
113                 'beamformer', beamformerTxIso(:, a), ...
114                 'resultsMW', resultsMW_iso, ...
115                 'freqIndex', f, ...
116                 'ZoAnt', ZoAnt, ...
117                 'ZoTxSubSys', ZoTxSubSys, ...
118                 'inputPorts', portsTx, ...
119                 'outputPorts', portsArrayElement, ...
120                 'gainType', 'radiation' ...
121                 );
122
123         end
124
125         if any(FREQ(f) == F_RX_ARRAY)
126
127             gainRxIso(:, f) = calculate_system_gain_rx( ...
128                 'rxWaves', rxWaves, ...
129                 'beamformer', beamformerRxIso(:, a), ...
130                 'lambda', lambda, ...
131                 'freqIndex', f, ...
132                 'yAnt', yAnt, ...
133                 'zSys', zSysArray  ...
134                 );
135
136         end
137
138         rxInputTxIso = resultsMW_iso.a(portsLoadRx, 1, f);
139         iso = beamformerRxRef(:, a)' * (rxInputTxIso) * ...
140             rxInputTxIso' * beamformerRxRef(:, a);
141         isoTxBF(f) = 10 * log10(abs(iso));
142
143         rxInputRef = resultsMW_ref.a(portsLoadRx, 1, f);
144         iso = beamformerRxIso(:, a)' * (rxInputRef) * ...
145             rxInputRef' * beamformerRxIso(:, a);
146         isoRxBF(f) = 10 * log10(abs(iso));
147
148     end
149
150     %% Checking conditions for isolation and gain requirements
151
152     gainTxIsoPeak = gainTxIso(scanLogic, :);
153     gainRxIsoPeak = gainRxIso(scanLogic, :);
154
155     % Tx check
156     if (gainTxIsoPeak - gainTxInit) > -accGainLoss
157         isoStepTx = abs(isoStepTx);
158     else
159         isoStepTx = abs(isoStepTx) / -2;
160     end
```

```
161     targetIsoTx = targetIsoTx + isoStepTx;
162
163     % Rx check
164     if (gainRxIsoPeak - gainRxInit) > -accGainLoss
165         isoStepRx = abs(isoStepRx);
166     else
167         isoStepRx = abs(isoStepRx) / -2;
168     end
169     targetIsoRx = targetIsoRx + isoStepRx;
170
171     % Convergence check
172     if (abs(isoStepTx) < isoStepMin) ...
173             && (abs(isoStepRx) < isoStepMin)
174         loop = false;
175     end
176
177     if (abs(targetIsoRx) > maxIso) ...
178             || (abs(targetIsoTx) > maxIso)
179         loop = false;
180     end
181
182 end
183
184 %% Looping the combination beamformers
185
186 targetIsoRx_TxRx = targetIsoTx + isoStepRx_TxRx/2;
187 targetIsoTx_RxTx = targetIsoRx + isoStepTx_RxTx/2;
188
189 loop = true;
190 while loop
191     %% Tx then Rx
192     beamformerRxIso_TxRx(:, a) = calculate_isolation_rx_beamformer( ...
193         'beamformerRxRef', beamformerRxRef(:, a), ...
194         'targetIso', targetIsoRx_TxRx, ...
195         'resultsMW_MGB', resultsMW_iso, ...
196         'portsLoadRx', portsLoadRx, ...
197         'FREQ', FREQ, ...
198         'F_RX_ARRAY', F_RX_ARRAY, ...
199         'convergenceDelta', convergenceDelta ...
200         );
201
202     %% Rx then Tx
203     beamformerTxIso_RxTx(:, a) = calculate_isolation_tx_beamformer( ...
204         'beamformerRxRef', beamformerRxIso(:, a), ...
205         'beamformerTxRef', beamformerTxRef(:, a), ...
206         'targetIso', targetIsoTx_RxTx, ...
207         'sSystem', sParamsWithAntenna, ...
208         'sParamPortsRx', [nChannels+1:2*nChannels], ...
209         'sParamPortsTx', [1:nChannels], ...
210         'FREQ', FREQ, ...
211         'F_RX_ARRAY', F_RX_ARRAY, ...
212         'MW', MW, ...
213         'portsLoadTx', portsLoadTx, ...
214         'portsLoadRx', portsLoadRx, ...
215         'LoadTx', LoadTx, ...
216         'transceiverSystem', transceiverSystem, ...
217         'antenna', antenna, ...
218         'loadArray', loadArray, ...
219         'convergenceDelta', convergenceDelta ...
220         );
221
222     [MW, loadMGBArray] = define_load_source_ports( ...
223         'MW', MW, ...
224         'sourcePortArray', portsLoadTx, ...
225         'excitationArray', beamformerTxIso_RxTx(:, a), ...
226         'Load', LoadTx ...
227         );
228
229
230     % === START of MW-Engine ===
231     resultsMW_iso_RxTx = RunMWSimulator(...
232         [transceiverSystem, antenna, loadMGBArray, ...
233         loadArray(nChannels+1:2*nChannels)], MW);
234     % === End of MW-Engine ===
```

```
235
236
237     %% Calculate Tx gain and Tx-Rx isolation
238
239     lambda = 3e8 ./ FREQ;
240     gainTxIso_RxTx = zeros(size(EEPs_SC, 1), length(FREQ));
241     gainRxIso_TxRx = zeros(size(EEPs_SC, 1), length(FREQ));
242     isoTxBF_RxTx = zeros(length(FREQ), 1);
243     isoRxBF_TxRx = zeros(length(FREQ), 1);
244     parfor f = 1:length(FREQ)
245         if any(FREQ(f) == F_TX_ARRAY)
246
247             gainTxIso_RxTx(:, f) = calculate_gain_tx( ...
248                 'EEPs', EEPs_SC, ...
249                 'observationDistance', observationDistance, ...
250                 'Const', Const, ...
251                 'beamformer', beamformerTxIso_RxTx(:, a), ...
252                 'resultsMW', resultsMW_iso_RxTx, ...
253                 'freqIndex', f, ...
254                 'ZoAnt', ZoAnt, ...
255                 'ZoTxSubSys', ZoTxSubSys, ...
256                 'inputPorts', portsTx, ...
257                 'outputPorts', portsArrayElement, ...
258                 'gainType', 'radiation' ...
259                 );
260         end
261
262         if any(FREQ(f) == F_RX_ARRAY)
263
264             gainRxIso_TxRx(:, f) = calculate_system_gain_rx( ...
265                 'rxWaves', rxWaves, ...
266                 'beamformer', beamformerRxIso_TxRx(:, a), ...
267                 'lambda', lambda, ...
268                 'freqIndex', f, ...
269                 'yAnt', yAnt, ...
270                 'zSys', zSysArray  ...
271                 );
272         end
273
274         rxInputIsoRxTx = resultsMW_iso_RxTx.a(portsLoadRx, 1, f);
275         iso = beamformerRxIso(:, a)' * (rxInputIsoRxTx) * ...
276             rxInputIsoRxTx' * beamformerRxIso(:, a);
277         isoTxBF_RxTx(f) = 10 * log10(abs(iso));
278
279         rxInputTxIso = resultsMW_iso.a(portsLoadRx, 1, f);
280         iso = beamformerRxIso_TxRx(:, a)' * (rxInputTxIso) * ...
281             rxInputTxIso' * beamformerRxIso_TxRx(:, a);
282         isoRxBF_TxRx(f) = 10 * log10(abs(iso));
283
284     end
285
286     %% Checking conditions for isolation and gain requirements
287
288     gainTxIsoPeakRxTx = gainTxIso_RxTx(scanLogic, :);
289     gainRxIsoPeakTxRx = gainRxIso_TxRx(scanLogic, :);
290
291     % Tx check - RxTx
292     if (gainTxIsoPeakRxTx - gainTxInit) > -accGainLoss
293         isoStepTx_RxTx = abs(isoStepTx_RxTx);
294     else
295         isoStepTx_RxTx = abs(isoStepTx_RxTx) / -2;
296     end
297     targetIsoTx_RxTx = targetIsoTx_RxTx + isoStepTx_RxTx;
298
299     % Rx check - TxRx
300     if (gainRxIsoPeakTxRx - gainRxInit) > -accGainLoss
301         isoStepRx_TxRx = abs(isoStepRx_TxRx);
302     else
303         isoStepRx_TxRx = abs(isoStepRx_TxRx) / -2;
304     end
305     targetIsoRx_TxRx = targetIsoRx_TxRx + isoStepRx_TxRx;
306
307     % Convergence check
308     if (abs(isoStepTx_RxTx) < isoStepMin) ...
```

```
309             && (abs(isoStepRx_TxRx) < isoStepMin)
310         loop = false;
311     end
312
313     if (abs(targetIsoTx_RxTx) > maxIso) ...
314             && (abs(targetIsoRx_TxRx) > maxIso)
315         loop = false;
316     end
317
318     if (abs(targetIsoTx_RxTx) > maxIso)
319             targetIsoTx_RxTx = maxIso;
320     end
321
322     if (abs(targetIsoRx_TxRx) > maxIso)
323             targetIsoRx_TxRx = maxIso;
324     end
325
326 end
```

D.1.3 MaskApplication.m

```
1  % Main.m must have run first!
2  beamformerRxRef = beamformerRxCFMSys;
3  beamformerTxRef = beamformerTxCFMSys;
4
5  scanIndex = a;
6  % todo: combine the mask and isolation application
7  convDelta  = 1e-2;
8  compOffset = 0;
9
10 % set rx mask constraints
11 maxMaskGainRx      = 20;
12 maxSllRx           = -18;
13 mainBwRx           = 9;
14 bottomBwRx         = 12;
15
16 % set tx mask constraints
17 maxMaskGainTx      = 18;
18 maxSllTx           = -20;
19 mainBwTx           = 8;
20 bottomBwTx         = 11;
21
22 %% Mask Application
23
24 % create mask array
25 rxMaskPoint = zeros(length(phiArray), length(thetaArray));
26 for j = 1:length(phiArray)
27     for t = 1:length(thetaArray)
28         rxMaskPoint(j, t) = rx_mask_etsien303979_adapted( ...
29             thetaArrayDeg(t), ...
30             phiArrayDeg(j), ...
31             thetaScanDeg(scanIndex), ...
32             phiScanDeg);
33     end
34 end
35
36 rxMaskPoint = rxMaskPoint + gainRxPathdBRxFreq;
37
38 txMaskPoint = zeros(length(phiArray), length(thetaArray));
39 for j = 1:length(phiArray)
40     for t = 1:length(thetaArray)
41         txMaskPoint(j, t) = tx_mask_FCC_47CFR_25_218_adapted( ...
42             thetaArrayDeg(t), ...
43             phiArrayDeg(j), ...
44             thetaScanDeg(scanIndex), ...
45             phiScanDeg, ...
46             'co');
47     end
48 end
49
50 beamformerRxMaskPoint = apply_mask( ...
51     'beamformerOpt', beamformerRxRef(:, scanIndex), ...
52     'gainMask', rxMaskPoint, ...
```

```
    'EEPs', rxWaves, ...
    'FREQ', FREQ, ...
    'F_ARRAY', F_RX_ARRAY, ...
    'gainOpt', gainRxCFMSys(:, :, scanIndex), ...
    'polarisation', 'x', ...
    'beamformerConvergenceDelta', convDelta, ...
    'compensatingOffset', compOffset, ...
    'lambda', lambda, ...
    'yAnt', yAnt, ...
    'zSys', zSysArray, ... zSysArrayReceive, ...
    'direction', 'Rx' ...
    );

beamformerTxMaskPoint = apply_mask( ...
    'beamformerOpt', beamformerTxRef(:, scanIndex), ...
    'gainMask', txMaskPoint, ...
    'EEPs', EEPs_SC, ...
    'FREQ', FREQ, ...
    'F_ARRAY', F_TX_ARRAY, ...
    'gainOpt', gainTxCFMSys(:, :, scanIndex), ...
    'polarisation', 'x', ...
    'beamformerConvergenceDelta', convDelta, ...
    'compensatingOffset', compOffset, ...
    'observationDistance', observationDistance, ...
    'Const', Const, ...
    'ZoAnt', ZoAnt, ...
    'ZoTxSubSys', ZoAnt, ...
    'inputPorts', portsTx, ...
    'outputPorts', portsArrayElement, ...
    'gainType', 'radiation', ...
    'MW', MW, ...
    'sourcePortArray', portsLoadTx, ... portsLoadSignal, ...
    'LoadTx', LoadTx, ...
    'antenna', antenna, ...
    'MWSystem', transceiverSystem, ... TxSystem, ...
    'loadArray', loadArray, ...
    'direction', 'Tx' ...
    );

%% Microwave System with MGBs to scale excitation vectors

[MW, loadMGBArray] = define_load_source_ports( ...
    'MW', MW, ...
    'sourcePortArray', portsLoadTx, ...portsLoadSignal, ...
    'excitationArray', beamformerTxMaskPoint, ...
    'Load', LoadTx ...
    );

% For use in Main.m
% === START of MW-Engine ===
resultsMW_TxMask = RunMWSimulator(...
    [transceiverSystem, ...
     antenna, ...
     loadMGBArray, ...
     loadArray(nChannels+1:2*nChannels)], ...
    MW);
% === End of MW-Engine ===

% For use in ReciprocalSystem.m
% % === START of MW-Engine ===
% resultsMW_TxMask = RunMWSimulator([antenna, TxSystem, loadMGBArray], MW);
% % === End of MW-Engine ===

%% Calculate Tx and Rx gain with mask applied

lambda = 3e8 ./ FREQ;
gainTxMaskPoint = zeros(size(EEPs_SC, 1), length(FREQ));
gainRxMaskPoint = zeros(size(EEPs_SC, 1), length(FREQ));
for i = 1:length(FREQ)
    if any(FREQ(i) == F_TX_ARRAY)
        gainTxMaskPoint(:, i) = calculate_gain_tx( ...
            'EEPs', EEPs_SC, ...
            'observationDistance', observationDistance, ...
            'Const', Const, ...
```

```
127             'beamformer', beamformerTxMaskPoint, ...
128             'resultsMW', resultsMW_TxMask, ...
129             'freqIndex', i, ...
130             'ZoAnt', ZoAnt, ...
131             'ZoTxSubSys', ZoTxSubSys, ...
132             'inputPorts', portsTx, ...
133             'outputPorts', portsArrayElement, ...
134             'gainType', 'radiation' ...
135             );
136     end
137
138     if any(FREQ(i) == F_RX_ARRAY)
139
140         gainRxMaskPoint(:, i) = calculate_system_gain_rx( ...
141             'rxWaves', rxWaves, ...
142             'beamformer', beamformerRxMaskPoint, ...
143             'lambda', lambda, ....
144             'freqIndex', i, ...
145             'yAnt', yAnt, ...
146             'zSys', zSysArray ... zSysArrayReceive  ...
147             );
148     end
149
150 end
```

D.1.4 MaskIsoDual.m

```
1  % Main.m must have run first!
2
3  convDelta  = 1e-3;
4  compOffset = 0;
5
6  % set rx mask constraints
7  maxMaskGainRx      = 10;
8  maxSllRx           = -15;
9  mainBwRx           = 30;
10 bottomBwRx         = 35;
11
12 % set tx mask constraints
13 maxMaskGainTx      = 10;
14 maxSllTx           = -18;
15 mainBwTx           = 25;
16 bottomBwTx         = 30;
17
18 % isolation constraints
19 targetIso = 100;
20
21 %% Build SParams
22
23 sParamsWithAntenna = calculate_system_s_params( ...
24     'FREQ', FREQ ...
25    ,'deviceArray', [antenna, transceiverSystem] ...
26    ,'portConnectionTopology', [portsArrayElement; portsAnt].' ...
27    ,'externalPorts', [portsTx, portsRx]...
28     );
29
30 %% Create Masks
31
32 % create mask array
33 rxMask = zeros(length(phiArray), length(thetaArray));
34 for j = 1:length(phiArray)
35     for t = 1:length(thetaArray)
36         rxMask(j, t) = rx_mask_generic( ...
37             thetaArray(t)*180/pi, ...
38             phiArray(j)*180/pi, ...
39             thetaScanDeg, ...
40             phiScanDeg, ...
41             maxMaskGainRx, ...
42             maxSllRx, ...
43             mainBwRx, ...
44             bottomBwRx);
45     end
46 end
```

```
47
48 rxMask = rxMask + gainRxPathdB;
49
50 txMask = zeros(length(phiArray), length(thetaArray));
51 for j = 1:length(phiArray)
52     for t = 1:length(thetaArray)
53         txMask(j, t) = rx_mask_generic( ...
54             thetaArray(t)*180/pi, ...
55             phiArray(j)*180/pi, ...
56             thetaScanDeg, ...
57             phiScanDeg, ...
58             maxMaskGainTx, ...
59             maxSllTx, ...
60             mainBwTx, ...
61             bottomBwTx);
62     end
63 end
64
65 txMask = txMask + gainTxPathdB;
66
67 %% end
68
69 beamformerRxMaskIso = calculate_mask_isolation_rx_beamformer( ...
70     'beamformerRxRef', beamformerRxMaxSens ...
71    ,'beamformerTxRef', beamformerTxLoadedAWave ...
72    ,'targetIso', targetIso ...
73    ,'FREQ', FREQ ...
74    ,'F_RX_ARRAY', F_RX_ARRAY ...
75    ,'F_ARRAY', F_RX_ARRAY ...
76    ,'MW', MW ...
77    ,'portsLoadTx', portsLoadTx ...
78    ,'portsLoadRx', portsLoadRx ...
79    ,'Load', Load ...
80    ,'transceiverSystem', transceiverSystem ...
81    ,'antenna', antenna ...
82    ,'loadArray', loadArray ...
83    ,'convergenceDelta', convDelta ...
84    ,'gainMask', rxMask ...
85    ,'EEPs', rxWaves ...
86    ,'gainOpt', gainRxSys ...
87    ,'polarisation', 'x' ...
88    ,'compensatingOffset', compOffset ...
89    ,'lambda', lambda ...
90    ,'yAnt', yAnt ...
91    ,'zSys', zSysArray ...
92     );
93
94 beamformerTxMaskIso = calculate_mask_isolation_tx_beamformer( ...
95     'beamformerRxRef', beamformerRxMaxSens ...
96    ,'beamformerTxRef', beamformerTxLoadedAWave ...
97    ,'targetIso', targetIso ...
98    ,'sSystem', sParamsWithAntenna ...
99    ,'sParamPortsRx', [nChannels+1:2*nChannels] ...
100   ,'sParamPortsTx', [1:nChannels] ...
101   ,'FREQ', FREQ ...
102   ,'F_RX_ARRAY', F_RX_ARRAY ...
103   ,'F_ARRAY', F_TX_ARRAY ...
104   ,'MW', MW ...
105   ,'portsLoadTx', portsLoadTx ...
106   ,'portsLoadRx', portsLoadRx ...
107   ,'Load', Load ...
108   ,'transceiverSystem', transceiverSystem ...
109   ,'antenna', antenna ...
110   ,'loadArray', loadArray ...
111   ,'convergenceDelta', convDelta ...
112   ,'gainMask', txMask ...
113   ,'EEPs', EEPs_SC ...
114   ,'gainOpt', gainTxSys ...
115   ,'polarisation', 'x' ...
116   ,'compensatingOffset', compOffset ...
117   ,'observationDistance', observationDistance ...
118   ,'Const', Const ...
119   ,'Zo', Zo ...
120   ,'inputPorts', portsTx ...
```

```
    ,'outputPorts', portsArrayElement ...
    ,'gainType', 'radiation' ...
     );

%% Microwave System with MGBs to scale excitation vectors

[MW, loadMGBArray] = define_load_source_ports( ...
     'MW', MW ...
    ,'sourcePortArray', portsLoadTx ...portsLoadSignal ...
    ,'excitationArray', beamformerTxMaskIso ...
    ,'Load', Load ...
     );

% For use in Main.m
% === START of MW-Engine ===
resultsMW_TxMaskIso = RunMWSimulator(...
     [transceiverSystem, antenna, loadMGBArray, ...
     loadArray(nChannels+1:2*nChannels)], MW);
% === End of MW-Engine ===

%% Calculate Tx and Rx gain with mask applied

lambda = 3e8 ./ FREQ;
gainTxIso = zeros(size(EEPs_SC, 1), length(FREQ));
gainRxIso = zeros(size(EEPs_SC, 1), length(FREQ));
isoTxBF = zeros(length(FREQ), 1);
isoRxBF = zeros(length(FREQ), 1);
isoMGB = zeros(length(FREQ), 1);
for i = 1:length(FREQ)
    if any(FREQ(i) == F_TX_ARRAY)

        gainTxIso(:, i) = calculate_gain_tx( ...
             'EEPs', EEPs_SC ...
            ,'observationDistance', observationDistance ...
            ,'Const', Const ...
            ,'beamformer', beamformerTxMaskIso ...
            ,'resultsMW', resultsMW_TxMaskIso ...
            ,'freqIndex', i ...
            ,'Zo', Zo ...
            ,'inputPorts', portsTx ...
            ,'outputPorts', portsArrayElement ...
            ,'gainType', 'radiation' ...
             );
    end

    if any(FREQ(i) == F_RX_ARRAY)

        gainRxIso(:, i) = calculate_system_gain_rx( ...
             'rxWaves', rxWaves ...
            ,'beamformer', beamformerRxMaskIso ...
            ,'lambda', lambda ...
            ,'freqIndex', i ...
            ,'yAnt', yAnt ...
            ,'zSys', zSysArray  ...
             );
    end


    iso = resultsMW_TxMaskIso.a(portsLoadRx, 1, i).' * ...
        conj(beamformerRxMaxSens);
    isoTxBF(i) = 10 * log10(abs(iso));

    iso = resultsMW_MGB.a(portsLoadRx, 1, i).' * conj(beamformerRxMaxSens);
    isoMGB(i) = 10 * log10(abs(iso));

    iso = resultsMW_MGB.a(portsLoadRx, 1, i).' * conj(beamformerRxMaskIso);
    isoRxBF(i) = 10 * log10(abs(iso));

end

gainTxMask = gainTxIso;
gainRxMask = gainRxIso;
```

D.2 functions

D.2.1 apply_mask.m

```
function beamformerMask = apply_mask(varargin)
% Applies a specified gain mask to an embedded array
%
% Applies a specified gain mask given an optimal beamformer weighting
% vector and embedded element patters. A maximum number of iterations or
% accuracy can be specified to prevent infinite loops or limits runtime.
%
% Args:
% beamformerOpt
% gainMask
% EEPs
% maxGainScalar
% polarisation
% beamfomerConvergenceDelta
% compensatingOffset

%% input parsing
p = inputParser();
p.CaseSensitive = true;

p.addOptional('beamformerOpt', []);
p.addOptional('gainMask', []);
p.addOptional('EEPs', []);
p.addOptional('FREQ', []);
p.addOptional('F_ARRAY', []);
p.addOptional('gainOpt', []);
p.addOptional('polarisation', 'x');
p.addOptional('beamformerConvergenceDelta', 1e-5);
p.addOptional('compensatingOffset', 0);
p.addOptional('observationDistance', []);
p.addOptional('Const', []);
p.addOptional('ZoAnt', []);
p.addOptional('ZoTxSubSys', []);
p.addOptional('inputPorts', []);
p.addOptional('outputPorts', []);
p.addOptional('gainType', 'radiation');
p.addOptional('MW', []);
p.addOptional('sourcePortArray', []);
p.addOptional('LoadTx', []);
p.addOptional('antenna', []);
p.addOptional('MWSystem', []);
p.addOptional('loadArray', []);
p.addOptional('lambda', []);
p.addOptional('yAnt', []);
p.addOptional('zSys', []);
p.addOptional('direction', []);

p.parse(varargin{:});
beamformerOpt                = p.Results.beamformerOpt;
gainMask                     = p.Results.gainMask;
EEPs                         = p.Results.EEPs;
FREQ                         = p.Results.FREQ;
F_ARRAY                      = p.Results.F_ARRAY;
gainOpt                      = p.Results.gainOpt;
polarisation                 = p.Results.polarisation;
beamformerConvergenceDelta   = p.Results.beamformerConvergenceDelta;
compensatingOffset           = p.Results.compensatingOffset;
observationDistance          = p.Results.observationDistance;
Const                        = p.Results.Const;
ZoAnt                        = p.Results.ZoAnt;
ZoTxSubSys                   = p.Results.ZoTxSubSys;
inputPorts                   = p.Results.inputPorts;
outputPorts                  = p.Results.outputPorts;
gainType                     = p.Results.gainType;
MW                           = p.Results.MW;
sourcePortArray              = p.Results.sourcePortArray;
LoadTx                       = p.Results.LoadTx;
antenna                      = p.Results.antenna;
MWSystem                     = p.Results.MWSystem;
```

```
loadArray                   = p.Results.loadArray;
lambda                      = p.Results.lambda;
yAnt                        = p.Results.yAnt;
zSys                        = p.Results.zSys;
direction                   = p.Results.direction;

%% create E-field vectors and maximum E-field

if strcmp(direction, 'Rx')
    beamformerOpt = conj(beamformerOpt);
end

freqIndex = arrayfun(@(x) find(FREQ == x, 1) , F_ARRAY);
freqPos = zeros(length(FREQ), 1);
for i = 1:length(freqIndex)
    freqPos(freqIndex(i)) = 1;
end
freqPos = logical(freqPos);


E_p = EEPs(:, :, :, freqPos);
if strcmp(polarisation, 'x')
    E_p = squeeze(E_p(:, 1, :, :));
elseif strcmp(polarisation, 'y')
    E_p = squeeze(E_p(:, 2, :, :));
elseif strcmp(polarisation, 'rhc')
    E_x = squeeze(E_p(:, 1, :, :));
    E_y = squeeze(E_p(:, 2, :, :));
    E_p = 1 / sqrt(2) * (E_x + 1j * E_y);
elseif strcmp(polarisation, 'lhc')
    E_x = squeeze(E_p(:, 1, :, :));
    E_y = squeeze(E_p(:, 2, :, :));
    E_p = 1 / sqrt(2) * (E_x - 1j * E_y);
end

nFreq = length(F_ARRAY);
nChannels = length(beamformerOpt);

gainOpt = gainOpt(:, freqIndex);

gainMask = reshape(gainMask, [], 1);
gainMask = gainMask - compensatingOffset;

if nFreq == 1
    E_p = squeeze(E_p);
    E_pMaxGain          = E_p * beamformerOpt;
    E_pMaxGainScalar    = max(abs(E_pMaxGain));
    E_p                 = E_p ./ E_pMaxGainScalar;
else
    for f = 1:nFreq
        E_pMaxGain          = squeeze(E_p(:, :, f)) * beamformerOpt;
        E_pMaxGainScalar    = max(abs(E_pMaxGain));
        E_p(:, :, f)        = E_p(:, :, f) ./ E_pMaxGainScalar;
    end

    E_p = permute(E_p, [1, 3, 2]);
    E_p = reshape(E_p, [], size(E_p, 3));

    oldGainMask = gainMask;
    for i = 2:nFreq
        gainMask(:, i) = oldGainMask;
    end
    gainMask = reshape(gainMask, [], 1);
end


maxGainScalar = zeros(size(EEPs, 1), nFreq);
for f = 1:nFreq
    maxGainScalar(:, f) = ones(size(EEPs, 1), 1) * max(gainOpt(:, f));
end

maxGainScalar = reshape(maxGainScalar, [], 1);

E_pNorm = E_p;
```

```
144
145 %% determine the E-field mask in the cutting plane
146 E_mask    = 10 .^ ((gainMask - maxGainScalar) / 20);
147 for i = 1:size(E_mask, 1)
148     if isinf(E_mask(i))
149         E_mask(i) = 1;
150     end
151 end
152
153 %% state the convex problem for determining the weighting vector to fit the
154 %% E-field mask
155 beamformerCompare   = beamformerOpt;
156 beamformerPrevious  = beamformerCompare;
157 loop                = true;
158
159 n = size(beamformerOpt, 1);
160 while loop
    ↪                                                                        %
    ↪ loop while normalising beamformerMask to converge closer to the mask
161     cvx_begin quiet
162         variable beamformerMask(n) complex
163         minimize( norm(beamformerMask - beamformerOpt, 2) )
164         subject to
165             abs(E_pNorm * beamformerMask) <= E_mask;
166     cvx_end
167
168     beamformerCompare = beamformerMask / norm(beamformerMask, 2);
169     deltaV = abs(beamformerCompare - beamformerPrevious) ...
170              ./ abs(beamformerPrevious);
171
172     if strcmp(direction, 'Tx')
173
174         [MW, loadMGBArray] = define_load_source_ports( ...
175             'MW', MW, ...
176             'sourcePortArray', sourcePortArray, ...
177             'excitationArray', beamformerCompare, ...
178             'Load', LoadTx ...
179             );
180
181         % For Main.m
182         % === START of MW-Engine ===
183         resultsMW_TxMask = RunMWSimulator([MWSystem, antenna, loadMGBArray,
                ↪  loadArray(nChannels+1:2*nChannels)], MW);
184         % === End of MW-Engine ===
185
186         gainTxOvershoot = zeros(size(EEPs, 1), nFreq);
187         maskOvershoot = zeros(size(EEPs, 1), nFreq);
188         for i = 1:nFreq
189             gainTxOvershoot(:, i) = calculate_gain_tx( ...
190                     'EEPs', EEPs, ...
191                     'observationDistance', observationDistance, ...
192                     'Const', Const, ...
193                     'beamformer', beamformerCompare, ...
194                     'resultsMW', resultsMW_TxMask, ...
195                     'freqIndex', find(FREQ(:) == F_ARRAY(i)), ...
196                     'ZoAnt', ZoAnt, ...
197                     'ZoTxSubSys', ZoTxSubSys, ...
198                     'inputPorts', inputPorts, ...
199                     'outputPorts', outputPorts, ...
200                     'gainType', gainType ...
201                     );
202
203             gainMaskReshape = reshape(gainMask, [], nFreq);
204             for p = 1:length(gainTxOvershoot(:, i))
205                 maskOvershoot(p, i) = gainTxOvershoot(p, i) -
                        ↪ gainMaskReshape(p, i);
206                 if maskOvershoot(p, i) < 0
207                     maskOvershoot(p, i) = 0;
208                 end
209             end
210         end
211         maskOvershoot = reshape(maskOvershoot, [], 1);
212     elseif strcmp(direction, 'Rx')
213
```

```
214         gainRxUpdate = zeros(size(EEPs, 1), nFreq);
215         maskOvershoot = zeros(size(EEPs, 1), nFreq);
216         for i = 1:nFreq
217             gainRxUpdate(:, i) = calculate_system_gain_rx( ...
218                 'rxWaves', EEPs, ...
219                 'beamformer', conj(beamformerCompare), ...
220                 'lambda', lambda, ....
221                 'freqIndex', find(FREQ(:) == F_ARRAY(i)), ...
222                 'yAnt', yAnt, ...
223                 'zSys', zSys  ...
224                 );
225 
226             gainMaskReshape = reshape(gainMask, [], nFreq);
227             for p = 1:length(gainRxUpdate(:, i))
228                 maskOvershoot(p, i) = gainRxUpdate(p, i) - gainMaskReshape(
                        ↪ p, i);
229                 if maskOvershoot(p, i) < 0
230                     maskOvershoot(p, i) = 0;
231                 end
232             end
233         end
234         maskOvershoot = reshape(maskOvershoot, [], 1);
235     end
236 
237     if maskOvershoot <= 0
238         loop = false;
239     elseif deltaV < beamformerConvergenceDelta
240         loop = false;
241     else
242         beamformerPrevious = beamformerCompare;
243         maskOvershoot = 10 .^ (maskOvershoot / 20);
244         E_mask = E_mask ./ maskOvershoot;
245     end
246 end
247 
248 if strcmp(direction, 'Rx')
249     beamformerMask = conj(beamformerCompare);
250 else
251     beamformerMask = beamformerCompare;
252 end
253 
254 end
```

D.2.2 calculate_array_gain.m

```
1  function arrayGain = calculate_array_gain(varargin)
2  % Calculates the array gain given the short circuit embeddded element
3  % patterns and a beamformed input
4  %
5  % Args:
6  % EEPs_SC
7  % observationDistance
8  % Const
9  % beamformer
10 % yAnt
11 % freqIndex
12 
13 %% input parsing
14     p = inputParser();
15     p.CaseSensitive = false;
16 
17     p.addOptional('EEPs_SC',[]);
18     p.addOptional('observationDistance',[]);
19     p.addOptional('Const',[]);
20     p.addOptional('beamformer',[]);
21     p.addOptional('yAnt',[]);
22     p.addOptional('freqIndex',[]);
23 
24     p.parse(varargin{:});
25     EEPs_SC             = p.Results.EEPs_SC;
26     observationDistance = p.Results.observationDistance;
27     Const               = p.Results.Const;
28     beamformer          = p.Results.beamformer;
```

```
29      yAnt                = p.Results.yAnt;
30      freqIndex           = p.Results.freqIndex;
31
32      EEPs_SC = EEPs_SC(:, :, :, freqIndex);
33      Const   = Const(:, freqIndex);
34      yAnt    = yAnt(:, :, freqIndex);
35
36  %% Gain calculation
37      Etot = zeros(size(EEPs_SC,1),3);
38      for n = 1 : size(yAnt,2)
39          Etot = Etot + EEPs_SC(:,:,n)*beamformer(n);
40      end
41      farField        = Etot * observationDistance * exp(1j * Const.k * ...
42                          observationDistance);
                                ↪                                   % Far field
                                ↪ function
43      U               = (abs(farField(:,1)).^2 + abs(farField(:,2)).^2 +...
44                          abs(farField(:,3)).^2) / (2 * 120 * pi);
45      Pin             = real(1 / 2 * beamformer' * real(yAnt') * beamformer);
46      Uiso            = Pin / (4 * pi);
47      arrayGain       = 10 * log10(U / Uiso);
48  end
```

D.2.3 calculate_cfm_beamformer_weights.m

```
1   function beamformerCFM = calculate_cfm_beamformer_weights(varargin)
2
3   %% input parsing
4
5   p = inputParser();
6   p.CaseSensitive = true;
7   p.addOptional('planeWaveExcitations', []);
8   p.addOptional('thetaMesh', []);
9   p.addOptional('phiMesh', []);
10  p.addOptional('thetaScanRad', []);
11  p.addOptional('phiScanRad', []);
12  p.addOptional('freqIndex', []);
13  p.addOptional('polarisation', '')
14
15  p.parse(varargin{:});
16  planeWaveExcitations    = p.Results.planeWaveExcitations;
17  thetaMesh               = p.Results.thetaMesh;
18  phiMesh                 = p.Results.phiMesh;
19  thetaScanRad            = p.Results.thetaScanRad;
20  phiScanRad              = p.Results.phiScanRad;
21  freqIndex               = p.Results.freqIndex;
22  polarisation            = p.Results.polarisation;
23
24  %% Beamformer Calculation
25
26  scanLogic = phiMesh(:) == phiScanRad & thetaMesh(:) == thetaScanRad;
27  switch polarisation
28      case 'y'
29          CFM = squeeze(planeWaveExcitations(scanLogic, 2, :, freqIndex)).';
30      case 'z'
31          CFM = squeeze(planeWaveExcitations(scanLogic, 3, :, freqIndex)).';
32      case 'x'
33          CFM = squeeze(planeWaveExcitations(scanLogic, 1, :, freqIndex)).';
34      otherwise
35          CFM = squeeze(planeWaveExcitations(scanLogic, 1, :, freqIndex)).' +
                ↪  ...
36              squeeze(planeWaveExcitations(scanLogic, 2, :, freqIndex)).' +
                    ↪ ...
37              squeeze(planeWaveExcitations(scanLogic, 3, :, freqIndex)).';
38  end
39
40  beamformerCFM = CFM.'/norm(CFM);
```

D.2.4 calculate_conventional_beamformer.m

```
function beamformerConventional = calculate_conventional_beamformer(
    ↪ varargin)

% the idea:
% create a surface, the same shape and size of the aperture, of phase shift
% over the aperture for ideal theta and phi scan directions. Pick from this
% surface points that correspond to the element positions. This will be
% your beamformer.

p = inputParser();
p.CaseSensitive = false;

p.addOptional('elementPos',[]);
p.addOptional('thetaScanRad',[]);
p.addOptional('phiScanRad',[]);
p.addOptional('lambda',[]);

p.parse(varargin{:});
elementPos      = p.Results.elementPos;
thetaScanRad    = p.Results.thetaScanRad;
phiScanRad      = p.Results.phiScanRad;
lambda          = p.Results.lambda;

beamformerConventional = zeros(size(elementPos, 1), 1);

for i = 1:size(elementPos, 1)
    x = elementPos(i, 1);
    y = elementPos(i, 2);

    ang = 2 * pi / lambda * ...
        sin(thetaScanRad) * ...
        sqrt(x^2 + y^2) * ...
        sin(pi/2 + phiScanRad - atan(y/x));

    if x < 0
        ang = ang * -1;
    end

    if isnan(ang)
        beamformerConventional(i) = 1;
    else
        beamformerConventional(i) = cos(ang) + 1j * sin(ang);
    end
end

beamformerConventional = conj(beamformerConventional) / ...
    norm(beamformerConventional, 2);
end
```

D.2.5 calculate_gain_tx.m

```
function gain = calculate_gain_tx(varargin)
% Determines the transmitter gain between the specified input and output
% ports, either microwave gain or radiated gain. This can be used to
% determine circuit, system, and array gain depending on which input and
% output ports are specified.
%
% Args:
%   EEPs
%   observationDistance
%   Const
%   beamformer
%   resultsMW
%   freqIndex
%   Zo
%   inputPorts
%   outputPorts
%   gainType    (string: ['radiation', 'MW'])

%% input parsing

p = inputParser();
p.CaseSensitive = false;
```

```
23
24 p.addOptional('EEPs', []);
25 p.addOptional('observationDistance', []);
26 p.addOptional('Const', []);
27 p.addOptional('beamformer', []);
28 p.addOptional('resultsMW', []);
29 p.addOptional('freqIndex', []);
30 p.addOptional('ZoAnt', 50);
31 p.addOptional('ZoTxSubSys', 50);
32 p.addOptional('inputPorts', []);
33 p.addOptional('outputPorts', []);
34 p.addOptional('gainType', []);
35
36 p.parse(varargin{:});
37 EEPs                = p.Results.EEPs;
38 observationDistance = p.Results.observationDistance;
39 Const               = p.Results.Const;
40 beamformer          = p.Results.beamformer;
41 resultsMW           = p.Results.resultsMW;
42 freqIndex           = p.Results.freqIndex;
43 ZoAnt               = p.Results.ZoAnt;
44 ZoTxSubSys          = p.Results.ZoTxSubSys;
45 inputPorts          = p.Results.inputPorts;
46 outputPorts         = p.Results.outputPorts;
47 gainType            = p.Results.gainType;
48
49 if isempty(gainType)
50     error('Gain type must be specified')
51 end
52
53 %% Gain calculation
54
55 EEPs    = EEPs(:, :, :, freqIndex);
56 Const   = Const(:, freqIndex);
57
58 if strcmp(gainType, 'radiation')
59     portsArrayElement = outputPorts;
60
61     [vAnt, ~, ~] = abToVIP( ...
62         resultsMW.a(portsArrayElement, 1, freqIndex), ...
63         resultsMW.b(portsArrayElement, 1, freqIndex), ...
64         ZoAnt);
65
66     Etot = zeros(size(EEPs, 1), 3);
67     for n = 1 : length(beamformer)
68         Etot = Etot + EEPs(:,:,n) * vAnt(n);
69     end
70
71     G               = Etot * observationDistance * ...
72                         exp(1j * Const.k * observationDistance);
                            ↪            % Far field function
73     U               = ( ...
74                         abs(G(:, 1)).^2 + ...
75                         abs(G(:, 2)).^2 + ...
76                         abs(G(:, 3)).^2 ...
77                         ) / ...
78                         (2 * 120 * pi);
79
80     [~, ~, Pin]     = abToVIP( ...
81                               resultsMW.a(inputPorts, 1, freqIndex), ...
82                               zeros(size(resultsMW.b(inputPorts, 1, freqIndex
                                  ↪ ))), ...
83                               ZoTxSubSys);
84     Pin             = sum(abs(Pin)) / 2;
85     Uiso            = Pin / (4 * pi);
86     gain            = 10 * log10(U / Uiso);
87
88 elseif strcmp(gainType, 'MW')
89     powerRatio      = resultsMW.b(outputPorts, 1, freqIndex) ./ ...
90                       resultsMW.a(inputPorts, 1, freqIndex);
91     acceptedPower   = real(powerRatio.' * beamformer);
92
93     gain = zeros(size(EEPs)) + acceptedPower;
94     gain = 10 * log10(gain);
```

```
95  else
96      error(['Gain type ', gainType, ' not understood'])
97  end
98
99  % todo: check Pin and acceptedPower, should be real or abs?
100 end
```

D.2.6 calculate_isolation_dual_beamformers.m

```
1   function [beamformerTxIso, beamformerRxIso] =
        ↪ calculate_isolation_dual_beamformers(varargin)
2   % Determines the best tx beamformer to satisfy the isolation requirements.
3   % This method will only improve isolation after the beamformers, and could
4   % remove the necessity for filters before sampling. However, this does not
5   % solve the receiver saturation problem.
6   %
7   % Args:
8   %   beamformerRxRef
9   %   beamformerTxRef
10  %
11  % Outputs:
12  %   beamformerTxIso
13
14  %% input parsing
15
16  p = inputParser();
17  p.CaseSensitive = false;
18
19  p.addOptional('beamformerRxRef',[]);
20  p.addOptional('beamformerTxRef',[]);
21  p.addOptional('targetIso',[]);
22  p.addOptional('sSystem',[]);
23  p.addOptional('sParamPortsRx',[]);
24  p.addOptional('sParamPortsTx',[]);
25  p.addOptional('FREQ',[]);
26  p.addOptional('F_RX_ARRAY',[]);
27  p.addOptional('MW', []);
28  p.addOptional('portsLoadTx', []);
29  p.addOptional('portsLoadRx', []);
30  p.addOptional('Load', []);
31  p.addOptional('transceiverSystem', []);
32  p.addOptional('antenna', []);
33  p.addOptional('loadArray', []);
34  p.addOptional('convergenceDelta',1e-2);
35
36  p.parse(varargin{:});
37  beamformerRxRef      = p.Results.beamformerRxRef;
38  beamformerTxRef      = p.Results.beamformerTxRef;
39  targetIso            = p.Results.targetIso;
40  sSystem              = p.Results.sSystem;
41  sParamPortsRx        = p.Results.sParamPortsRx;
42  sParamPortsTx        = p.Results.sParamPortsTx;
43  FREQ                 = p.Results.FREQ;
44  F_RX_ARRAY           = p.Results.F_RX_ARRAY;
45  MW                   = p.Results.MW;
46  portsLoadTx          = p.Results.portsLoadTx;
47  portsLoadRx          = p.Results.portsLoadRx;
48  Load                 = p.Results.Load;
49  transceiverSystem    = p.Results.transceiverSystem;
50  antenna              = p.Results.antenna;
51  loadArray            = p.Results.loadArray;
52  convergenceDelta     = p.Results.convergenceDelta;
53
54  beamformerRxRef      = conj(beamformerRxRef);
55  beamformerTxPrevious  = beamformerTxRef;
56  beamformerRxPrevious  = beamformerRxRef;
57  loop                 = true;
58
59  %% Tx transfer matrix computation
60
61  fRxLogic = zeros(size(FREQ, 1), 1);
62  for i = 1:length(F_RX_ARRAY)
63      fRxLogic = fRxLogic + (FREQ(:) == F_RX_ARRAY(i));
```

```
64  end
65  fRxLogic = logical(fRxLogic);
66
67  sSystem = sSystem(:, :, fRxLogic);
68
69  nFreq = length(F_RX_ARRAY);
70  nChannels = length(beamformerRxRef);
71
72  sRxTx = zeros(nChannels * nFreq, ...
73                nChannels);
74
75  rxBfFreqTrans = diag(ones(nChannels, 1));
76  for f = 1:nFreq
77      for i = 1:nChannels
78          outputPort = sParamPortsRx(i);
79          for j = 1:nChannels
80              inputPort = sParamPortsTx(j);
81              sRxTx(i + nChannels * (f - 1), j) = sSystem(outputPort,
                    ↪ inputPort, f);
82          end
83      end
84      if f ~= 1
85          rxBfFreqTrans = [rxBfFreqTrans; diag(ones(nChannels, 1))];
86      end
87  end
88
89  beamformerRxIso = 1;
90
91
92  %% Isolation beamformer calculation
93
94  comparitiveIso = zeros(nFreq, 1) + 10 ^ (-targetIso / 10);
95
96  while loop
97      cvx_begin quiet
98          variable beamformerTxIso(nChannels) complex
99          variable beamformerRxIso(nChannels) complex
100         minimize(max(norm(beamformerTxIso - beamformerTxRef, 2), norm(
                ↪ beamformerRxIso - beamformerRxRef, 2)))
101         subject to
102             abs((sRxTx * beamformerTxIso).' * (rxBfFreqTrans *
                    ↪ beamformerRxIso)) ...
103                 <= comparitiveIso;
104     cvx_end
105
106     beamformerTxIso = beamformerTxIso / norm(beamformerTxIso, 2);
107     beamformerRxIso = beamformerRxIso / norm(beamformerRxIso, 2);
108     deltaVTx = abs(beamformerTxIso - beamformerTxPrevious) ...
109              ./ abs(beamformerTxPrevious);
110     deltaVRx = abs(beamformerTxIso - beamformerRxPrevious) ...
111              ./ abs(beamformerRxPrevious);
112
113     deltaV = max([deltaVTx; deltaVRx]);
114
115     [MW, loadIsoArray] = define_load_source_ports( ...
116         'MW', MW, ...
117         'sourcePortArray', portsLoadTx, ...
118         'excitationArray', beamformerTxIso, ...
119         'Load', Load ...
120         );
121
122     % === START of MW-Engine ===
123     resultsMW_iso = RunMWSimulator([transceiverSystem, antenna,
            ↪ loadIsoArray, loadArray(nChannels+1:2*nChannels)], MW);
124     % === End of MW-Engine ===
125
126     i = 1;
127     isoBF = zeros(nFreq, 1);
128     for f = 1:length(FREQ)
129         if any(FREQ(f) == F_RX_ARRAY)
130             iso = resultsMW_iso.a(portsLoadRx, 1, f).' * beamformerRxIso;
131             isoBF(i) = abs(iso);
132             i = i + 1;
133         end
```

```
    end

    isoMiss = comparitiveIso ./ isoBF;
    for i = 1:nFreq
        if isoMiss(i) > 1
            isoMiss(i) = 1;
        end
    end

    if deltaV < convergenceDelta
        loop = false;
    elseif isoMiss >= 1
        loop = false;
    else
        beamformerTxPrevious = beamformerTxIso;
        beamformerRxPrevious = beamformerRxIso;
        comparitiveIso = comparitiveIso .* isoMiss;
    end
end

end
```

D.2.7 calculate_isolation_rx_beamformer.m

```
function beamformerRxIso = calculate_isolation_rx_beamformer(varargin)
% Determines the best rx beamformer to satisfy the isolation requirements.
% This method will only improve isolation after the beamformers, and could
% remove the necessity for filters before sampling. However, this does not
% solve the receiver saturation problem.
%
% Args:
%   beamformerRxRef
%   beamformerTxRef
%
% Outputs:
%   beamformerTxIso

%% input parsing

p = inputParser();
p.CaseSensitive = false;

p.addOptional('beamformerRxRef',[]);
p.addOptional('targetIso',[]);
p.addOptional('resultsMW_MGB',[]);
p.addOptional('portsLoadRx',[]);
p.addOptional('FREQ',[]);
p.addOptional('F_RX_ARRAY',[]);
p.addOptional('convergenceDelta',1e-2);

p.parse(varargin{:});
beamformerRxRef     = p.Results.beamformerRxRef;
targetIso           = p.Results.targetIso;
resultsMW_MGB       = p.Results.resultsMW_MGB;
portsLoadRx         = p.Results.portsLoadRx;
FREQ                = p.Results.FREQ;
F_RX_ARRAY          = p.Results.F_RX_ARRAY;
convergenceDelta    = p.Results.convergenceDelta;

beamformerCompare   = beamformerRxRef;
beamformerPrevious  = beamformerRxRef;
loop                = true;

nFreq = length(F_RX_ARRAY);

%% Isolation beamformer calculation

fRxLogic = zeros(size(FREQ, 1), 1);
for f = 1:length(F_RX_ARRAY)
    fRxLogic = fRxLogic + (FREQ(:) == F_RX_ARRAY(f));
end
fRxLogic = logical(fRxLogic);

```

```

rxInput = squeeze(resultsMW_MGB.a(portsLoadRx, 1, fRxLogic));

comparitiveIso = zeros(nFreq, 1) + 10 ^ (-targetIso / 10) * isoScale;

n = size(beamformerRxRef, 1);
while loop
    cvx_begin quiet
        variable beamformerRxIso(n) complex
        minimize( norm(beamformerRxIso - beamformerCompare, 2) )
        subject to
            abs(rxInput.' * conj(beamformerRxIso)) <= sqrt(comparitiveIso);
    cvx_end

    beamformerRxIso = beamformerRxIso / norm(beamformerRxIso, 2);
    deltaV = abs(beamformerRxIso - beamformerPrevious) ...
             ./ abs(beamformerPrevious);

    beamformerCompare = beamformerRxIso;
    isoTxBF = zeros(nFreq, 1);
    for f = 1:length(F_RX_ARRAY)
        iso =  abs(rxInput(:, f).' * conj(beamformerRxIso));
        isoTxBF(f) = abs(iso);
    end

    isoMiss = comparitiveIso ./ isoTxBF;
    for f = 1:nFreq
        if isoMiss(f) > 1
            isoMiss(f) = 1;
        end
    end

    if deltaV < convergenceDelta
        loop = false;
    elseif isoMiss >= 1
        loop = false;
    else
        beamformerPrevious = beamformerRxIso;
        comparitiveIso = comparitiveIso .* isoMiss;
    end
end

end
```

D.2.8 calculate_isolation_tx_beamformer.m

```
function beamformerTxIso = calculate_isolation_tx_beamformer(varargin)
% Determines the best tx beamformer to satisfy the isolation requirements.
% This method will only improve isolation after the beamformers, and could
% remove the necessity for filters before sampling. However, this does not
% solve the receiver saturation problem.
%
% Args:
%   beamformerRxRef
%   beamformerTxRef
%
% Outputs:
%   beamformerTxIso

%% input parsing

p = inputParser();
p.CaseSensitive = false;

p.addOptional('beamformerRxRef',[]);
p.addOptional('beamformerTxRef',[]);
p.addOptional('targetIso',[]);
p.addOptional('sSystem',[]);
p.addOptional('sParamPortsRx',[]);
p.addOptional('sParamPortsTx',[]);
p.addOptional('FREQ',[]);
p.addOptional('F_RX_ARRAY',[]);
```

```
27 p.addOptional('MW', []);
28 p.addOptional('portsLoadTx', []);
29 p.addOptional('portsLoadRx', []);
30 p.addOptional('LoadTx', []);
31 p.addOptional('transceiverSystem', []);
32 p.addOptional('antenna', []);
33 p.addOptional('loadArray', []);
34 p.addOptional('convergenceDelta',1e-2);
35
36 p.parse(varargin{:});
37 beamformerRxRef      = p.Results.beamformerRxRef;
38 beamformerTxRef      = p.Results.beamformerTxRef;
39 targetIso            = p.Results.targetIso;
40 sSystem              = p.Results.sSystem;
41 sParamPortsRx        = p.Results.sParamPortsRx;
42 sParamPortsTx        = p.Results.sParamPortsTx;
43 FREQ                 = p.Results.FREQ;
44 F_RX_ARRAY           = p.Results.F_RX_ARRAY;
45 MW                   = p.Results.MW;
46 portsLoadTx          = p.Results.portsLoadTx;
47 portsLoadRx          = p.Results.portsLoadRx;
48 LoadTx               = p.Results.LoadTx;
49 transceiverSystem    = p.Results.transceiverSystem;
50 antenna              = p.Results.antenna;
51 loadArray            = p.Results.loadArray;
52 convergenceDelta     = p.Results.convergenceDelta;
53
54 beamformerRxRef      = beamformerRxRef;
55 beamformerPrevious   = beamformerTxRef;
56 beamformerCompare    = beamformerTxRef;
57 loop                 = true;
58
59 %% Tx transfer matrix computation
60
61 fRxLogic = zeros(size(FREQ, 1), 1);
62 for i = 1:length(F_RX_ARRAY)
63     fRxLogic = fRxLogic + (FREQ(:) == F_RX_ARRAY(i));
64 end
65 fRxLogic = logical(fRxLogic);
66
67 sSystem = sSystem(:, :, fRxLogic);
68
69 nFreq = length(F_RX_ARRAY);
70 nChannels = length(beamformerRxRef);
71
72 sRxTx = zeros(nChannels * nFreq, ...
73               nChannels);
74
75 beamformerFreqExp = zeros(nChannels * nFreq, nFreq);
76 for f = 1:nFreq
77     for i = 1:nChannels
78         outputPort = sParamPortsRx(i);
79         for j = 1:nChannels
80             inputPort = sParamPortsTx(j);
81             sRxTx(i + nChannels * (f - 1), j) = sSystem(outputPort,
                   ↪ inputPort, f);
82         end
83     end
84     beamformerFreqExp([(f-1) * nChannels + 1: f * nChannels], f) =
       ↪ beamformerRxRef;
85 end
86
87
88 %% Isolation beamformer calculation
89
90 isoScale = 1;
91
92 comparitiveIso = zeros(nFreq, 1) + 10 ^ (-targetIso / 10) * isoScale;
93
94 while loop
95     cvx_begin quiet
96         variable beamformerTxIso(nChannels) complex
97         minimize( norm(beamformerTxIso - beamformerCompare, 2) )
98         subject to
```

```
99              (abs((sRxTx * beamformerTxIso).' * conj(beamformerFreqExp))).'
                    ↪ ...
100                  <= sqrt(comparitiveIso);
101     cvx_end
102
103     if any(not(isnumeric(beamformerTxIso)))
104         disp('fail')
105     end
106     if any(isnan(beamformerTxIso))
107         disp('fail')
108     end
109     if any(isinf(beamformerTxIso))
110         disp('fail')
111     end
112
113     beamformerTxIso = beamformerTxIso / norm(beamformerTxIso, 2);
114     deltaV = abs(beamformerTxIso - beamformerPrevious) ...
115               ./ abs(beamformerPrevious);
116
117     beamformerCompare = beamformerTxIso;
118     if any(not(isnumeric(beamformerCompare)))
119         disp('fail')
120     end
121     if any(isnan(beamformerCompare))
122         disp('fail')
123     end
124     if any(isinf(beamformerCompare))
125         disp('fail')
126     end
127
128     [MW, loadIsoArray] = define_load_source_ports( ...
129         'MW', MW, ...
130         'sourcePortArray', portsLoadTx, ...
131         'excitationArray', beamformerTxIso, ...
132         'Load', LoadTx ...
133         );
134
135     % === START of MW-Engine ===
136     resultsMW_iso = RunMWSimulator([transceiverSystem, antenna,
            ↪ loadIsoArray, loadArray(nChannels+1:2*nChannels)], MW);
137     % === End of MW-Engine ===
138
139     i = 1;
140     isoTxBF = zeros(nFreq, 1);
141
142     for f = 1:length(FREQ)
143         if any(FREQ(f) == F_RX_ARRAY)
144             iso = resultsMW_iso.a(portsLoadRx, 1, f).' * conj(
                        ↪ beamformerRxRef);
145             isoTxBF(i) = abs(iso);
146             i = i + 1;
147         end
148     end
149
150     isoMiss = comparitiveIso ./ isoTxBF;
151     for i = 1:nFreq
152         if isoMiss(i) > 1
153             isoMiss(i) = 1;
154         end
155     end
156
157     if deltaV < convergenceDelta
158         loop = false;
159     elseif isoMiss >= 1
160         loop = false;
161     else
162         beamformerPrevious = beamformerTxIso;
163         comparitiveIso = comparitiveIso .* isoMiss;
164     end
165 end
166
167 end
```

D.2.9 calculate_loaded_element_patterns.m

```
function [EEPs_loaded, voltageAntennaPorts] = ...
    calculate_loaded_element_patterns(varargin )
% Calculate the embedded element patterns for a realistically loaded
% system. All ports are terminated in matched loads. One port is excited
% with a a-wave of magnitude 1. The resulting voltage across the ports at
% the array elements is recorded and multiplied with the EEPs_SC ref plane
% antenna to get the corresponding radiation pattern.
%
% Args:
%   nChannels
%   resultsMW
%   EEPs_SC
%   freqIndex
%   Zo
%   portsArrayElements

%% input parsing

p = inputParser();
p.CaseSensitive = true;

p.addOptional('nChannels', []);
p.addOptional('resultsMW', []);
p.addOptional('EEPs_SC', []);
p.addOptional('freqIndex', []);
p.addOptional('Zo', 50);
p.addOptional('portsArrayElements', []);

p.parse(varargin{:});
nChannels           = p.Results.nChannels;
resultsMW           = p.Results.resultsMW;
EEPs_SC             = p.Results.EEPs_SC;
freqIndex           = p.Results.freqIndex;
Zo                  = p.Results.Zo;
portsArrayElements  = p.Results.portsArrayElements;

%% Loaded EEP calculation

voltageAntennaPorts   = zeros(nChannels, nChannels);
EEPs_loaded           = zeros(size(EEPs_SC(:, :, :, 1)));

for n = 1 : nChannels
    [voltageAntennaPorts(:, n), ~, ~] = abToVIP( ...
        resultsMW.a(portsArrayElements, n, freqIndex), ...
        resultsMW.b(portsArrayElements, n, freqIndex), ...
        Zo);
            ↪
            ↪ % Effect of the excitation of 1 Tx port on all the antenna
            ↪ ports
    flattenedArray = reshape( ...
        EEPs_SC(:, :, :, freqIndex), [], nChannels);
    voltageArray                = flattenedArray * voltageAntennaPorts(:, n
        ↪ );
    EEPs_loaded(:, :, n)        = reshape(voltageArray, [], 3);
end
```

D.2.10 calculate_mask_isolation_rx_beamformer.m

```
function beamformerRxMaskIso = calculate_mask_isolation_rx_beamformer(
    ↪ varargin)
% Determines the best tx beamformer to satisfy the isolation requirements.
% This method will only improve isolation after the beamformers, and could
% remove the necessity for filters before sampling. However, this does not
% solve the receiver saturation problem.
%
% Args:
%   beamformerRxRef
%   beamformerTxRef
%
% Outputs:
%   beamformerRxMaskIso
```

```
13
14 % todo: change the rx_array and things to arrays in which the mask and
15 % isolation has to be applied
16
17 %% input parsing
18
19 p = inputParser();
20 p.CaseSensitive = false;
21
22 p.addOptional('beamformerRxRef',[]);
23 p.addOptional('beamformerTxRef',[]);
24 p.addOptional('targetIso',[]);
25 p.addOptional('FREQ',[]);
26 p.addOptional('F_RX_ARRAY',[]);
27 p.addOptional('F_ARRAY', []);
28 p.addOptional('MW', []);
29 p.addOptional('portsLoadTx', []);
30 p.addOptional('portsLoadRx', []);
31 p.addOptional('Load', []);
32 p.addOptional('transceiverSystem', []);
33 p.addOptional('antenna', []);
34 p.addOptional('loadArray', []);
35 p.addOptional('convergenceDelta',1e-2);
36 p.addOptional('gainMask', []);
37 p.addOptional('EEPs', []);
38 p.addOptional('gainOpt', []);
39 p.addOptional('polarisation', 'x');
40 p.addOptional('compensatingOffset', 0);
41 p.addOptional('lambda', []);
42 p.addOptional('yAnt', []);
43 p.addOptional('zSys', []);
44
45 p.parse(varargin{:});
46 beamformerRxRef     = p.Results.beamformerRxRef;
47 beamformerTxRef     = p.Results.beamformerTxRef;
48 targetIso           = p.Results.targetIso;
49 FREQ                = p.Results.FREQ;
50 F_RX_ARRAY          = p.Results.F_RX_ARRAY;
51 F_ARRAY             = p.Results.F_ARRAY;
52 MW                  = p.Results.MW;
53 portsLoadTx         = p.Results.portsLoadTx;
54 portsLoadRx         = p.Results.portsLoadRx;
55 Load                = p.Results.Load;
56 transceiverSystem   = p.Results.transceiverSystem;
57 antenna             = p.Results.antenna;
58 loadArray           = p.Results.loadArray;
59 convergenceDelta    = p.Results.convergenceDelta;
60 gainMask            = p.Results.gainMask;
61 EEPs                = p.Results.EEPs;
62 gainOpt             = p.Results.gainOpt;
63 polarisation        = p.Results.polarisation;
64 compensatingOffset  = p.Results.compensatingOffset;
65 lambda              = p.Results.lambda;
66 yAnt                = p.Results.yAnt;
67 zSys                = p.Results.zSys;
68
69 beamformerRxRef     = conj(beamformerRxRef);
70 beamformerPrevious  = beamformerTxRef;
71 loop                = true;
72
73 %% Rx input matrix computation
74
75 fRxLogic = zeros(size(FREQ, 1), 1);
76 for i = 1:length(F_RX_ARRAY)
77     fRxLogic = fRxLogic + (FREQ(:) == F_RX_ARRAY(i));
78 end
79 fRxLogic = logical(fRxLogic);
80
81 nFreq = length(F_RX_ARRAY);
82 nChannels = length(beamformerRxRef);
83
84 [MW, loadIsoArray] = define_load_source_ports( ...
85     'MW', MW, ...
86     'sourcePortArray', portsLoadTx, ...
```

```
87      'excitationArray', beamformerTxRef, ...
88      'Load', Load ...
89      );
90
91  % === START of MW-Engine ===
92  resultsMW = RunMWSimulator([transceiverSystem, antenna, loadIsoArray,
        ↪ loadArray(nChannels+1:2*nChannels)], MW);
93  % === End of MW-Engine ===
94
95  rxInput = squeeze(resultsMW.a(portsLoadRx, 1, fRxLogic));
96
97  %% Mask setup
98
99  freqIndex = arrayfun(@(x) find(FREQ == x, 1) , F_ARRAY);
100 freqPos = zeros(length(FREQ), 1);
101 for i = 1:length(freqIndex)
102     freqPos(freqIndex(i)) = 1;
103 end
104 freqPos = logical(freqPos);
105
106
107 E_p = EEPs(:, :, :, freqPos);
108 if strcmp(polarisation, 'x')
109     E_p = squeeze(E_p(:, 1, :, :));
110 elseif strcmp(polarisation, 'y')
111     E_p = squeeze(E_p(:, 2, :, :));
112 elseif strcmp(polarisation, 'rhc')
113     E_x = squeeze(E_p(:, 1, :, :));
114     E_y = squeeze(E_p(:, 2, :, :));
115     E_p = 1 / sqrt(2) * (E_x + 1j * E_y);
116 elseif strcmp(polarisation, 'lhc')
117     E_x = squeeze(E_p(:, 1, :, :));
118     E_y = squeeze(E_p(:, 2, :, :));
119     E_p = 1 / sqrt(2) * (E_x - 1j * E_y);
120 end
121
122 gainOpt = gainOpt(:, freqIndex);
123
124 gainMask = reshape(gainMask, [], 1);
125 gainMask = gainMask - compensatingOffset;
126
127 if nFreq == 1
128     E_p = squeeze(E_p);
129     E_pMaxGain          = E_p * beamformerOpt;
130     E_pMaxGainScalar    = max(abs(E_pMaxGain));
131     E_p                 = E_p ./ E_pMaxGainScalar;
132 else
133     for f = 1:nFreq
134         E_pMaxGain          = squeeze(E_p(:, :, f)) * beamformerTxRef;
135         E_pMaxGainScalar    = max(abs(E_pMaxGain));
136         E_p(:, :, f)        = E_p(:, :, f) ./ E_pMaxGainScalar;
137     end
138
139     E_p = permute(E_p, [1, 3, 2]);
140     E_p = reshape(E_p, [], size(E_p, 3));
141
142     oldGainMask = gainMask;
143     for i = 2:nFreq
144         gainMask(:, i) = oldGainMask;
145     end
146     gainMask = reshape(gainMask, [], 1);
147 end
148
149
150 maxGainScalar = zeros(size(EEPs, 1), nFreq);
151 for f = 1:nFreq
152     maxGainScalar(:, f) = ones(size(EEPs, 1), 1) * max(gainOpt(:, f));
153 end
154
155 maxGainScalar = reshape(maxGainScalar, [], 1);
156
157 E_pNorm = E_p;
158
159 %% determine the E-field mask in the cutting plane
```

```
E_mask    = 10 .^ ((gainMask - maxGainScalar) / 20);
for i = 1:size(E_mask, 1)
    if isinf(E_mask(i))
        E_mask(i) = 1;
    end
end

%% Isolation beamformer calculation

% todo: compensate for active gain on the Rx side

comparitiveIso = zeros(nFreq, 1) + 10 ^ (-targetIso/ 10);

beamformerRxCompare = beamformerRxRef;
while loop
    cvx_begin quiet
        variable beamformerRxMaskIso(nChannels) complex
        minimize( norm(beamformerRxMaskIso - beamformerRxCompare, 2) )
        subject to
            abs(rxInput.' * beamformerRxMaskIso) <= comparitiveIso;
            abs(E_pNorm * beamformerRxMaskIso) <= E_mask;
    cvx_end

    beamformerRxMaskIso = beamformerRxMaskIso / norm(beamformerRxMaskIso,
        ↪ 2);
    deltaV = abs(beamformerRxMaskIso - beamformerPrevious) ...
             ./ abs(beamformerPrevious);

    beamformerRxCompare = beamformerRxMaskIso;

    i = 1;
    isoTxBF = zeros(nFreq, 1);
    for f = 1:length(FREQ)
        if any(FREQ(f) == F_RX_ARRAY)
            iso = resultsMW.a(portsLoadRx, 1, f).' * beamformerRxMaskIso;
            isoTxBF(i) = abs(iso);
            i = i + 1;
        end
    end

    isoMiss = comparitiveIso ./ isoTxBF;

    gainRxOvershoot = zeros(size(EEPs, 1), nFreq);
    maskOvershoot = zeros(size(EEPs, 1), nFreq);
    for i = 1:nFreq
        gainRxOvershoot(:, i) = calculate_system_gain_rx( ...
            'rxWaves', EEPs ...
           ,'beamformer', conj(beamformerRxMaskIso) ...
           ,'lambda', lambda ...
           ,'freqIndex', i ...
           ,'yAnt', yAnt ...
           ,'zSys', zSys ...
            );

        gainMaskReshape = reshape(gainMask, [], nFreq);
        for p = 1:length(gainRxOvershoot(:, i))
            maskOvershoot(p, i) = gainRxOvershoot(p, i) - gainMaskReshape(p
                ↪ , i);
            if maskOvershoot(p, i) < 0
                maskOvershoot(p, i) = 0;
            end
        end
    end
    maskOvershoot = reshape(maskOvershoot, [], 1);

    requirementsLogic = [isoMiss >= 1; maskOvershoot <= 0];
    if requirementsLogic
        loop = false;
    elseif deltaV < convergenceDelta
        loop = false;
        if any(isoMiss < 1)
            disp('Rx isolation failed')
        end
        if any(maskOvershoot > 0)
```

```
232             disp('Rx mask failed')
233         end
234         disp('Rx beamformer convergence reached')
235     else
236         beamformerPrevious = beamformerRxMaskIso;
237         maskOvershoot = 10 .^ (maskOvershoot / 20);
238         E_mask = E_mask ./ maskOvershoot;
239     end
240 end
241
242 beamformerRxMaskIso = conj(beamformerRxMaskIso);
243
244 end
```

D.2.11 calculate_mask_isolation_tx_beamformer.m

```
1  function beamformerTxMaskIso = calculate_mask_isolation_tx_beamformer(
       ↪ varargin)
2  % Determines the best tx beamformer to satisfy the isolation requirements.
3  % This method will only improve isolation after the beamformers, and could
4  % remove the necessity for filters before sampling. However, this does not
5  % solve the receiver saturation problem.
6  %
7  % Args:
8  %   beamformerRxRef
9  %   beamformerTxRef
10 %
11 % Outputs:
12 %   beamformerTxMaskIso
13
14 % todo: change the rx_array and things to arrays in which the mask and
15 % isolation has to be applied
16
17 %% input parsing
18
19 p = inputParser();
20 p.CaseSensitive = false;
21
22 p.addOptional('beamformerRxRef',[]);
23 p.addOptional('beamformerTxRef',[]);
24 p.addOptional('targetIso',[]);
25 p.addOptional('sSystem',[]);
26 p.addOptional('sParamPortsRx',[]);
27 p.addOptional('sParamPortsTx',[]);
28 p.addOptional('FREQ',[]);
29 p.addOptional('F_RX_ARRAY',[]);
30 p.addOptional('F_ARRAY', []);
31 p.addOptional('MW', []);
32 p.addOptional('portsLoadTx', []);
33 p.addOptional('portsLoadRx', []);
34 p.addOptional('Load', []);
35 p.addOptional('transceiverSystem', []);
36 p.addOptional('antenna', []);
37 p.addOptional('loadArray', []);
38 p.addOptional('convergenceDelta',1e-2);
39 p.addOptional('gainMask', []);
40 p.addOptional('EEPs', []);
41 p.addOptional('gainOpt', []);
42 p.addOptional('polarisation', 'x');
43 p.addOptional('compensatingOffset', 0);
44 p.addOptional('observationDistance', []);
45 p.addOptional('Const', []);
46 p.addOptional('Zo', []);
47 p.addOptional('inputPorts', []);
48 p.addOptional('outputPorts', []);
49 p.addOptional('gainType', 'radiation');
50
51 p.parse(varargin{:});
52 beamformerRxRef     = p.Results.beamformerRxRef;
53 beamformerTxRef     = p.Results.beamformerTxRef;
54 targetIso           = p.Results.targetIso;
55 sSystem             = p.Results.sSystem;
56 sParamPortsRx       = p.Results.sParamPortsRx;
```

```
57  sParamPortsTx        = p.Results.sParamPortsTx;
58  FREQ                 = p.Results.FREQ;
59  F_RX_ARRAY           = p.Results.F_RX_ARRAY;
60  F_ARRAY              = p.Results.F_ARRAY;
61  MW                   = p.Results.MW;
62  portsLoadTx          = p.Results.portsLoadTx;
63  portsLoadRx          = p.Results.portsLoadRx;
64  Load                 = p.Results.Load;
65  transceiverSystem    = p.Results.transceiverSystem;
66  antenna              = p.Results.antenna;
67  loadArray            = p.Results.loadArray;
68  convergenceDelta     = p.Results.convergenceDelta;
69  gainMask             = p.Results.gainMask;
70  EEPs                 = p.Results.EEPs;
71  gainOpt              = p.Results.gainOpt;
72  polarisation         = p.Results.polarisation;
73  compensatingOffset   = p.Results.compensatingOffset;
74  observationDistance  = p.Results.observationDistance;
75  Const                = p.Results.Const;
76  Zo                   = p.Results.Zo;
77  inputPorts           = p.Results.inputPorts;
78  outputPorts          = p.Results.outputPorts;
79  gainType             = p.Results.gainType;
80
81  beamformerRxRef      = conj(beamformerRxRef);
82  beamformerPrevious   = beamformerTxRef;
83  loop                 = true;
84
85  %% Tx transfer matrix computation
86
87  fRxLogic = zeros(size(FREQ, 1), 1);
88  for i = 1:length(F_RX_ARRAY)
89      fRxLogic = fRxLogic + (FREQ(:) == F_RX_ARRAY(i));
90  end
91  fRxLogic = logical(fRxLogic);
92
93  sSystem = sSystem(:, :, fRxLogic);
94
95  nFreq = length(F_RX_ARRAY);
96  nChannels = length(beamformerRxRef);
97
98  sRxTx = zeros(nChannels * nFreq, ...
99                nChannels);
100
101 beamformerFreqExp = zeros(nChannels * nFreq, nFreq);
102 for f = 1:nFreq
103     for i = 1:nChannels
104         outputPort = sParamPortsRx(i);
105         for j = 1:nChannels
106             inputPort = sParamPortsTx(j);
107             sRxTx(i + nChannels * (f - 1), j) = sSystem(outputPort,
                    ↪ inputPort, f);
108         end
109     end
110     beamformerFreqExp([(f-1) * nChannels + 1: f * nChannels], f) =
        ↪ beamformerRxRef;
111 end
112
113 %% Mask setup
114
115 freqIndex = arrayfun(@(x) find(FREQ == x, 1) , F_ARRAY);
116 freqPos = zeros(length(FREQ), 1);
117 for i = 1:length(freqIndex)
118     freqPos(freqIndex(i)) = 1;
119 end
120 freqPos = logical(freqPos);
121
122
123 E_p = EEPs(:, :, :, freqPos);
124 if strcmp(polarisation, 'x')
125     E_p = squeeze(E_p(:, 1, :, :));
126 elseif strcmp(polarisation, 'y')
127     E_p = squeeze(E_p(:, 2, :, :));
128 elseif strcmp(polarisation, 'rhc')
```

```
129     E_x = squeeze(E_p(:, 1, :, :));
130     E_y = squeeze(E_p(:, 2, :, :));
131     E_p = 1 / sqrt(2) * (E_x + 1j * E_y);
132 elseif strcmp(polarisation, 'lhc')
133     E_x = squeeze(E_p(:, 1, :, :));
134     E_y = squeeze(E_p(:, 2, :, :));
135     E_p = 1 / sqrt(2) * (E_x - 1j * E_y);
136 end
137
138 gainOpt = gainOpt(:, freqIndex);
139
140 gainMask = reshape(gainMask, [], 1);
141 gainMask = gainMask - compensatingOffset;
142
143 if nFreq == 1
144     E_p = squeeze(E_p);
145     E_pMaxGain          = E_p * beamformerOpt;
146     E_pMaxGainScalar    = max(abs(E_pMaxGain));
147     E_p                 = E_p ./ E_pMaxGainScalar;
148 else
149     for f = 1:nFreq
150         E_pMaxGain          = squeeze(E_p(:, :, f)) * beamformerTxRef;
151         E_pMaxGainScalar    = max(abs(E_pMaxGain));
152         E_p(:, :, f)        = E_p(:, :, f) ./ E_pMaxGainScalar;
153     end
154
155     E_p = permute(E_p, [1, 3, 2]);
156     E_p = reshape(E_p, [], size(E_p, 3));
157
158     oldGainMask = gainMask;
159     for i = 2:nFreq
160         gainMask(:, i) = oldGainMask;
161     end
162     gainMask = reshape(gainMask, [], 1);
163 end
164
165
166 maxGainScalar = zeros(size(EEPs, 1), nFreq);
167 for f = 1:nFreq
168     maxGainScalar(:, f) = ones(size(EEPs, 1), 1) * max(gainOpt(:, f));
169 end
170
171 maxGainScalar = reshape(maxGainScalar, [], 1);
172
173 E_pNorm = E_p;
174
175 %% determine the E-field mask in the cutting plane
176 E_mask    = 10 .^ ((gainMask - maxGainScalar) / 20);
177 for i = 1:size(E_mask, 1)
178     if isinf(E_mask(i))
179         E_mask(i) = 1;
180     end
181 end
182
183 %% Isolation beamformer calculation
184
185 comparitiveIso = zeros(nFreq, 1) + 10 ^ (-targetIso/ 10);
186 beamformerTxCompare = beamformerTxRef;
187
188 while loop
189     cvx_begin quiet
190         variable beamformerTxMaskIso(nChannels) complex
191         minimize( norm(beamformerTxMaskIso - beamformerTxCompare, 2) )
192         subject to
193             (abs((sRxTx * beamformerTxMaskIso).' * beamformerFreqExp)).'
                ↪ ...
194                 <= comparitiveIso;
195             abs(E_pNorm * beamformerTxMaskIso) <= E_mask;
196     cvx_end
197
198     beamformerTxMaskIso = beamformerTxMaskIso / norm(beamformerTxMaskIso,
        ↪ 2);
199     deltaV = abs(beamformerTxMaskIso - beamformerPrevious) ...
200              ./ abs(beamformerPrevious);
```

```
201
202     beamformerTxCompare = beamformerTxMaskIso;
203
204     [MW, loadIsoArray] = define_load_source_ports( ...
205         'MW', MW, ...
206         'sourcePortArray', portsLoadTx, ...
207         'excitationArray', beamformerTxMaskIso, ...
208         'Load', Load ...
209         );
210
211     % === START of MW-Engine ===
212     resultsMW = RunMWSimulator([transceiverSystem, antenna, loadIsoArray,
            ↪ loadArray(nChannels+1:2*nChannels)], MW);
213     % === End of MW-Engine ===
214
215     i = 1;
216     isoTxBF = zeros(nFreq, 1);
217     for f = 1:length(FREQ)
218         if any(FREQ(f) == F_RX_ARRAY)
219             iso = resultsMW.a(portsLoadRx, 1, f).' * beamformerRxRef;
220             isoTxBF(i) = abs(iso);
221             i = i + 1;
222         end
223     end
224
225     isoMiss = comparitiveIso ./ isoTxBF;
226
227     gainTxOvershoot = zeros(size(EEPs, 1), nFreq);
228     maskOvershoot = zeros(size(EEPs, 1), nFreq);
229     for i = 1:nFreq
230         gainTxOvershoot(:, i) = calculate_gain_tx( ...
231                 'EEPs', EEPs, ...
232                 'observationDistance', observationDistance, ...
233                 'Const', Const, ...
234                 'beamformer', beamformerTxMaskIso, ...
235                 'resultsMW', resultsMW, ...
236                 'freqIndex', find(FREQ(:) == F_ARRAY(i)), ...
237                 'Zo', Zo, ...
238                 'inputPorts', outputPorts, ...
239                 'outputPorts', outputPorts, ...
240                 'gainType', gainType ...
241                 );
242
243         gainMaskReshape = reshape(gainMask, [], nFreq);
244         for p = 1:length(gainTxOvershoot(:, i))
245             maskOvershoot(p, i) = gainTxOvershoot(p, i) - gainMaskReshape(p
                    ↪ , i);
246             if maskOvershoot(p, i) < 0
247                 maskOvershoot(p, i) = 0;
248             end
249         end
250     end
251     maskOvershoot = reshape(maskOvershoot, [], 1);
252
253     requirementsLogic = [isoMiss >= 1; maskOvershoot <= 0];
254     if requirementsLogic
255         loop = false;
256     elseif deltaV < convergenceDelta
257         loop = false;
258         if any(isoMiss < 1)
259             disp('Tx isolation failed')
260         end
261         if any(maskOvershoot > 0)
262             disp('Tx mask failed')
263         end
264         disp('Tx beamformer convergence reached')
265     else
266         beamformerPrevious = beamformerTxMaskIso;
267         maskOvershoot = 10 .^ (maskOvershoot / 20);
268         E_mask = E_mask ./ maskOvershoot;
269     end
270 end
271
272 end
```

D.2.12 calculate_max_gain_beamformer_weights.m

```
function MGB = calculate_max_gain_beamformer_weights( ...
    varargin)
% Calculate the maximum gain beamformer weights for given short-circuit
% embedded element patterns
%
% Args:
%   yAnt
%   EEPs_SC
%   thetaMesh
%   phiMesh
%   thetaScanRad
%   phiScanRad
%   observationDistance
%   freqIndex

%% input parsing

p = inputParser();
p.CaseSensitive = true;

p.addOptional('yAnt', []);
p.addOptional('EEPs_SC', []);
p.addOptional('thetaMesh', []);
p.addOptional('phiMesh', []);
p.addOptional('thetaScanRad', []);
p.addOptional('phiScanRad', []);
p.addOptional('observationDistance', []);
p.addOptional('freqIndex', []);
p.addOptional('polarisation', 'x')

p.parse(varargin{:});
yAnt                = p.Results.yAnt;
EEPs_SC             = p.Results.EEPs_SC;
thetaMesh           = p.Results.thetaMesh;
phiMesh             = p.Results.phiMesh;
thetaScanRad        = p.Results.thetaScanRad;
phiScanRad          = p.Results.phiScanRad;
observationDistance = p.Results.observationDistance;
freqIndex           = p.Results.freqIndex;
polarisation        = p.Results.polarisation;

%% Beamformer Calculation

EEPs_SC = EEPs_SC(:, :, :, freqIndex);
yAnt = yAnt(:, :, freqIndex);
scanLogic =  (thetaMesh(:)==thetaScanRad) & (phiMesh(:)==phiScanRad);
switch polarisation
        case 'x'
            EEPs = squeeze(EEPs_SC( ...
                scanLogic, 1, :)).';
        case 'y'
            EEPs = squeeze(EEPs_SC( ...
                scanLogic, 2, :)).';
        case 'z'
            EEPs = squeeze(EEPs_SC( ...
                scanLogic, 3, :)).';
        otherwise
        EEPs = squeeze(EEPs_SC(scanLogic, 1, :)).' + ...
            squeeze(EEPs_SC(scanLogic, 2, :)).' + ...
            squeeze(EEPs_SC(scanLogic, 3, :)).';
end

A           = 4 * pi * observationDistance^2 * (EEPs' * EEPs) / (120 * pi);
B           = real(yAnt);
[U,D]       = eig(A,B);
[~,index]   = max(diag(abs(D)));
beamformer  = U(:,index(1));
MGB         = beamformer / norm(beamformer,2);
end
```

D.2.13 calculate_max_sens_beamformer_weights.m

```
function beamformer = calculate_max_sens_beamformer_weights( ...
    varargin)
% Calculates the maximum sensitivity beamformer based on an incident plane
% wave. Beamformer => conjugate of signals at Rx Ports
%
% Args:
%   RxVoltages
%   thetaMesh
%   phiMesh
%   thetaScanRad
%   phiScanRad
%   freqIndex

%% input parsing

p = inputParser();
p.CaseSensitive = true;
p.addOptional('rxWaves', []);
p.addOptional('thetaMesh', []);
p.addOptional('phiMesh', []);
p.addOptional('thetaScanRad', []);
p.addOptional('phiScanRad', []);
p.addOptional('freqIndex', []);
p.addOptional('noiseWaveCorrMatrix', []);
p.addOptional('polarisation', '')
p.addOptional('sensType', 'CFM')

p.parse(varargin{:});
thetaMesh       = p.Results.thetaMesh;
phiMesh         = p.Results.phiMesh;
thetaScanRad    = p.Results.thetaScanRad;
phiScanRad      = p.Results.phiScanRad;
rxWaves         = p.Results.rxWaves;
freqIndex       = p.Results.freqIndex;
noiseWaveCorrMatrix            = p.Results.noiseWaveCorrMatrix;
polarisation    = p.Results.polarisation;
sensType        = p.Results.sensType;

%% Beamformer calculation

noiseWaveCorrMatrix = noiseWaveCorrMatrix(:, :, freqIndex);

scanLogic = phiMesh(:) == phiScanRad & thetaMesh(:) == thetaScanRad;
switch polarisation
    case 'y'
        e = squeeze(rxWaves(scanLogic, 2, :, freqIndex)).';
    case 'z'
        e = squeeze(rxWaves(scanLogic, 3, :, freqIndex)).';
    case 'x'
        e = squeeze(rxWaves(scanLogic, 1, :, freqIndex)).';
    otherwise
        e = squeeze(rxWaves(scanLogic, 1, :, freqIndex)).' + ...
            squeeze(rxWaves(scanLogic, 2, :, freqIndex)).' + ...
            squeeze(rxWaves(scanLogic, 3, :, freqIndex)).';
end

POI_matchedLoads = noiseWaveCorrMatrix;

switch sensType
    case 'CFM'
        beamformer = e;
    case 'SNR'
        beamformer = e * inv(POI_matchedLoads);
    otherwise
        beamformer = e;
end

beamformer = beamformer.' / norm(beamformer, 2);
end
```

D.2.14 calculate_opt_isolation_beamformer.m

```
function [beamformerRxIso, isolationMaxGainBF, isolationOptIso] = ...
```

```
2  calculate_opt_isolation_beamformer(varargin)
3  % Calculates the optimal isolation beamformer from the Rx port perspective
4  %
5  % Args:
6  %   FREQ
7  %   nChannels
8  %   nFreqPoints
9  %   resultsMGBTx
10 %   portsLoadRx
11 %   beamformerRxMaxSens
12 %   Zo
13
14 %% input parsing
15
16 p = inputParser();
17 p.CaseSensitive = true;
18 p.addOptional('FREQ', []);
19 p.addOptional('nChannels', []);
20 p.addOptional('resultsMGBTx', []);
21 p.addOptional('portsLoadRx', []);
22 p.addOptional('beamformerRxMaxSens', []);
23 p.addOptional('Zo', '')
24
25 p.parse(varargin{:});
26 FREQ                = p.Results.FREQ;
27 nChannels           = p.Results.nChannels;
28 resultsMGBTx        = p.Results.resultsMGBTx;
29 portsLoadRx         = p.Results.portsLoadRx;
30 beamformerRxMaxSens = p.Results.beamformerRxMaxSens;
31 Zo                  = p.Results.Zo;
32
33 %% Beamformer Calculation
34
35 vInput           = zeros(nChannels, length(FREQ));
36 pInput           = zeros(nChannels, length(FREQ));
37 beamformerRefExp = zeros(nChannels, length(FREQ), 1);
38
39 for f = length(FREQ)/2+1:length(FREQ)
40     [vInput(:, f), ~, pInput(:, f)] = abToVIP( ...
41         resultsMGBTx.a(portsLoadRx, 1, f),...
42         resultsMGBTx.b(portsLoadRx, 1, f), Zo);
            ↪                                  % Voltages and power at RX
            ↪ beamformer inputs
43     beamformerRefExp(:, f) = beamformerRxMaxSens;
44 end
45
46 vInput           = reshape(vInput, [], 1);
47 pInput           = reshape(pInput, [], 1);
48 beamformerRefExp = reshape(beamformerRefExp, [], 1);
49
50 vOutput = beamformerRefExp.' * vInput;
    ↪                                       % RX beam port output voltage
51 pOutput = abs(vOutput).^2 / (2*Zo);
    ↪                                        % RX beam port output power,
    ↪  also: Po = abs(Gain_LNA)^2*(w'*conj(Vi)*Vi.'*w)/(2*Zo);
52
53 gradient           = (conj(vInput) * vInput.' * beamformerRefExp) / (2*Zo);
54 optGamma           = 1 / (2*Zo) * beamformerRefExp' * conj(vInput) *...
55             vInput.' * conj(vInput) * vInput.' * beamformerRefExp / ...
56             ((1 / (2*Zo))^2 * beamformerRefExp' * conj(vInput)*...
57             vInput.' * conj(vInput) * vInput.' * conj(vInput) * vInput.' *
                   ↪ ...
58              beamformerRefExp);
59 beamformerRxIso    = beamformerRefExp - optGamma*gradient;
60 beamformerRxIso    = beamformerRxIso / norm(beamformerRxIso,2);
61 pOutputNew         = 1 / (2*Zo) * real(beamformerRxIso' * ...
62     conj(vInput) * vInput.' * beamformerRxIso);
63
64 pInput             = reshape(pInput, nChannels, length(FREQ));
65
66 isolationMaxGainBF = 1;
67 isolationOptIso    = 1;
68
69 end
```

D.2.15 calculate_Rx_loaded_EEPs.m

```
function [loadedEEPs, Voc] = ...
    calculate_Rx_loaded_EEPs(varargin )
% Calculate the embedded element patterns for a realistically loaded system
%
% Args:
%   nChannels   (int)
%   resultsMW   (struct)
%   EEPs_SC  (complex: nPhi x nTheta, [x, y, z], nChannels)
%   freqIndex   (int)
%   Const   (struct)
%   yAnt
%   MW
%   Load
%   transceiverSystem
%   RxPorts
%   Zo  (complex)
%   direction (string)
%   arrayElementPorts (int: nChannels)
%
% Outputs:
%   EEPs_loaded   (complex: nPhi x nTheta, [x, y, z], nChannels)
%   MWExcitation
%   EEPs_loadedCurrent   (complex: nPhi x nTheta, [x, y, z], nChannels)



p = inputParser();
p.CaseSensitive = true;

p.addOptional('EEPs_SC', []);
p.addOptional('freqIndex', []);
p.addOptional('Const', []);
p.addOptional('yAnt', []);
p.addOptional('MW', []);
p.addOptional('zSys', []);
p.addOptional('observationDistance', 2)

p.parse(varargin{:});
EEPs_SC             = p.Results.EEPs_SC;
freqIndex           = p.Results.freqIndex;
Const               = p.Results.Const;
yAnt                = p.Results.yAnt;
MW                  = p.Results.MW;
zSys                = p.Results.zSys;
observationDistance = p.Results.observationDistance;

for i = 1:size(yAnt, 3)
    zAnt(:, :, i) = inv(yAnt(:, :, i));
end

for p = 1:3

    if p == 1
        MW.IncidentPlaneWavePolarization = [1, 0, 0];
    elseif p==2
        MW.IncidentPlaneWavePolarization = [0, 1, 0];
    elseif p == 3
        MW.IncidentPlaneWavePolarization = [0, 0, 1];
    end

    Voc(:, p, :) = voc_from_eep_sc( ...
        'incidentEWave', MW.IncidentPlaneWavePolarization, ...
        'freqIndex', freqIndex, ...
        'Const', Const, ...
        'EEPs_SC', EEPs_SC, ...
        'zAnt', zAnt, ...
        'observationDistance', observationDistance);

    V_System = zSys(:, :, freqIndex) / (zSys(:, :, freqIndex) + ...
                zAnt(:, :, freqIndex)) * (squeeze(Voc(:, p, :))).';

    temp = inv(zSys(:, :, freqIndex)) * V_System * Const(:, freqIndex).
        ↪ Omega...
```

```
73          * Const(:, freqIndex).Mu / (4 * pi * 1j * observationDistance ...
74          * exp(1j * Const(:, freqIndex).k * observationDistance));
75      temp = temp.';
76      loadedEEPs(:, p, :) = temp;
77  end
```

D.2.16 calculate_rx_voltages.m

```
1   function [rxVoltages, rxWaves] = ...
2       calculate_rx_voltages(varargin )
3   % Calculate the voltages and currents at the Rx beamformer ports due to
4   % plane waves excitation from all possible theta phi angles
5   %
6   % Args:
7   %   nChannels
8   %   EEPs_SC
9   %   freqIndex
10  %   Const
11  %   yAnt
12  %   MW
13  %   arrayElementPorts
14  %   LoadRx
15  %   LoadTx
16  %   transceiverSystem
17  %   RxPorts
18  %   zSys
19  %   observationDistance
20  %   Mesh
21  %   Geom
22  %   thetaArray
23  %   phiArray
24  %   loadPorts
25
26  %% input parsing
27
28  p = inputParser();
29  p.CaseSensitive = true;
30
31  p.addOptional('nChannels', []);
32  p.addOptional('freqIndex', []);
33  p.addOptional('yAnt', []);
34  p.addOptional('MW', []);
35  p.addOptional('arrayElementPorts', []);
36  p.addOptional('LoadRx', []);
37  p.addOptional('LoadTx', []);
38  p.addOptional('transceiverSystem', []);
39  p.addOptional('RxPorts', []);
40  p.addOptional('thetaArray', []);
41  p.addOptional('phiArray', []);
42  p.addOptional('loadPorts', []);
43  p.addOptional('ZoAnt', '');
44  p.addOptional('planeWaveExcitations', []);
45  p.addOptional('sAnt', []);
46  p.addOptional('solverMethod', 's_params');
47
48  p.parse(varargin{:});
49  nChannels           = p.Results.nChannels;
50  freqIndex           = p.Results.freqIndex;
51  yAnt                = p.Results.yAnt;
52  MW                  = p.Results.MW;
53  arrayElementPorts   = p.Results.arrayElementPorts;
54  LoadRx              = p.Results.LoadRx;
55  LoadTx              = p.Results.LoadTx;
56  transceiverSystem   = p.Results.transceiverSystem;
57  RxPorts             = p.Results.RxPorts;
58  thetaArray          = p.Results.thetaArray;
59  phiArray            = p.Results.phiArray;
60  loadPorts           = p.Results.loadPorts;
61  ZoAnt               = p.Results.ZoAnt;
62  planeWaveExcitations= p.Results.planeWaveExcitations;
63  sAnt                = p.Results.sAnt;
64  solverMethod        = p.Results.solverMethod;
65
```

```
%% Rx beamformer output calculations

phiThetaSize = length(phiArray)*length(thetaArray);
rxVoltages  = zeros(phiThetaSize, 3, nChannels);
rxWaves     = zeros(phiThetaSize, 3, nChannels);
planeWaveExcitations = planeWaveExcitations(:, :, :, freqIndex);

sAnt = sAnt(:, :, freqIndex);
sSys = transceiverSystem.S(:, :, freqIndex);
I = eye(nChannels);

%% S-parameter approach
portsRxSp = 2*nChannels + 1:3*nChannels;
portsAntSp = nChannels+1 : 2*nChannels;

sSys21 = sSys(portsRxSp, portsAntSp);
sSys11 = sSys(portsAntSp, portsAntSp);

matB = sSys21 * (I + sAnt / (I - sSys11 * sAnt) * sSys11);

%% Calculate voltages at Rx beamformers due to excitations
for p = 1:3
    for q = 1:phiThetaSize
        inputSignal = (squeeze(planeWaveExcitations(q, p, :))).';

        switch solverMethod
            case 'mw'

                [MW, antennaMSB, loadMSBArray] = ...
                    define_antenna_source_ports(...
                    'MW', MW, ...
                    'nChannels', nChannels, ...
                    'sourcePortArray', arrayElementPorts, ...
                    'excitationArray', inputSignal, ...
                    'yAnt', yAnt, ...
                    'LoadRx', LoadRx, ...
                    'LoadTx', LoadTx, ...
                    'Zo', ZoAnt, ...
                    'oldLoadArray', loadPorts ...
                    );

                % === START of MW-Engine ===
                resultsMW_Rx = RunMWSimulator( ...
                    [transceiverSystem, antennaMSB, loadMSBArray], MW);
                % === End of MW-Engine ===

                [V_exct, ~, ~] = abToVIP( ...
                    resultsMW_Rx.a(RxPorts, 1, freqIndex), ...
                    resultsMW_Rx.b(RxPorts, 1, freqIndex), ...
                    LoadRx.Znorm);

                rxVoltages(q, p, :) = V_exct;
                rxWaves(q, p, :)   = resultsMW_Rx.b(RxPorts, 1, freqIndex);

            case 's_params'
                rxWaves(q, p, :) = matB * inputSignal.';
                rxVoltages(q, p, :) = 0;
            otherwise
                error('options are s_params or mw for rx waves')
        end
    end
end
```

D.2.17 calculate_system_gain_rx.m

```
function arrayGain = calculate_system_gain_rx(varargin)
% calculate the array gain when loaded with the microwave system and given
% beamformer
%
% Args:
%   RxVoltages
%   RxCurrents
%   beamformer
```

```
%   lambda
%   polarisation
%   freqIndex
%   yAnt

%% input parsing

    p = inputParser();
    p.CaseSensitive = false;

    p.addOptional('rxWaves', []);
    p.addOptional('beamformer', []);
    p.addOptional('lambda', []);
    p.addOptional('polarisation', '')
    p.addOptional('freqIndex', []);
    p.addOptional('yAnt', [])
    p.addOptional('zSys', [])

    p.parse(varargin{:});
    rxWaves          = p.Results.rxWaves;
    beamformer          = p.Results.beamformer;
    lambda              = p.Results.lambda;
    polarisation        = p.Results.polarisation;
    freqIndex           = p.Results.freqIndex;
    yAnt                = p.Results.yAnt;
    zSys                = p.Results.zSys;

%% Rx Gain calculation

    rxWaves  = rxWaves(:, :, :, freqIndex);
    yAnt        = yAnt(:, :, freqIndex);
    zSys        = zSys(:, :, freqIndex);
    lambda      = lambda(freqIndex);

    zAnt = diag(inv(yAnt));
    zSys = diag(zSys);
    inputRefl = (zAnt - zSys) ./ (zAnt + zSys);

    P_rec = zeros(size(rxWaves, 1), 3);

    for p = 1:3

        rxWaves_loop = squeeze(rxWaves(:, p, :));

        RxVoltagesReflScaled = zeros(size(rxWaves_loop));
        for i = 1:length(zAnt)
            RxVoltagesReflScaled(:, i) = rxWaves_loop(:, i) ./ (1 - (
                ↪ inputRefl(i) ^ 2));
        end


        for i = 1:size(RxVoltagesReflScaled, 1)
            P_rec(i, p)     = (beamformer' * ...
                               rxWaves_loop(i, :).' * ...
                               conj(rxWaves_loop(i, :)) * ...
                               beamformer) ...
                               ./ 2;
        end
    end

    eta       = 120 * pi;
    E_inc     = 1;
    P_recTotal = abs(P_rec(:, 1)) + abs(P_rec(:, 2)) + abs(P_rec(:, 3));

    G         = 8 * eta * pi * P_recTotal / abs(E_inc) ^ 2 / lambda ^ 2;

    arrayGain = 10*log10(abs(G));
end
```

D.2.18 calculate_system_gain_tx.m

```
function arrayGain = calculate_system_gain_tx(varargin)
    p = inputParser();
```

```
    p.CaseSensitive = false;

    p.addOptional('E_field',[]);
    p.addOptional('observationDistance',[]);
    p.addOptional('Const',[]);
    p.addOptional('beamformer',[]);
    p.addOptional('freqIndex',[]);
    p.addOptional('yAnt', [])

    p.parse(varargin{:});
    E_field             = p.Results.E_field;
    observationDistance = p.Results.observationDistance;
    Const               = p.Results.Const;
    beamformer          = p.Results.beamformer;
    freqIndex           = p.Results.freqIndex;
    yAnt                = p.Results.yAnt;

    E_field = E_field(:, :, :, freqIndex);
    Const   = Const(:, freqIndex);

    Etot = zeros(size(E_field,1),3);
    for n = 1 : size(beamformer,1)
        Etot = Etot + E_field(:,:,n)*beamformer(n);
    end

    yAnt = yAnt(:, :, freqIndex);
    sAnt = y2s(yAnt);

    G               = Etot*observationDistance*exp(1j*Const.k*
        ↪ observationDistance); % Far field function
    U               = (abs(G(:,1)).^2 + abs(G(:,2)).^2 + abs(G(:,3)).^2) /
        ↪ (2*120*pi);
    [~, ~, Pin]     = abToVIP(resultsMW.a(inputPorts, 1, freqIndex),
        ↪ resultsMW.b(inputPorts, 1, freqIndex), Zo);
    Pin             = real(1/2*beamformer'*real(yAnt')*beamformer);
    Uiso            = Pin/(4*pi);
    arrayGain       = 10*log10(U/Uiso);
end
```

D.2.19 calculate_system_impedance.m

```
function systemImpedanceArray = calculate_system_impedance(varargin)
% Calculate the system impedance as seen from the specified observation
% ports
%
% Args
%   MW
%   MW_system (microwave system without the observation ports / devices)
%   observationPorts
%   nChannels
%   Load

%% input parsing

    p = inputParser();
    p.CaseSensitive = false;
    p.addOptional('MW', []);
    p.addOptional('MW_system', []);
    p.addOptional('observationPorts', []);
    p.addOptional('nChannels', 1);
    p.addOptional('Load', []);

    p.parse(varargin{:});
    MW                  = p.Results.MW;
    MW_system           = p.Results.MW_system;
    observationPorts    = p.Results.observationPorts;
    nChannels           = p.Results.nChannels;
    Load                = p.Results.Load;

%% System impedance calculation

    % Assume all other ports are terminated in matched loads
    MW.ComputeOverallSparMatrix = true;
```

```
33
34      index = observationPorts(1);
35      for n = 1:nChannels
36          [obsLoadArray(n), ~] = ImpedanceDevice( ...
37              MW, 0, Load.Ta, Load.Znorm, index+n-1);
38          obsLoadArray(n).PortNumbering(2) = max(max(MW.
                ↪ PortConnectionTopology))+n;
39          MW.ExternalPorts(n) = max(max(MW.PortConnectionTopology))+n;
40      end
41
42      MW.CircuitSourcePorts    = [];
43      results                  = RunMWSimulator([MW_system, obsLoadArray],  MW
            ↪ );
44      systemImpedanceArray     = s2z(results.Snet);
45  end
```

D.2.20 calculate_system_s_params.m

```
1   function sNet = calculate_system_s_params(varargin)
2   p = inputParser();
3   p.CaseSensitive = false;
4
5   p.addOptional('deviceArray',[]);
6   p.addOptional('portConnectionTopology',[]);
7   p.addOptional('FREQ',[]);
8   p.addOptional('externalPorts',[]);
9
10  p.parse(varargin{:});
11  deviceArray              = p.Results.deviceArray;
12  portConnectionTopology   = p.Results.portConnectionTopology;
13  FREQ                     = p.Results.FREQ;
14  externalPorts            = p.Results.externalPorts;
15
16  MW.FreqArray = FREQ;
17
18  MW.IncidentPlaneWavePolarization    = [];
19  MW.Tsky                             = [];
20  MW.CircuitSourcePorts               = [];
21  MW.ComputeOverallSparMatrix         = true;
22  MW.ComputeOverallNoiseWaveMatrix    = false;
23
24
25  MW.ExternalPorts             = externalPorts;
26  MW.PortConnectionTopology    = portConnectionTopology;
27
28  Results = RunMWSimulator(deviceArray, MW);
29
30  sNet = Results.Snet;
31  end
```

D.2.21 calculate_Yant_and_element_pattern.m

```
1   function [yAnt, EEP_SC] = calculate_Yant_and_element_pattern(varargin)
2   % Calculates the admittance matrix and short circuit embedded element
3   % patterns for a given element and array geometry
4   %
5   % Args
6   %   Const
7   %   Mesh
8   %   Geom
9   %   thetaMesh
10  %   phiMesh
11  %   elementPatternParameters
12  %   observationDistance
13
14      %% input parsing
15
16      p = inputParser();
17      p.CaseSensitive = false;
18      p.addOptional('Const', []);
19      p.addOptional('Mesh', []);
```

```
20      p.addOptional('Geom', []);
21      p.addOptional('thetaMesh', 1);
22      p.addOptional('phiMesh', []);
23      p.addOptional('elementPatternParameters', []);
24      p.addOptional('observationDistance', []);
25
26      p.parse(varargin{:});
27      Const                       = p.Results.Const;
28      Mesh                        = p.Results.Mesh;
29      Geom                        = p.Results.Geom;
30      thetaMesh                   = p.Results.thetaMesh;
31      phiMesh                     = p.Results.phiMesh;
32      elementPatternParameters    = p.Results.elementPatternParameters;
33      observationDistance         = p.Results.observationDistance;
34
35      %% Solve for current at given frequency
36      [zMatrix,~] = MexBuildImpedanceMatrixBlock_v3(Const, Geom, Mesh, ...
37          char('L'), elementPatternParameters.sourceRWGs, char('J'), ...
38          elementPatternParameters.observationRWGs, 1, 0.0, false);
39
40      vMesh       = zeros(size(zMatrix,1), ...
41          size(elementPatternParameters.vExcitation, 2));
            ↪                       % Voltage per RWG
42
43      for n = 1 : size(elementPatternParameters.vExcitation,2)
44          vMesh(Geom.RWGPortIndex(n), n) = -Mesh.l(Geom.RWGPortIndex(n)) *
                ↪ ...
45              elementPatternParameters.vExcitation(n, n);
46      end
47      iMesh = zMatrix \ vMesh; % Current per RWG
48
49      %% Compute antenna input admittance matrix
50      yAnt = zeros(size(elementPatternParameters.vExcitation, 2));
51
52      for n = 1 : size(elementPatternParameters.vExcitation,2)
53          I_total = Mesh.l(Geom.RWGPortIndex).*iMesh(Geom.RWGPortIndex,n);
                ↪ % Total current through load
54          yAnt(:,n)  = I_total/elementPatternParameters.vExcitation(n,n);
55      end
56
57      %% Compute element patterns
58      elementPatternParameters.rObserv = [...
59          observationDistance * cos(phiMesh(:)) .* sin(thetaMesh(:)) ...
60          observationDistance * sin(phiMesh(:)) .* sin(thetaMesh(:)) ...
61          observationDistance * cos(thetaMesh(:))];
62
63      FieldTypeOut    = 'E';
64      SourceTypeIn    = 'J';
65      EEP_SC          = zeros(size(elementPatternParameters.rObserv, 1),
            ↪ 3,...
66          size(elementPatternParameters.vExcitation, 2));
67
68      for n = 1 : size(elementPatternParameters.vExcitation,2)
69          FieldOut        = NearField_MoMEngine (Const, Geom, Mesh, ...
70              elementPatternParameters.sourceRWGs, iMesh(:,n), SourceTypeIn,
                    ↪ ...
71              elementPatternParameters.rObserv, FieldTypeOut, 0.0, 1);
72
73          EEP_SC(:,:,n)    = FieldOut.E;
74      end
75  end
```