Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication

Tessio Perotti Arruda

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication

Téssio Perotti Arruda

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Prof. Sébastien Roland Marie Joseph Rondineau, D.Sc. (ENE-UnB)
(Orientador)

Prof. Renato Alves Borges, PhD (ENE-UnB)
(Examinador Interno)

Prof. Allan Kardec Barros, PhD (UFMA)
(Examinador Externo)

Prof. Daniel Costa Araujo, PhD (ENE-UnB)
(Suplente)

Himilcon Carvalho, PhD (Visiona)
(Membro Convidado)

Brasília/DF, 31 de maio de 2024.

FICHA CATALOGRÁFICA

ARRUDA, TÉSSIO PEROTTI

Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication

[Distrito Federal] 2024.

xv, 138p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado, 2024).

Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica.

Departamento de Engenharia Elétrica

1. Antenna Arrays
2. Satellite Communication
3. Ground Station Distribution
4. Convex Optimization
5. Differential Evolution
6. Link Budget

I. ENE/FT/UnB

II. Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication

REFERÊNCIA BIBLIOGRÁFICA

ARRUDA, TÉSSIO PEROTTI (2024). Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGEE 812/24. Maio/2024, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 138p.

CESSÃO DE DIREITOS

AUTOR: Téssio Perotti Arruda

TÍTULO: Optimizing 3D Antenna Arrays and Ground Station Distribution for Satellite Communication.

GRAU: Mestre   ANO: 2024

Téssio Perotti Arruda

Universidade de Brasília (UnB)

Campus Darcy Ribeiro

Faculdade de Tecnologia - FT

Departamento de Eng. Elétrica (ENE)

Brasília - DF CEP 70919-970

Ad maiorem Dei gloriam.

Acknowledgements

Firstly, I thank God for all the given graces and inspiration during this work.

I also thank my beloved wife, Débora Cristina Perotti, for your love and patience. Without your support, it would not be possible to conclude this work.

Thanks to my mentor, Sébastien Rondineau, for your advice and guidance, which were essential to this work's development. Thanks also to my friend, Rafael Luz, for all the discussions and valuable insights.

At last, thanks to the Brazilian Air Force for the opportunity to develop this work and to CNPq, for supporting the project "Low-Cost and High-Download-Rate Autonomous Distributed Ground Station".

Abstract

This work investigates the design and optimization of 3D antenna arrays and ground station distribution for satellite communication systems. It has the objective of proposing a ground station distribution that maximizes the contact and downlink data with an LEO satellite over a given territory.

The dissertation begins with a theoretical foundation, discussing antenna types such as parabolic reflectors and microstrip antennas. It discusses radiation fields from apertures and directivity from electric fields. It also discusses the link budget evaluation in the context of satellite communication. Finally, it presents the optimization algorithms that will be used in the work. All the antenna field simulation and link budget evaluation are realized through a developed Python package, *arraytools*.

The work continues by proposing a multi-step process to distribute ground stations inside a given territory to maximize the link coverage regarding a given satellite. The process consists of three concatenated algorithms: convex optimization, sequential least squares and differential evolution. Each one has its disadvantages, which are compensated by the following algorithms. It also considers an alternative scenario, in which the ground stations must be placed into some specific locations due to legacy infrastructure. In this case, the optimization variables are the antenna parameters instead of the positions. These algorithm implementations are also done by *arraytools*.

The next chapter then discusses a method to design a static array that can maintain an approximately constant power level during satellite passes. The difference between the proposed algorithm and the more conventional ones is that the antennas are static, physically steered and can be placed in any 3D position. The developed package also includes all the functions necessary to combine the fields of a single element into an array.

Finally, it presents the overall results and considerations for future works. First, it was possible to successfully propose a ground station distribution that maximizes the link coverage over the Brazilian territory considering two different LEO satellites. Then, it was proposed a ground station distribution that matches the downlink capability of a ground station positioned near the South Pole. At last, it was proposed static arrays that can maintain constant power levels during satellite passes.

Keywords: Antenna Arrays, Satellite Communication, Ground Station Distribution, Convex Optimization, Differential Evolution, Link Budget.

Resumo Estendido

Título: Otimização de Arranjos de Antenas 3D e Distribuicão de Estações de Solo para Comunicações via Satélite.

Este trabalho tem o objetivo de propor uma distribuição de estações de solo que maximizem o contato e o downlink com um satélite de órbita baixa em um dado território.

Ele está inserido no contexto do projeto “Estação Terrena Autônoma Distribuída de Baixo Custo e Alta Taxa de Download”, aprovado na chamada CNPq/AEB/MCTI/FNDCT Nº 20/2022 do programa UNIESPAÇO, cujo objetivo é determinar a viabilidade de uma estação terrestre programável remotamente, idealmente sem partes móveis, em regiões não polares, e com uma meta de custo inferios a um décimo do custo das estações terrestres contemporâneas para construir e manter. Esse tipo de estudo é pouco explorado com aplicações ao território brasileiro, que foi o escolhido como cenário das análises deste trabalho, embora as ferramentas desenvolvidas possuam aplicação para qualquer região de interesse. Ainda com a ideia de explorar aplicações nacionais, um dos satélites que foi utilizado neste trabalho foi o VCUB1, o primeiro satélite de observação da Terra que foi projetado pela indústria nacional.

Para atingir essa finalidade, uma ferramenta em python foi desenvolvida com as seguintes funcionalidades: simular antenas parabólicas e de microfita, realizar a rotação de campos distantes de antenas fisicamente rotacionadas, combinar várias antenas fisicamente rotacionadas e posicionadas de forma não uniforme em um arranjo, avaliar o link budget de um enlace de comunicação satelital, distribuir estações de solo através de um dado território de forma a maximizar a cobertura, maximizar a cobertura de estações de solo restritas a posições fixas em um território e projetar um array sem partes móveis capaz de manter um nível constante de potência em relação a um passe de satélite de órbita baixa.

O trabalho inicia com um capítulo relacionado com a fundamentação teórica do problema, em que são apresentadas as equações que foram implementadas nos módulos do pacote em python. As necessidades de propagação de órbitas de satélites e cálculos de acesso foram

supridas por meio do software SMET, também desenvolvido no contexto do projeto. Um dos objetivos é integrar as ferramentas desenvolvidas neste trabalho com esse software já existente. Dessa forma, ao fim do projeto, espera-se entregar um software completo para realizar análises envolvendo enlaces de comunicação satelitais. Nesse capítulo, o satélite argentino SAC-C foi utilizado como benchmark das ferramentas desenvolvidas, por já ter sido alvo de estudos de link budget e possuir dados disponíveis.

A seguir, apresenta a solução do problema de distribuição de estações de solo para maximizar a cobertura de um dado satélite de órbita baixa. Para isso, foi sugerido um processo em três etapas utilizando diferentes técnicas de otimização, com cada etapa resolvendo deficiências das etapas passadas. O primeiro passo envolveu um relaxamento do problema para que ele fosse capaz de ser modelado de forma convexa, restringindo as estações de solo a um grid sob o território brasileiro. O resultado dessa otimização convexa foi, então, usado como entrada para outra abordagem desconsiderando as restrições das estações de solo a um grid. No entanto, considerava uma aproximação para o padrão de cobertura das antenas, considerando-as circulares em um grid de latitude e longitude e também não restringia as antenas a estarem dentro do território de interesse. Por fim, esse resultado foi usado como entrada de um algoritmo genético de evolução diferencial, que foi capaz de incluir todas as restrições desejadas. Cada etapa refinou o resultado inicialmente encontrado pela otimização convexa, que se mostrou eficaz mesmo com o relaxamento do problema.

Também foi apresentada uma modificação do problema, considerando as estações de solo restritas a locais específicos do país, em que haja infraestrutura suficiente para facilitar a manutenção dessas estações. Nesse caso, as variáveis a serem otimizadas foram os parâmetros das antenas utilizadas, como o diâmetro do refletor parabólico. O resultado encontrado, então, foi comparado com uma estação posicionada próximo ao polo sul, obtendo capacidades semelhantes de download.

Finalmente, foi proposta uma otimização de forma a obter um arranjo de antenas sem partes móveis capaz de manter um nível de potência aproximadamente constante ao longo da trajetória de um satélite de órbita baixa. No entanto, embora esse objetivo tenha sido alcançado, o arranjo ainda apresenta um grande espalhamento de energia, dificultando a capacidade de fechar enlaces de comunicação satelitais. Nesse cenário do cálculo de arranjo de antenas, foi integrada outra ferramenta desenvolvida no contexto do projeto, chamada AFTK. Esse módulo é capaz de

reconstruir os campos de uma antena em qualquer ponto a partir de uma amostra dos campos utilizando modos esféricos.

Em resumo, foi possível encontrar um procedimento que é capaz de maximizar a área coberta por estações de solo considerando os passes de um satélite, além de também propor uma solução de estações de solo dentro do território nacional com capacidades de downlink similares a uma estação próxima ao Polo Sul e também propor um arranjo sem partes móveis capaz de manter um nível de potência aproximadamente constante durante um passe de satélite. Isso foi alcançado com o desenvolvimento de ferramentas que foram integradas a outros softwares desenvolvidos no contexto do projeto CNPq.

Palavras-Chave: Arranjo de Antenas, Comunicação Satelital, Distribuição de Estações de Solo, Otimização Convexa, Evolução Diferencial, Link Budget.

Contents

VI Appendices 70

List of Figures

List of Tables

List of Symbols

Antenna

D	Antenna parabolic reflector effective diameter.
U	Antenna Radiation intensity.
P_{rad}	Antenna Radiated Power.
$D(\theta, \phi)$	Antenna Directivity in (θ, ϕ) direction.
F	Antenna parabolic reflector focal length.
ψ	Antenna parabolic reflector aperture angle.
ψ_0	Antenna parabolic reflector maximum aperture angle.
G	Antenna gain in dB.
g	Antenna gain.
A_h	Horn x dimension.
B_h	Horn y dimension.
L	Microstrip patch x length.
W	Microstrip patch y length.

Electromagnetism

$\boldsymbol{H}_a$	Aperture magnetic field.
E_ϕ	Component of electric field in spherical coordinates $\hat{\phi}$.
e_a	Aperture efficiency.
$\boldsymbol{f}(\theta, \phi)$	Fourier transform of $\boldsymbol{E}_a$.
f	Carrier frequency.
$\boldsymbol{E}_a$	Aperture electric field.
η_0	Impedance of free space.
λ	Carrier wavelength.
ϵ	Dielectric permittivity.
ϵ_r	Dielectric relative permittivity.

$\boldsymbol{k}$	Wavenumber $= \frac{2\pi}{\lambda}$.
c_0	Speed of light in free space.
E_θ	Component of electric field in spherical coordinates $\hat{\theta}$.

Ground Station Distribution

M	Size of longitude grid.
P	Size of latitude grid.
N_{grid}	Number of possible positions to distribute the ground stations. $N_{\text{grid}} = MP$.
a_i	Binary matrix representing in which points inside the $M \times P$ grid it is possible to establish a link. This $\mathbb{R}^{M\times P}$ matrix is parsed into a vector $\mathbb{R}^{MP}$
A	Binary matrix representing in which points inside the $M \times P$ grid it is possible to establish a link for each N_{grid} possible ground stations. The i^{th} column represents the ground station in position i.
S_{best}	Maximum potential coverage area considering all Q ground stations.
S_{cov}	Coverage area considering all Q ground stations.
$S_{\cup}$	Coverage area considering all Q ground stations for the scenario considering antennas with different parameters.
$S_{\cap}$	Intersection areas considering all Q ground stations.

Link Budget

p_{out}	Power emitted by satellite.
B	Communication link bandwidth.
k	Boltzmann constant.
R	Communication link data rate.
R_{spec}	Communication link specified data rate.
E_b/N_0	Energy Bit per Noise ratio.
P_e	Bit error probability.
P_{EIRP}	Transmitted EIRP power in dB.
G_{sat}	Satellite antenna gain in dB.
P_{out}	Power emitted by satellite in dB.
g_{sat}	Satellite antenna gain.
p_{EIRP}	Transmitted EIRP power.
g_{f_u}	Free space losses.
G_{f_u}	Free space losses in dB.

P_{rec} Received power in dB.

G_{ground} Ground station gain amplification in dB.

SNR Signal to Noise Ratio.

T_{sys} System noise temperature.

Otimizations

Q Number of employed ground stations.

$D_{\min}$ Minimum parabolic diameter for the parabolic reflectors with different diameters minimization algorithm.

$D_{\max}$ Maximum parabolic diameter for the parabolic reflectors with different diameters minimization algorithm.

$d_{\min}$ Minimum acceptable distance between two elements in an array.

$\epsilon_u, \epsilon_E, \epsilon_d, \delta$ Maximum acceptable error.

S_d Percentage of territory that is covered for Sequential Least Squares.

F_α, F_β Control variable for Differential Evolution. Controls the mutation process.

C_1, C_2 Control variables for the parabolic reflectors with different diameters minimization algorithm.

C_R Control variable for Differential Evolution. Controls the crossover process.

N_P Size of population for Differential Evolution.

G Number of generations for Differential Evolution.

f_d Desired pattern of the optimization.

Arrays

N Number of elements in array.

α_i Steer angle around local x-axis of the i^{th} element.

β_i Steer angle around local y-axis of the i^{th} element.

γ_i Steer angle around local z-axis of the i^{th} element.

d_i Cartesian position of the i^{th} element.

$\boldsymbol{d}$ Matrix with all elements Cartesian positions. The i^{th} column represents the i^{th} element positions (x_i, y_i, z_i).

$\boldsymbol{\psi}$ Matrix with all elements rotation. The i^{th} column represents the i^{th} element rotations $(\alpha_i, \beta_i, \gamma_i)$.

$\boldsymbol{x}$ Matrix with all elements to be optimized in an array. It can represent the rotations and input for all array elements or the positions and rotations for all array elements.

Miscellaneous

$\hat{\boldsymbol{n}}$	Unit normal vector.
r	Radial distance from spherical coordinates.
θ	Polar angle from spherical coordinates.
ϕ	Azimuthal angle from spherical coordinates.
ρ	Radial distance from cylindrical coordinates.
$\boldsymbol{w}$	Quadrature weights for evaluating integrals.
$l_{\max}$	Maximum degree of considered spherical harmonics.

Glossary

AFTK Antenna Fields Tool Kit. viii, 4, 29, 50, 52, 57, 58, 68

BFGS Broyden–Fletcher–Goldfarb–Shanno is a interative method for solving unconstrained nonlinear optimization problems. viii, 55–57

CVX Convex Optimization. vii, x, 34, 36–42

DE Differential Evolution. 35, 37, 38, 41

EIRP Effective Isotropic Radiated Power. 21, 23

HFSS High-Frequency Structure Simulator. vi–viii, 16, 19, 20, 49–52

LEO Leo Earth Orbit. 2–4

SGP4 Simplified General Perturbations 4. 20
SMET Space Mission Engineering Tools. 4, 20
SQLQ Sequential Least Squares Optimization. x, 34, 35, 37, 41, 43, 67
SSO Sun Synchronous Orbit. 2
STK System Tools Kit. 20, 69

Part I

Introduction

Chapter 1

Introduction

1.1 State-of-the-art

A typical state-of-the-art ground station for Low Earth Orbit (LEO) satellites uses a single large 11-meter parabolic antenna and tracks a single satellite at a time by mechanically sweeping the antenna up to 160 degrees. The downlink supports data rates ranging from 2 kbps to 150 Mbps. To maximize contact with Sun-Synchronous Orbit (SSO) satellites, ground stations are ideally located near the poles. These ground stations cost around $4 million each to build and have high maintenance costs, like the antenna located in Poker Flats. As said in [9], a static 1m dish parabolic reflector would cost around $5000. In this way, an array with elements like this would be significantly cheaper than the huge state-of-the-art parabolic antennas.

Maintaining consistent and uninterrupted contact with LEO satellites over a specific territory brings several advantages and benefits: getting more opportunities to acquire real-time telemetry, to send commands, to downlink data or to establish efficient and robust communication links in case of communication payloads. In this context, it is important to place ground stations in strategic positions, particularly when dealing with huge territories.

The problem of base station placement for maximizing coverage is usually tackled by fields like mobile communications and unmanned aerial vehicles [6] [21] [16].

To the best of our knowledge, there is no significant work dealing with ground station placement for satellite coverage optimization.

1.2 Contextualization

This work is supported by the project "Low-Cost and High-Download-Rate Autonomous Distributed Ground Station" approved in the CNPq/AEB/MCTI/FNDCT Call No. 20/2022, UNIESPAÇO Program.

The project objective is to determine the feasibility of a remotely programmable ground station, ideally without moving parts, in non-polar regions, with a cost target of less than one-tenth of the cost of contemporary ground stations to build and maintain. Instead of a single dish, the ground system would consist of a number of antenna arrays with small to moderate aperture sizes and the array outputs would be adaptively combined to maximize the signal-to-interference-and-noise ratio of the desired satellite transmission. The main focus of the project

Figure 1.1: UnB Telecommunications Laboratory - LCEPT [23].

is on the physical layer: the radio-frequency front end and the digital signal processing of the antenna array outputs.

This ground station will not support downlink speed data as the current state-of-the-art large dishes. However, as more ground stations are deployed, data could be downloaded in a distributed manner as the satellite passes through a series of ground stations. Ideally, the ground stations are connected via the internet, allowing any LEO satellite to be in almost continuous communication with the Earth network. While the current project focuses on the ground station to communicate with only one satellite at a time, the studied architecture is capable of rapid and electronically controlled reconfiguration to enable quick switching from one satellite to another within the same constellation or communication with multiple satellites. The UnB Telecommunications Laboratory [23] is involved in this project and already has an initial antenna site, as shown in Figure 1.1.

To accomplish the ground station design, it is necessary to develop a software capable of analyzing all steps required for a satellite link budget evaluation, like:

- Propagating a satellite orbit;
- Modeling an antenna or an antenna array;
- Embedding an antenna in a ground station and in a satellite;
- Analyzing the link budget and downlink data of a satellite; and other functions.

1.3 Goals

1.3.1 Main Goal

The main purpose of this work is to propose a low-cost ground station distribution that maximizes the contact and downlink data with a LEO satellite over a given territory.

1.3.2 Specific Goals

The specific goals of this work are:

- To propose array project methods of combining non-uniform 3D distributed elements physically steered;
- To propose a method to optimally distribute ground stations maximizing the coverage over a given territory;
- To evaluate the download capabilities of antennas placed inside a given territory in comparison with ground stations located near the poles; and
- To propose a low-cost static antenna array that can maintain a constant power level while in contact with a LEO satellite.

1.4 Work Contributions

The proposed goals are accomplished via the development of a python module named *arraytools* that is integrated with two other tools that are being developed in the CNPq project context, called SMET and AFTK.

The capabilities of *arraytools* package are:

- To simulate Parabolic Reflector and Micro Strip antennas;
- To perform far-field rotation after physically steering an antenna;
- To be able to perform 3D array far-field calculation, including array elements that are physically rotated;
- To evaluate satellite link budget in relation to a given ground station;
- To distribute ground stations over a given territory maximizing the covered area while minimizing the intersections;
- To maximize the covered area by ground stations that are constrained to be in fixed positions in a territory; and
- To project a fixed position array that can maintain a constant power level in a given solid angle of view.

1.5 Organization

This section provides an overview of the structure of this work, highlighting the key topics covered in each chapter and how they are related to the overall research objectives.

Chapter 2 presents a review of the theoretical principles underlying antenna design, array design and satellite link budget evaluation. This chapter discusses two types of antennas, parabolic reflectors and microstrip antennas, and explain their field radiation characteristics.

It also explores far-field pattern rotation and gives a brief explanation about optimization algorithms that are used.

Chapter 3 initially focuses on the distribution of ground stations maximizing the coverage of Brazil and explores various optimization approaches with its mathematical modeling. It is proposed the ground station distribution over Brazil considering two different satellites. It also considers a scenario were the ground station locations are constrained to be in specific points inside Brazil, while considering antennas with variable parameters and proposes a ground station distribution that maximizes the satellite coverage and compares its download performance against antennas positioned in the poles.

Chapter 4 explores the design of antenna arrays, proposing a model and validating with HFSS simulations. Additionally, it also proposes array designs that are capable of maintaining a constant power level in the direction of a satellite path.

Chapter 5 concludes this work by presenting commentaries on all obtained results.

Additionally, Appendix A contains additional technical details regarding traditional array design methods and Appendix B provides some scripts used in the production of this work results.

Part II

Theoretical Foundation

Chapter 2

Theoretical Foundation

This chapter presents the theoretical foundation regarding the used antennas, the link budget calculation focused in satellite links and also presents the optimization algorithms used in this work.

2.1 Antennas

This section briefly introduces the theory of antenna design, with a focus on satellite links. It starts with a explanation of the equations of radiation fields. Then it introduces some types of antennas, such as parabolic and microstrip. The first one is the most common antenna used in satellite communications. The last has a mathematical model simpler than other types of antennas and presents some advantages regarding size and manufacturing.

2.1.1 Radiation Fields from Apertures

$\boldsymbol{H}_a, \boldsymbol{E}_a$ are respectively the magnetic and electric aperture fields and that the Huygens source condition is valid, that is, $\mathbf{H}_a = \dfrac{1}{\eta_0}\,\hat{\mathbf{n}} \times \mathbf{E}_a$, at all points on the aperture, where η_0 is the free wave impedance.

The radiation field at some large distance r in the direction defined by the polar angles θ, ϕ are [15]:

$$\begin{aligned} E_\theta &= jk\frac{e^{-jkr}}{2\pi r}\frac{1+\cos\theta}{2}\left[f_x\cos\phi + f_y\sin\phi\right], \text{ and} \\ E_\phi &= jk\frac{e^{-jkr}}{2\pi r}\frac{1+\cos\theta}{2}\left[f_y\cos\phi - f_x\sin\phi\right], \end{aligned} \tag{2.1}$$

where the vector $\mathbf{f} = \hat{\mathbf{x}}f_x + \hat{\mathbf{y}}f_y$ is the Fourier transform over the aperture:

$$\mathbf{f}(\theta,\phi) = \int_0^a \int_0^{2\pi} \mathbf{E}_a(\rho',\chi)e^{j\mathbf{k}\cdot\mathbf{r}'}\rho' d\rho' d\chi. \tag{2.2}$$

2.1.2 Parabolic Reflector Antenna

Reflector antennas have very high gains and narrow main beams. They are widely used in satellite communications. A typical parabolic reflector, fed by a horn antenna positioned at the focus is shown in Figure 2.1 [15].

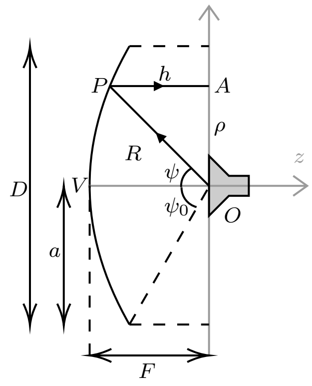

Figure 2.1: Parabolic reflector antenna with horn feed at the focus. $D = 2a$ is the reflector effective diameter, F is the focal length and ψ_0 is the maximum aperture angle.

The total optical path length from the focus to the aperture plane is constant, independent of ψ, and given by:

$$R + h = 2F. \tag{2.3}$$

From the geometry, it is possible to say:

$$h = R\cos\psi, \tag{2.4}$$

$$R + R\cos\psi = 2F, \text{ and} \tag{2.5}$$

$$R = \frac{2F}{1+\cos\psi}. \tag{2.6}$$

Also, the radial displacement ρ of the reflected ray on the aperture plane is given by $\rho = R\sin\psi$. Therefore:

$$\rho = 2F\frac{\sin\psi}{1+\cos\psi} = 2F\tan\left(\frac{\psi}{2}\right). \tag{2.7}$$

If $\rho = a = {}^{D}/_{2}$:

$$a = \frac{D}{2} = 2F\tan\left(\frac{\psi_0}{2}\right), \text{ and} \tag{2.8}$$

$$\psi_0 = 2\arctan\left(\frac{D}{4F}\right). \tag{2.9}$$

Gain of Reflector Antennas

From [15], the gain of a parabolic antenna can be summarized as:

$$g_{\max} = e_a\left(\frac{\pi D}{\lambda}\right)^2, \tag{2.10}$$

where the aperture efficiency e_a of practical parabolic reflectors is typically of the order of 0.55 - 0.65, λ is the wavelength and D is the parabolic reflector diameter.

Radiation Patterns of Reflector Antennas

The radiation patterns of the reflector antenna can be obtained from the aperture fields $\mathbf{E}_a, \mathbf{H}_a$ integrated over the effective aperture [15].

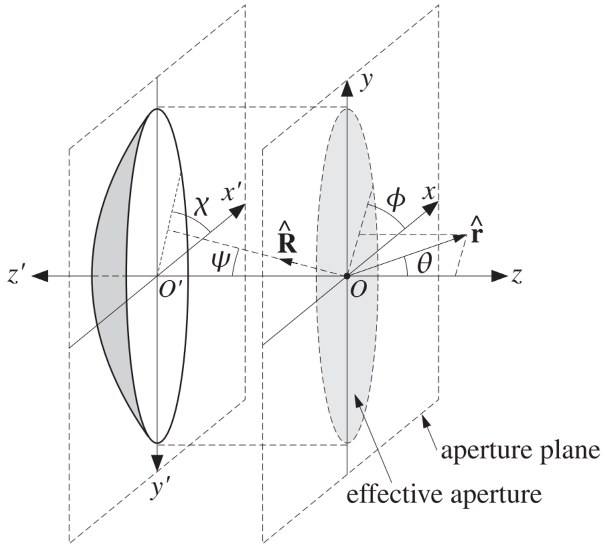

Figure 2.2: Projected effective aperture of parabolic antenna [15].

The vector $\mathbf{r}'$ lies on the aperture plane and is given in cylindrical coordinates by:

$$\mathbf{r}' = \rho'\hat{\rho} = \rho'\left(\hat{\mathbf{x}}\cos\chi + \hat{\mathbf{y}}\cos\chi\right). \tag{2.11}$$

Therefore, using Eq. (2.2), the Fourier transform over the aperture becomes

$$\mathbf{f}(\theta,\phi) = \int_0^a \int_0^{2\pi} \mathbf{E}_a(\rho,\chi) e^{jk\rho\sin\theta\cos(\phi-\chi)} \rho d\rho d\chi. \tag{2.12}$$

It is possible to convert this into an integral over the feed angles ψ, χ by using the following equations and $d\rho = Rd\psi, \rho = 2F\tan(\psi/2)$:

$$\mathbf{E}_a = \frac{e^{-2jkF}}{R}\mathbf{f}_a(\psi,\chi). \tag{2.13}$$

Therefore, (2.12) becomes:

$$\mathbf{f}(\theta,\phi) = 2Fe^{-2jkF} \int_0^{\psi_0} \int_0^{2\pi} \mathbf{f}_a(\psi,\chi) e^{2jkF\tan\frac{\psi}{2}\sin\theta\cos(\phi-\chi)} \tan\frac{\psi}{2} d\psi d\chi. \tag{2.14}$$

Given a feed pattern $\mathbf{f}_i(\psi,\chi)$, the aperture pattern $\mathbf{f}_a(\psi,\chi)$ is determined by:

$$\mathbf{f}_a = -\mathbf{f}_i + 2\hat{\mathbf{n}}(\hat{\mathbf{n}}\cdot\mathbf{f}_i). \tag{2.15}$$

From $\mathbf{f}_i(\psi,\chi)$, it is possible to solve the Eq. (2.14) numerically. As consequence of the condition $\hat{\mathbf{R}}\cdot\mathbf{f}_i = 0$, the vector $\mathbf{f}_i$ will only have components along $\hat{\psi}$ and $\hat{\chi}$:

$$\mathbf{f}_i = \hat{\psi}F_1\sin\chi + \hat{\chi}F_2\cos\chi, \tag{2.16}$$

where F_1, F_2 are functions of ψ, χ.

Such feeds are referred to as y-polarized. The x-polarized case is obtained by a rotation, replacing χ by $\chi + 90°$.

From Eq. (2.15):

$$\mathbf{f}_a = -\hat{\mathbf{y}}[F_1 \sin^2\chi + F_2 \cos^2\chi] - \hat{\mathbf{x}}[(F_1 - F_2)\cos\chi \sin\chi]. \tag{2.17}$$

The feed pattern is:

$$\mathbf{f}_i(\psi, \chi) = F_h(\psi, \chi)\left(\hat{\psi}\sin\chi + \hat{\chi}\cos\chi\right), \tag{2.18}$$

where:

$$F_h(\psi, \chi) = -\frac{jABE_0}{8\lambda}(1 + cos\psi)F_1(\nu_x, \sigma_a)F_0(\nu_y, \sigma_b), \tag{2.19}$$

and $\nu_x = (A_h/\lambda)\sin\psi\cos\chi$, $\nu_y = (B_h/\lambda)\sin\psi\sin\chi$, A_h, B_h are the horn dimensions, σ_a, σ_b are related to the maximum phase deviations in cycles. The horn pattern functions are:

$$F_0(\nu, \sigma) = \frac{1}{\sigma}e^{j(\pi/2)(\nu^2/\sigma^2)}\left[F\left(\frac{\nu}{\sigma} + \sigma\right) - F\left(\frac{\nu}{\sigma} - \sigma\right)\right], \text{ and} \tag{2.20}$$

$$F_1(\nu, \sigma) = \frac{1}{2}\left[F_0\left(\nu + 0.5, \sigma\right) + F_0\left(\nu - 0.5, \sigma\right)\right], \tag{2.21}$$

where $F(x) = C(x) - jS(x)$ is the standard Fresnel integration.

The corresponding aperture pattern is

$$\mathbf{f}_a = -\hat{\mathbf{y}}F_h(\psi, \chi). \tag{2.22}$$

In the general case, a more convenient form of (2.17) is obtained by writing it in terms of the sum and difference patterns:

$$A = \frac{F_1 + F_2}{2}, \quad B = \frac{F_1 - F_2}{2} \iff F_1 = A + B, \quad F_2 = A - B. \tag{2.23}$$

So, (2.17) becomes:

$$\mathbf{f}_a = -\hat{\mathbf{y}}\left(A - B\cos 2\chi\right) - \hat{\mathbf{x}}\left(B\sin 2\chi\right). \tag{2.24}$$

In general, A and B will be functions of ψ, χ. Assuming that they are functions only of ψ, then the χ-integration in the radiation pattern (2.14) can be done explicitly leaving an integral over ψ only. Using (2.24) and the Bessel-function identities, with $J_n(u)$ denoting the Bessel functions of the first kind of order n,

$$\int_0^{2\pi} e^{ju\cos(\phi-\chi)}\begin{bmatrix}\cos n\chi \\ \sin n\chi\end{bmatrix} d\chi = 2\pi j^n \begin{bmatrix}\cos n\phi \\ \sin n\phi\end{bmatrix} J_n(u), \tag{2.25}$$

it is obtained:

$$\mathbf{f}(\theta,\phi) = -\hat{\mathbf{y}}\left[f_A(\theta) - f_B(\theta)\cos 2\phi\right] - \hat{\mathbf{x}}\left[f_B(\theta)\sin 2\phi\right], \tag{2.26}$$

where the functions $f_A(\theta)$ and $f_B(\theta)$ are:

$$\begin{aligned} f_A(\theta) &= \quad 4\pi F e^{-2jkF}\int_0^{\psi_0} A(\psi) J_0\left(\frac{4\pi F}{\lambda}\tan\frac{\psi}{2}\sin\theta\right)\tan\frac{\psi}{2}d\psi, \text{ and} \\ f_B(\theta) &= -4\pi F e^{-2jkF}\int_0^{\psi_0} B(\psi) J_2\left(\frac{4\pi F}{\lambda}\tan\frac{\psi}{2}\sin\theta\right)\tan\frac{\psi}{2}d\psi. \end{aligned} \tag{2.27}$$

Using (2.24) and some trigonometric identities, the radiation fields (2.1) become:

$$\begin{aligned} E_\theta &= -j\frac{e^{-jkr}}{\lambda r}\frac{1+\cos\theta}{2}\left[f_A(\theta) + f_B(\theta)\right]\sin\phi, \text{ and} \\ E_\phi &= -j\frac{e^{-jkr}}{\lambda r}\frac{1+\cos\theta}{2}\left[f_A(\theta) - f_B(\theta)\right]\cos\phi. \end{aligned} \tag{2.28}$$

In the considered scenario, for the parabolic reflector, $B(\psi) = 0$. Therefore, $f_B(\theta) = 0$ and the electric field equations are reduced to:

$$\begin{aligned} E_\theta &= -j\frac{e^{-jkr}}{\lambda r}\frac{1+\cos\theta}{2}f_A(\theta)\sin\phi, \text{ and} \\ E_\phi &= -j\frac{e^{-jkr}}{\lambda r}\frac{1+\cos\theta}{2}f_A(\theta)\cos\phi. \end{aligned} \tag{2.29}$$

Equations for Numerical Evaluation of Radiation Patterns

The equation (2.14) for the horn feed will become [15]:

$$f_A(\theta,\phi) = \int_0^{\psi_0}\int_0^{2\pi} F_A(\psi,\chi,\theta,\phi)d\psi d\chi, \tag{2.30}$$

where the integrand depends on the feed pattern $A(\psi,\chi)$:

$$F_A(\psi,\chi,\theta,\phi) = A(\psi,\chi)e^{2jkF\tan\frac{\psi}{2}\sin\theta\cos(\phi-\chi)}\tan\frac{\psi}{2}, \tag{2.31}$$

and the function $A(\psi,\chi)$ is given by, up to constant factors:

$$A(\psi,\chi) = (1+cos\psi)F_1(\nu_x,\sigma_a)F_0(\nu_y,\sigma_b). \tag{2.32}$$

Once $f_A(\theta,\phi)$ is computed, the un-normalized gains along the H- and E-plane radiation patterns for the reflector are obtained by setting $\phi = 0°$ and $90°$. That is:

$$g_H(\theta) = \left|(1+\cos\theta)f_A(\theta,0°)\right|^2, \quad g_E(\theta) = \left|(1+\cos\theta)f_A(\theta,90°)\right|^2. \tag{2.33}$$

The numerical evaluation of the integral can be done with two-dimensional Gauss-Legendre quadratures, approximating the integral by the double sum:

$$f_A(\theta,\phi) = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} w_{1i}F_A(\psi_i,\chi_j)w_{2j} = \mathbf{w_1^T F_A w_2}, \tag{2.34}$$

where $[w_{1i},\psi_i]$ and $[w_{2j},\chi_j]$ are the quadrature weights and evaluation points over the intervals $[0,\psi_0]$ and $[0,2\pi]$, and $\mathbf{F_A}$ is the matrix $F_A(\psi_i,\chi_j)$.

It is possible to simplify (2.32) considering that the E- and H-plane illumination patterns are virtually identical over the angular range $[0,\psi_0]$, provided one chooses the horn sides that $A_h = 1.48B_h$. In this case, (2.32) becomes:

$$A(\psi) = (1 + cos\psi)F_0(\nu_y,\sigma_b), \tag{2.35}$$

and the function f_A can be calculated explicitly by (2.27).

Numerical Evaluation of Radiation Patterns

To implement the equations described in Section 2.1.2, the following antenna is considered:

Table 2.1: Ground antenna parameters.

Parameter	Description	Value
D	Antenna diameter	1 m
ψ_0	Parabola max aperture angle	60°
F	Parabola optical length (2.9)	0.433 m
f	Carrier frequency	8363 MHz
B_h	Horn side	0.7806 λ
A_h	Horn side	$1.48B_h$
e_a	Antenna aperture efficiency	0.75
σ_a	Horn σ parameter	1.2593
σ_b	Horn σ parameter	1.0246
G_{ground}	Maximum gain (2.10)	37.61 dB
T_{sys}	Receiver noise temperature	18.23 dBK

It was developed a python class inside *arraytools* that receives all the necessary design parameters and evaluates the gains, which are displayed in Figure 2.3. The script used to generate this graph is shown in Appendix B.1.

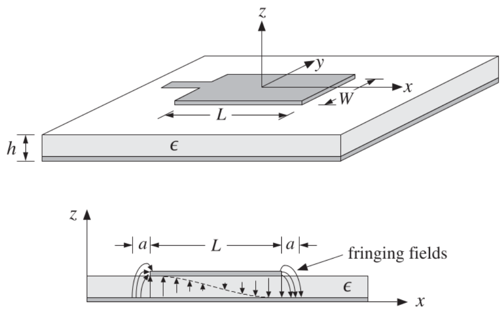

Figure 2.4: Microstrip antenna and E-field pattern in substrate [15]. ϵ is the dielectric permittivity. L is the x length, W is the y length, a is the extension of the length L due to fringing fields and h is the substrate height.

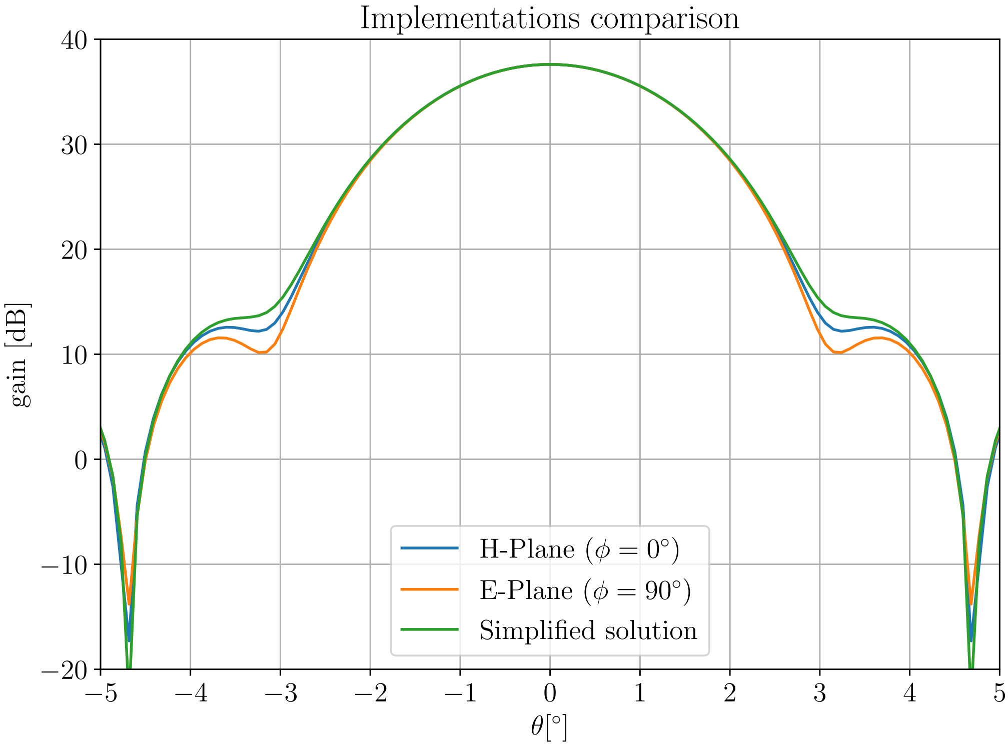

Figure 2.3: Parabolic horn feed gain pattern implementation comparison. The H- and E- Plane solutions are evaluated by the equations (2.33), whereas the simplified solution is calculated using equations (2.27) and (2.35). From this, it is concluded that the simplified implementation is close to the non-simplified one and has a computational cost significantly lower.

2.1.3 Microstrip Antenna

Another type of antenna proposed for the ground stations is the rectangular microstrip antenna as shown in Figure 2.4.

The patch acts as a resonant cavity with an electric field perpendicular to the patch, that is, along the z-direction. The magnetic field has vanishing tangential components at the four edges of the patch. The field of the lowest resonant mode (assuming $L \geq W$) are given by [15]:

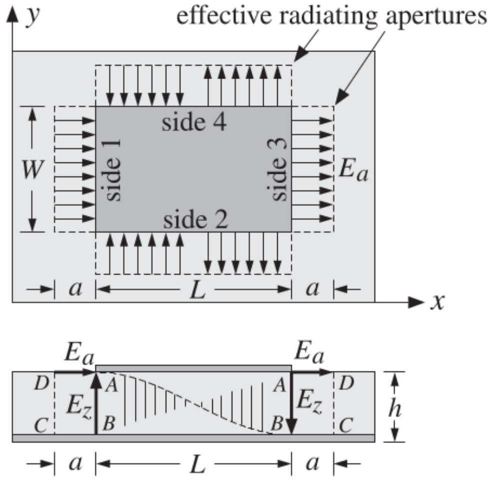

Figure 2.5: Aperture model for microstrip antenna [15].

$$\begin{aligned} E_z(x) &= -E_0 \sin\left(\frac{\pi x}{L}\right), \quad -\frac{L}{2} \leq x \leq \frac{L}{2}, \text{ and} \\ H_y(x) &= -H_0 \cos\left(\frac{\pi x}{L}\right), \quad -\frac{W}{2} \leq y \leq \frac{W}{2}, \end{aligned} \tag{2.36}$$

where $H_0 = -jE_0/\eta_0$.

The aperture model considered is shown in Figure 2.5. The fields are given by:

$$\begin{aligned} &\text{for sides 1 \& 3:} && \mathbf{E}_a = \hat{\mathbf{x}}\frac{hE_0}{a}, \\ &\text{and for sides 2 \& 4:} && \mathbf{E}_a = \pm\hat{\mathbf{y}}\frac{hE_z(x)}{a} = \mp\hat{\mathbf{y}}\frac{hE_0}{a}\sin\left(\frac{\pi x}{L}\right). \end{aligned} \tag{2.37}$$

The outward normal to the aperture plane is $\hat{\mathbf{n}} = \hat{\mathbf{z}}$ for all four sides. Therefore, the surface magnetic currents $\mathbf{J}_{ms} = -2\hat{\mathbf{n}} \times \mathbf{E}_a$ become:

$$\begin{aligned} &\text{for sides 1 \& 3:} && \mathbf{J}_{ms} = -\hat{\mathbf{y}}\frac{2hE_0}{a}, \\ &\text{and for sides 2 \& 4:} && \mathbf{J}_{ms} = \mp\hat{\mathbf{x}}\frac{2hE_0}{a}\sin\left(\frac{\pi x}{L}\right). \end{aligned} \tag{2.38}$$

The radiated electric field is obtained by:

$$\mathbf{E} = jk\frac{e^{-jkr}}{4\pi r}\hat{\mathbf{r}} \times [\mathbf{F}_{m1} + \mathbf{F}_{m2} + \mathbf{F}_{m3} + \mathbf{F}_{m4}], \tag{2.39}$$

where the vectors $\mathbf{F}_m$ are the two-dimensional Fourier transforms over the apertures:

$$\mathbf{F}_m(\theta, \phi) = \int_A \mathbf{J}_{ms}(x, y)e^{jk_x x + jk_y y}dS. \tag{2.40}$$

The Fourier transforms for each side becomes:

$$
\begin{aligned}
F_{m,13} &= -\hat{\mathbf{y}} 4E_0 hW \cos\left(\pi\nu_x\right) \operatorname{sinc}\left(\pi\nu_y\right), \text{ and} \\
F_{m,24} &= \hat{\mathbf{x}} 4E_0 hL \frac{4\nu_x \cos\left(\pi\nu_x\right)}{\pi\left(1-4\nu_x^2\right)} \sin\left(\pi\nu_y\right),
\end{aligned}
\tag{2.41}
$$

with:

$$
\begin{aligned}
\nu_x &= \frac{k_x L}{2\pi} = \frac{L}{\lambda}\frac{x}{r} = \frac{L}{\lambda}\sin\theta\cos\phi, \text{ and} \\
\nu_y &= \frac{k_y W}{2\pi} = \frac{W}{\lambda}\frac{y}{r} = \frac{W}{\lambda}\sin\theta\sin\phi.
\end{aligned}
\tag{2.42}
$$

Therefore, the radiated field from sides 1 & 3 are:

$$
\mathbf{E}(\theta,\phi) = -jk\frac{e^{-jkr}}{4\pi r} 4E_0 hW \left[\hat{\boldsymbol{\phi}}\cos\theta\sin\phi - \hat{\boldsymbol{\theta}}\cos\phi\right] F(\theta,\phi), \text{ and} \tag{2.43}
$$

$$
F(\theta,\phi) = \cos(\pi\nu_x)\operatorname{sinc}(\pi\nu_y). \tag{2.44}
$$

Similarly, for sides 2 & 4:

$$
\mathbf{E}(\theta,\phi) = jk\frac{e^{-jkr}}{4\pi r} 4E_0 hL \left[\hat{\boldsymbol{\phi}}\cos\theta\cos\phi + \hat{\boldsymbol{\theta}}\sin\phi\right] f(\theta,\phi), \text{ and} \tag{2.45}
$$

$$
f(\theta,\phi) = \frac{4\nu_x \cos\left(\pi\nu_x\right)}{\pi\left(1-4\nu_x^2\right)} \sin\left(\pi\nu_y\right). \tag{2.46}
$$

Rectangular Patch Design

For a given frequency f, a substrate height h and a substrate relative permittivity ϵ_r, it is possible to calculate the patch dimensions using the method presented in [8]:

1. A practical width that leads to good radiation efficiency is:

$$
W = \frac{c_0}{2f}\sqrt{\frac{2}{\epsilon_r + 1}}; \tag{2.47}
$$

2. The effective dielectric constant of the microstrip antenna is:

$$
\epsilon_{\text{reff}} = \frac{\epsilon_r + 1}{2} + \frac{\epsilon_r - 1}{2}\left[1 + 12\frac{h}{W}\right]^{-1/2}; \tag{2.48}
$$

3. Because of the fringing effects, electrically the patch of the microstrip antenna looks greater than its physical dimensions, as shown in Figure 2.5. For the principal E-plane, the extension a is given by:

$$
a = 0.412h\frac{\left(\epsilon_{\text{reff}} + 0.3\right)\left(\dfrac{W}{h} + 0.264\right)}{\left(\epsilon_{\text{reff}} - 0.258\right)\left(\dfrac{W}{h} + 0.8\right)}; \tag{2.49}
$$

4. The actual length of the patch can now be determined by:

$$L = \frac{c_0}{2f\sqrt{\epsilon_{\text{reff}}}} - 2a \ . \tag{2.50}$$

Using the design method presented above, consider the antenna with the parameters shown in Table 2.2.

Table 2.2: Parameters for the microstrip antenna.

Parameter	**Description**	**Value**
f	Carrier frequency	10 GHz
h	Substrate height	0.1588 cm
ϵ_r	Relative permittivity	2.2
W	Patch width	1.185 cm
L	Patch length	0.905 cm

For this antenna, the electric fields given by (2.44) and (2.46) are shown in Figure 2.6. It was implemented a class called *MicroStrip* inside the module *arraytools* that implements the rectangular patch equations. The result is compared against a simulation with the software ANSYS HFSS, as shown in Figure 2.6. The script used to generate this graph is displayed in Appendix B.2.

2.1.4 Directivity from Electric Field

The radiation intensity of an antenna is given by [8]:

$$U = B_0 F(\theta,\phi) \approx \frac{1}{2\eta_0}\left[|E_\theta(\theta,\phi)|^2 + |E_\phi(\theta,\phi)|^2\right], \tag{2.51}$$

where $B_0 \approx \dfrac{1}{2\eta_0}$ is a constant, η_0 is the free space wave impedance, E_θ and E_ϕ are the far-field components of the antenna in spherical coordinates.

The total radiated power is:

$$P_{rad} = \oiint_\Omega U(\theta,\phi)d\Omega = B_0\int_0^{2\pi}\int_0^{\pi} F(\theta,\phi)\sin\theta d\theta d\phi. \tag{2.52}$$

Therefore, the expression for the directivity is:

$$D(\theta,\phi) = \frac{4\pi U(\theta,\phi)}{P_{rad}} = 4\pi\frac{F(\theta,\phi)}{\int_0^{2\pi}\int_0^{\pi} F(\theta,\phi)\sin\theta d\theta d\phi}. \tag{2.53}$$

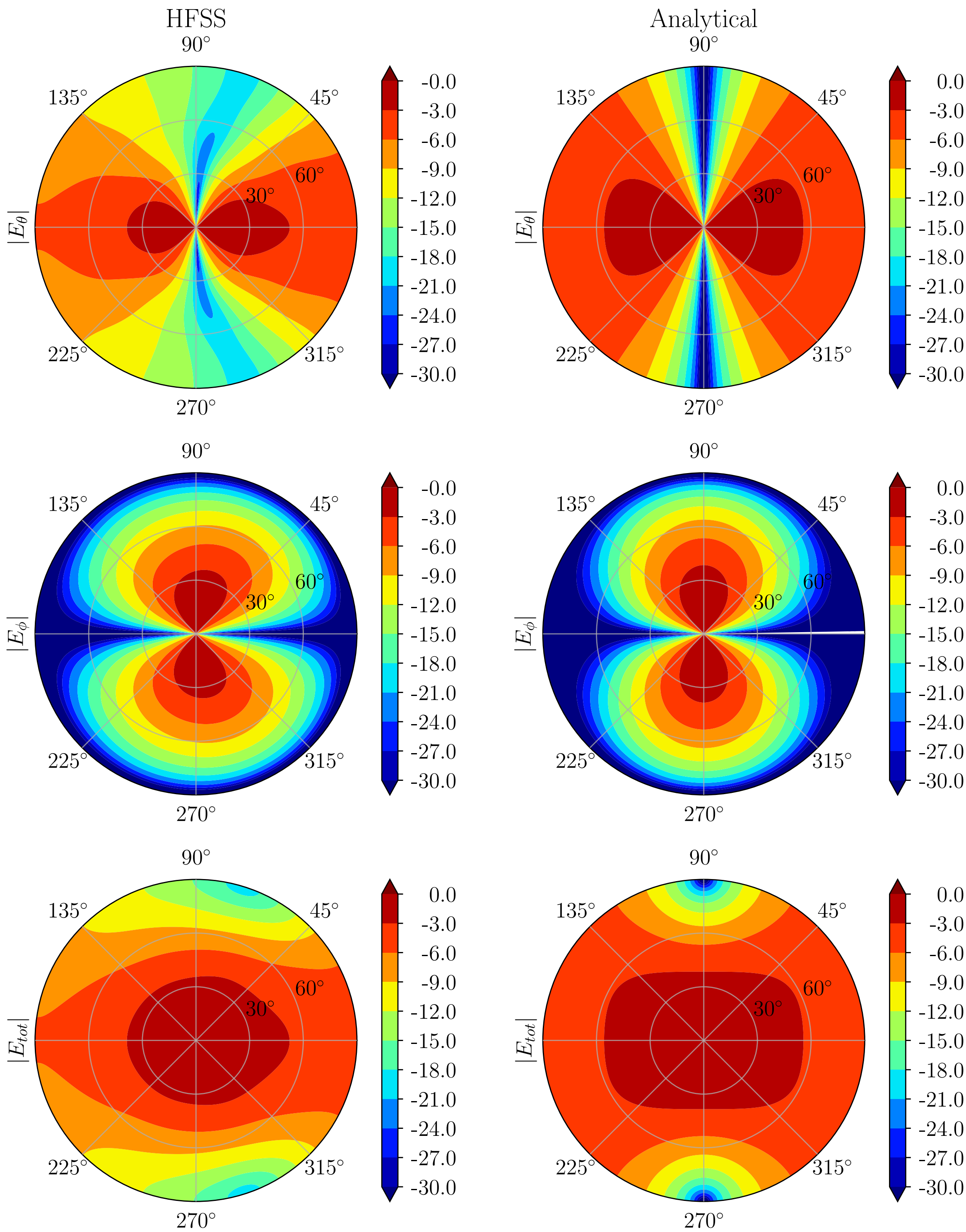

Figure 2.6: Electric fields for the microstrip antenna using the presented equations compared with a simulation in HFSS. The differences between the two are expected, as the presented equations are approximations and the HFSS software considers other factors, such as the asymmetry regarding the voltage input.

Given that the space θ and ϕ are divided, respectively, into N and M uniform intervals, the denominator of Eq. 2.53 can be approximated as:

$$\int_0^{2\pi}\int_0^{\pi} F(\theta,\phi)\sin\theta d\theta d\phi = \left(\frac{\pi}{N}\right)\left(\frac{2\pi}{M}\right)\sum_{j=1}^{M}\left[\sum_{i=1}^{N} F(\theta_i,\phi_j)\sin\theta_i\right]. \tag{2.54}$$

Therefore, the directivity can be approximated as:

$$D(\theta,\phi) = \frac{2MN}{\pi}\frac{F(\theta,\phi)}{\sum\limits_{j=1}^{M}\left[\sum\limits_{i=1}^{N} F(\theta_i,\phi_j)\sin\theta_i\right]}. \tag{2.55}$$

2.2 Far-Field Pattern Rotation

This section presents the mathematical model for rotating far-fields pattern of antennas. The rectangular patch antenna model is adopted, because it has simpler mathematical equations.

Generally, the patch from Figure 2.4 is rotated of angles α, β, γ around the local x, y, z-axes. As the rotations are represented around the local axis, it is necessary to transform this local coordinate system to match the initial global one.

The direction defined by the angles (θ, ϕ) of the original coordinate system is represented by the vector $\hat{r} = \begin{bmatrix}\cos\phi\sin\theta \\ \sin\phi\sin\theta \\ \cos\theta\end{bmatrix}_{xyz}$.

This direction expressed in the rotated coordinate system is given by $\hat{r}' = R_{\alpha\beta\gamma}\hat{r} = R_{\alpha\beta\gamma}\begin{bmatrix}\sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta\end{bmatrix} = \begin{bmatrix}x' \\ y' \\ z'\end{bmatrix}$, where $R_{\alpha\beta\gamma}$ is the rotation matrix and represents the base transformation from $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ to $(\hat{\mathbf{x}}', \hat{\mathbf{y}}', \hat{\mathbf{z}}')$.

From the vector $\hat{r}'$ it is possible to extract the direction (θ', ϕ') related to the rotated coordinate system by $\theta' = \arccos\dfrac{z'}{\sqrt{x'^2 + y'^2 + z'^2}}$ and $\phi' = \arctan\dfrac{y'}{x'}$, where the function arctan is evaluated choosing the quadrant correctly and its obtained values are in the range $-\pi \leq \arctan x \leq \pi$.

To ensure that the angles are in the classical spherical coordinates range, with $0 \leq \phi < 2\pi$ and $0 < \theta < \pi$, some transformations are necessary:

- If $\phi' < 0$:

$$\phi' \to \phi' + 2\pi, \text{ and} \tag{2.56}$$

- If $\theta' < 0$:

$$\begin{aligned}\theta' &\to -\theta' \\ \phi' &\to (\phi' + \pi) \mod 2\pi.\end{aligned} \tag{2.57}$$

With the angles in the local antenna field, it is possible to evaluate the fields E'_θ and E'_ϕ. The directions $\hat{\theta}'$ and $\hat{\phi}'$ are given by $\hat{\theta}' = \begin{bmatrix} \cos\phi' \cos\theta' \\ \sin\phi' \cos\theta' \\ -\sin\theta' \end{bmatrix}_{x'y'z'}$ and $\hat{\phi}' = \begin{bmatrix} -\sin\theta' \\ \cos\theta' \\ 0 \end{bmatrix}_{x'y'z'}$.
Finally, the field expressed in the original coordinate frame is:

$$\mathbf{E}_{xyz} = E'_\theta R^{-1}_{\alpha\beta\gamma} \hat{\theta}' + E'_\phi R^{-1}_{\alpha\beta\gamma} \hat{\phi}'. \tag{2.58}$$

Expressing it again in spherical coordinates:

$$\begin{aligned} E_\theta &= \begin{bmatrix} \cos\phi \cos\theta & \sin\phi \cos\theta & -\sin\theta \end{bmatrix} \mathbf{E}_{xyz}, \text{ and} \\ E_\phi &= \begin{bmatrix} -\sin\phi & \cos\phi & 0 \end{bmatrix} \mathbf{E}_{xyz}. \end{aligned} \tag{2.59}$$

To validate the results, the fields obtained by the *MicroStrip* class developed in Section 2.1.3 are rotated of $\beta = 45°$ around the y-axis. This rotated far field is then compared with a simulation from HFSS, as shown in Figure 2.7. This is done by the script in Appendix B.3.

Additionally, the field pattern of the non-rotated patch from HFSS is analytically rotated by $\beta = 45°$ around the y-axis. This analytically rotated pattern is compared with the pattern obtained by simulating the rotated patch in HFSS. The comparison is presented in Figure 2.8. This is done by the script in Appendix B.4.

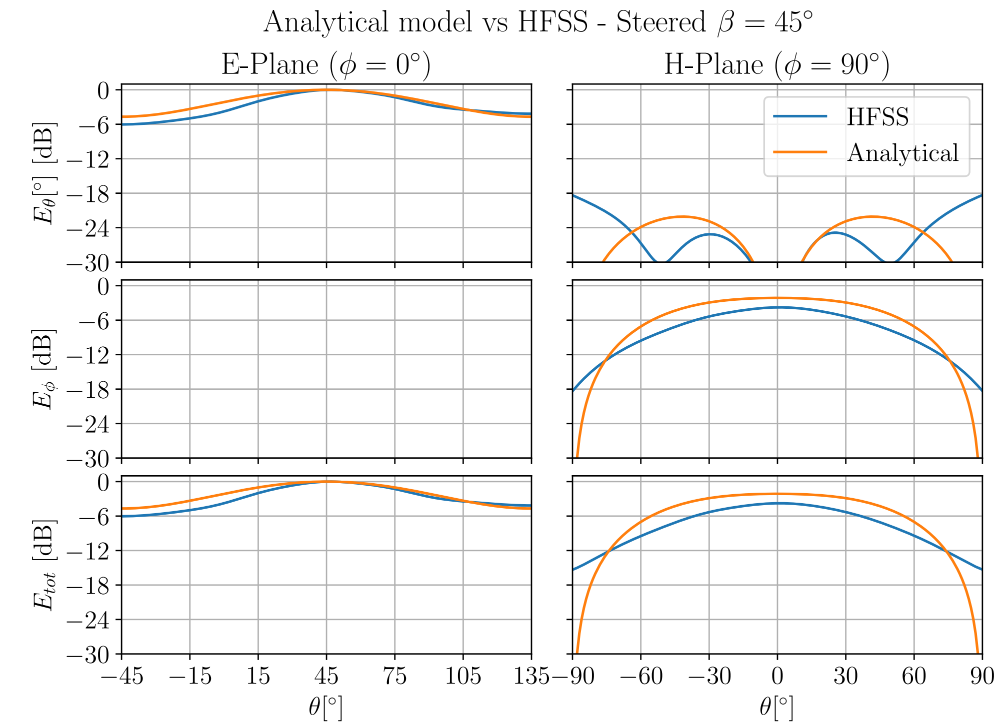

Figure 2.7: E-plane and H-plane for the microstrip antenna steered of $\beta = 45°$ around the y-axis using the presented equations compared with a simulation of a steered antenna by the same angle in HFSS. The differences exists because the model proposed by [8] is a simplification.

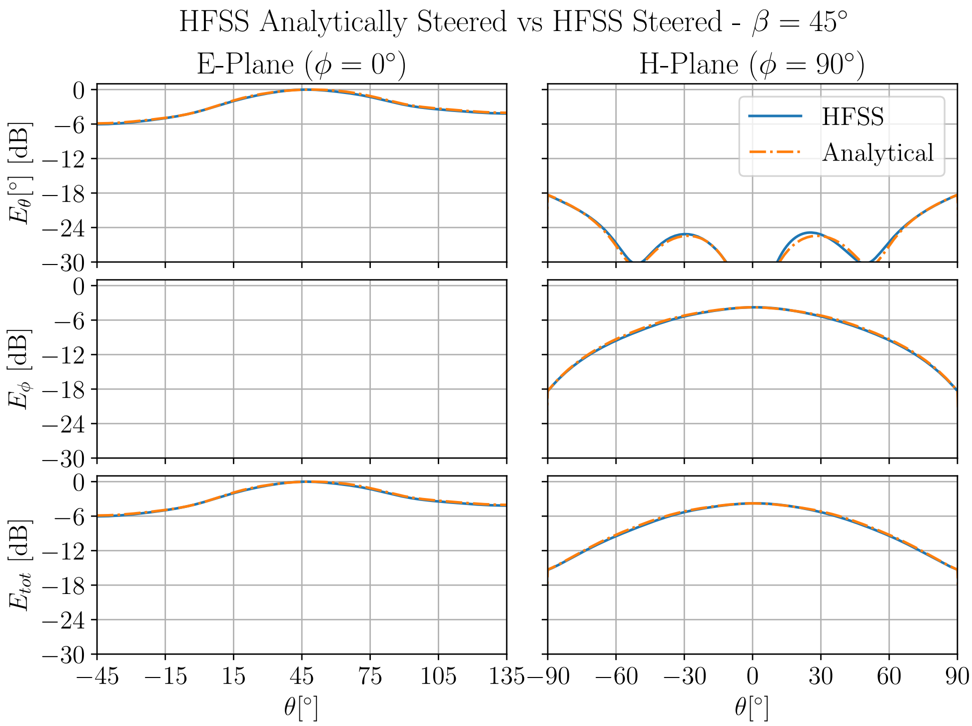

Figure 2.8: E-plane and H-plane for the microstrip antenna steered of $\beta = 45°$ around the y-axis using field patterns from HFSS and then analytically rotated compared with a simulation in HFSS. From this result, it is possible to see that the proposed algorithm produces virtually the same result as the HFSS simulation.

2.3 Satellite Propagation

The propagation of satellites in this work is done using the SMET software, developed by Rafael Rodrigues Luz Benevides in the context of the CNPq project.

The software is capable of propagating satellites from the Two-Line Elements using SGP4 propagator, calculating the accesses and line-of-sight angles (in azimuth and elevation) from a specific point, among other capabilities. It provides useful information for evaluating the antenna coverage, like geodetic coordinates of the satellite over time and distance in relation to the station over time.

Using SMET, it was not necessary any other software like ANSYS STK to deal with the satellite propagation or accesses necessities.

2.3.1 Considered Satellites

For this study, two satellites were chosen: SAC-C from Argentina, with which NASA did some works and VCUB1 from Visiona.

SAC-C Satellite

The Satellite for Scientific Applications (SAC-C) was developed through the partnership of Argentine CONAE (National Space Activities Commission) and NASA.

SAC-C satellite is used as a validation tool for the link budget proposed algorithms. Initially, the results of Section 2.4 are compared with the studies [9] [10] [12]. These studies provided the necessary information regarding the SAC-C link capabilities to validate the proposed algorithm.

After the procedures validation, it is proposed a method to optimize the link coverage of a territory, focusing on Brazil. After this, the scenario is modified to consider other satellites.

VCUB1 Satellite

The second satellite chosen for this study was VCUB1, developed by Visiona, which is a company that has ongoing collaborations with the University of Brasília on the scope of the CNPq project coordinated by UnB Telecommunications Laboratory. This partnership creates a link between academia and the Brazilian space industry and the choice of VCUB1 not only emphasizes Brazilian technological advancements but also highlight a greater contribution from academic research to the national space sector.

VCUB1 was launched on April 15^{th}, 2023 and is the first Earth Observation Satellite designed by the Brazilian national industry and should demonstrate the capabilities to realize advanced space missions. The satellite has a camera with spacial resolution of 3.5m, which allows it to do agricultural and environmental monitoring.

Adding to these facts, VCUB1 is a LEO satellite in Sun-Synchronous Orbit (inclination of 98°) and fits well with the proposed algorithms of distributing ground stations inside a given territory to maximize coverage and compare these results with a ground station positioned in the poles.

2.4 Link Budget

The procedure that will be followed to calculate the link budget is described in [15].

2.4.1 Transmitted Power

The transmitted power P_{EIRP} is given by:

$$p_{\mathrm{EIRP}} = p_{\mathrm{out}} g_{\mathrm{sat}}, \tag{2.60}$$

where p_{out} is the transmitted power of the satellite and g_{sat} is the satellite antenna gain.

2.4.2 Free Space Losses

The free space losses are given by:

$$g_{fu} = \left(\frac{\lambda}{4\pi r}\right)^2, \tag{2.61}$$

where λ is the carrier wavelength and r is the distance between the transmitter and the receiver. The free space losses in dB is represented by G_{fu}, which represents a negative value.

2.4.3 Power Received by Earth Antenna

The power received by the ground station in dB is given by:

$$P_{\text{rec}} = P_{\text{EIRP}} + G_{fu} + P_{\text{atm}} + G_{\text{ground}}, \tag{2.62}$$

with all values in dB. P_{atm} is modeling implementation and atmospheric losses, which represents a negative value and G_{ground} is the ground antenna gain.

2.4.4 Signal to Noise Ratio

The SNR, or Signal to Noise Ratio, is given by:

$$\text{SNR} = \frac{P_{\text{rec}}}{kT_{\text{sys}}B}, \tag{2.63}$$

where k is the Boltzmann constant, T_{sys} is the noise temperature of the receiver and B is the bandwidth of the transmitted signal.

2.4.5 Data Rate from Energy Bit per Noise Ratio

The data rate R can be obtained from the SNR:

$$R = \text{SNR}\frac{B}{E_b/N_0}, \tag{2.64}$$

where E_b/N_0 is the energy bit per noise ratio and depends on modulation and acceptable bit-error probability, P_e.

Alternatively, it is possible to express the ratio E_b/N_0, considering that the data is transmitted at a fixed data rate, R_{spec}:

$$E_b/N_0 = \text{SNR}\frac{B}{R_{\text{spec}}}. \tag{2.65}$$

In this scenario, the analysis is not dependant on the type of modulation used.

Data Rate from $P_{e_{\max}}$

When dealing with the bit-error probability, it is necessary to know the used modulation. For BPSK and QPSK modulations, the E_b/N_0 is given by:

$$\frac{E_b}{N_0} = [\text{erfcinv}(2P_e)]^2, \tag{2.66}$$

where erfcinv is the complementary error inverse function.

Inverting Eq. (2.66):

$$P_e = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right). \tag{2.67}$$

2.4.6 Satellite Link

The satellite used for this section is the SAC-C, which is a LEO satellite with the following characteristics:

Table 2.3: SAC-C parameters.

Parameter	Description	Value
i	Inclination	97.4°
r_p	Perigee	422.6 km
r_a	Apogee	434 km
P_{EIRP}	Satellite Antenna EIRP	42.38 dB
f	Carrier frequency	8363 MHz
R_{spec}	Output data rate	3.303 Mbps
B	Bandwidth	13.3 MHz

It is also considered that the ground stations is tracking the satellite. That is, the maximum gains will be used.

The 1 m ground antenna with parameters described in Table 2.1 is located at latitude −10.78° and longitude −53.07°.

To evaluate the link budget results, first the accesses and distances between the ground station and the propagated satellite are calculated. Then, using (2.61), the free losses are computed. Considering that the atmospheric losses are constant and equal to −1.6 dB and using (2.62), the received power at ground station is evaluated. The free losses and power received are shown in Figure 2.9.

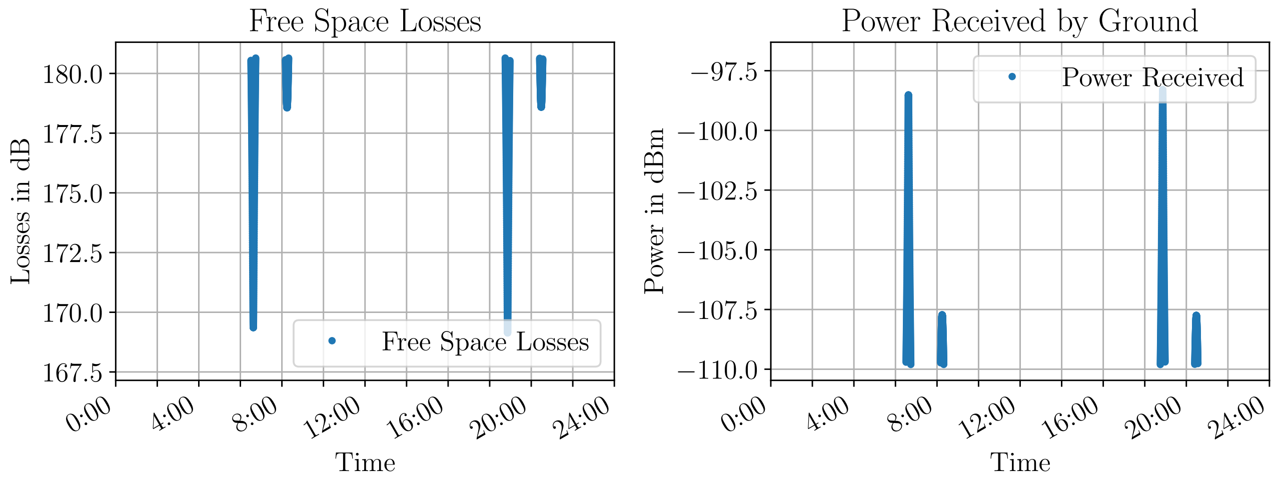

Figure 2.9: Free losses and power received during a day with 4 passes over the considered ground station, evaluated using the equations (2.61) and (2.62), respectively.

Using the received power in (2.63), the SNR is computed. Using the SNR and the specified data transmission, R_{spec}, in (2.65), the E_b/N_0 is evaluated. The SNR and E_b/N_0 are shown in Figure 2.10.

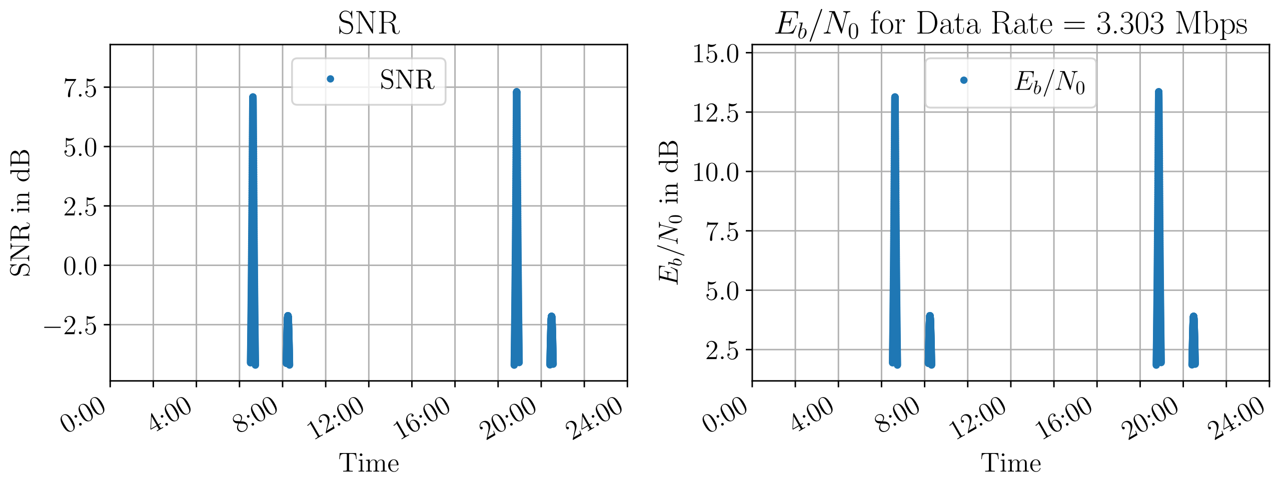

Figure 2.10: SNR and E_b/N_0 during a day with 4 passes over the considered ground station, evaluated using equations (2.63) and (2.65), respectively.

Considering only the points where the energy per bit ratio is greater than the threshold $(E_b/N_0)_{\text{min}} = 6.38\,\text{dB}$ it is possible to estimate how much data is downloaded during a day by computing how much time this condition is fulfilled in a day and multiplying by the data rate. The results is shown in Figure 2.11, together with the radiation pattern of the antenna, that is, in which points it is possible to establish a link. This pattern is the longitude and latitude points where the condition $(E_b/N_0) > 6.38\,\text{dB}$ is true projected on the ground.

Alternatively, it is possible to analyze the same problem using the bit-error probability approach. Considering that the modulation is BPSK and using Eq. (2.67), the bit-error probability is evaluated and shown in Figure 2.12.

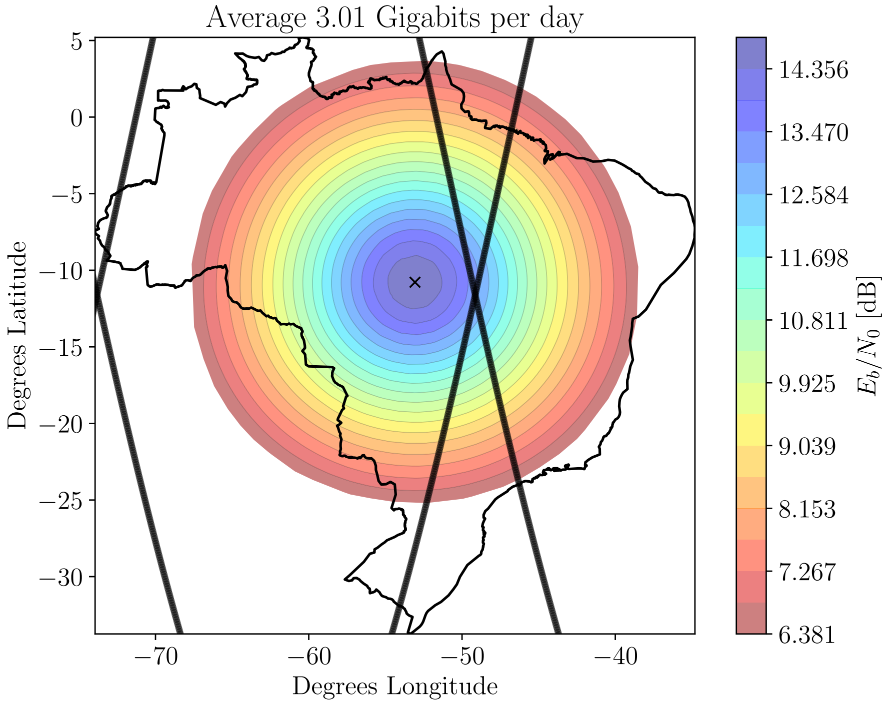

Figure 2.11: E_b/N_0 heat map. The black lines are the satellite path. From this it is possible to see the coverage of the antenna, as it shows the positions where the E_b/N_0 is greater than the necessary to establish the link.

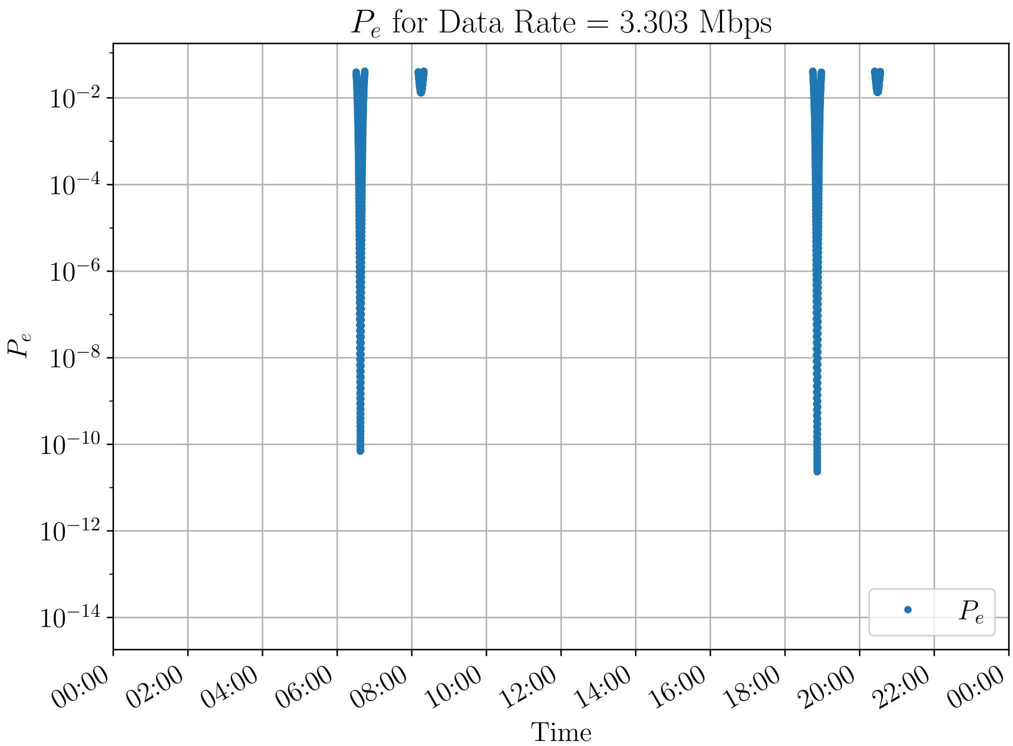

Figure 2.12: P_e during a day with 4 passes over the considered ground station.

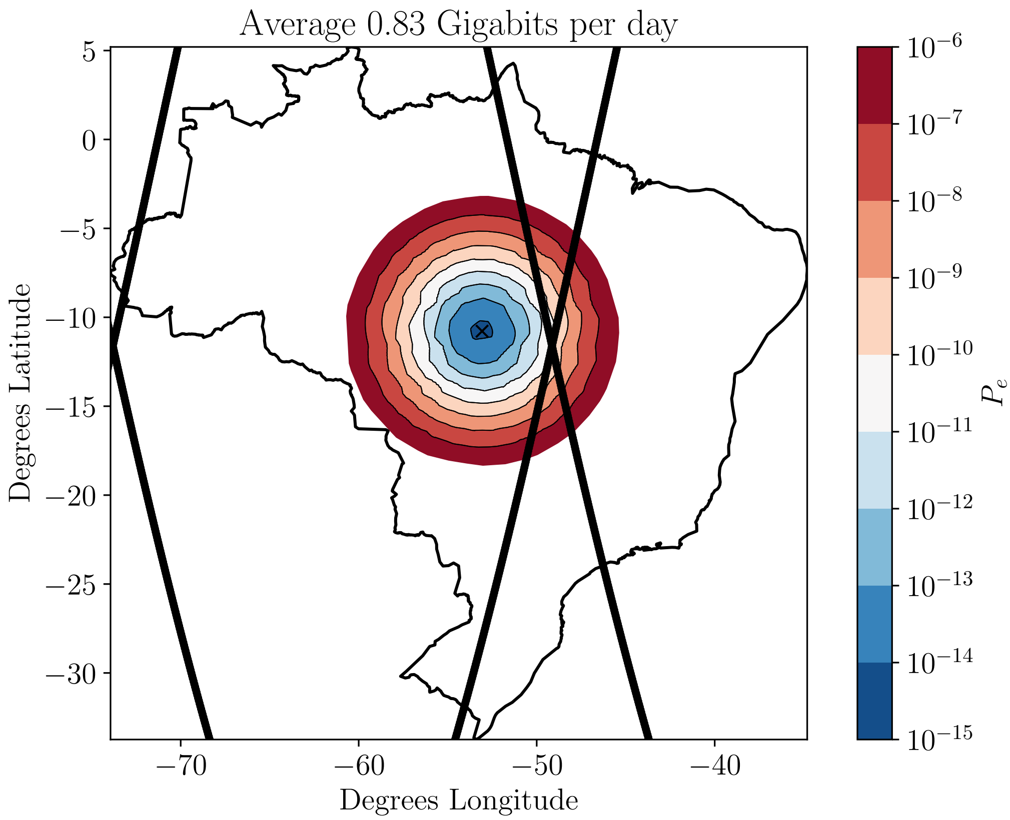

Figure 2.13: P_e heat map. The black points are the satellite path. From this it is possible to see the coverage of the antenna, as it shows the positions where the bit-error probability is less than the necessary to close the link.

Similarly, to the Energy per Bit over Noise ratio approach, considering a threshold $P_{e_{\max}} < 10^{-6}$ necessary for establish a link, it is possible to estimate the downloaded data during a day and to obtain the radiation pattern of the antenna, as shown in Figure 2.13.

There is a script that evaluates a link budget example in Appendix B.5. This script uses three new implemented classes in *arraytools*: the *Satellite*, *Station* and *LinkBudget* classes.

2.4.7 Secant Antenna Gain

The received power, given by Eq. (2.62), can be reformulated as:

$$P_R = \frac{p_{\text{out}} g_{\text{sat}} g_{\text{ground}} g(\theta, \phi) \lambda^2}{(4\pi)^2 r^2}, \tag{2.68}$$

where $g(\theta, \phi)$ is the normalized gain of the ground station, r is the distance to the tracked object and all values are represented in absolute units.

In ground stations tracking an approaching satellite at a constant altitude h, the ground station power received can be made independent of the distance r, within a specific range by selecting an appropriate gain function $g(\theta, \phi)$.

As shown in Figure 2.14, $h = r \cos\theta$.

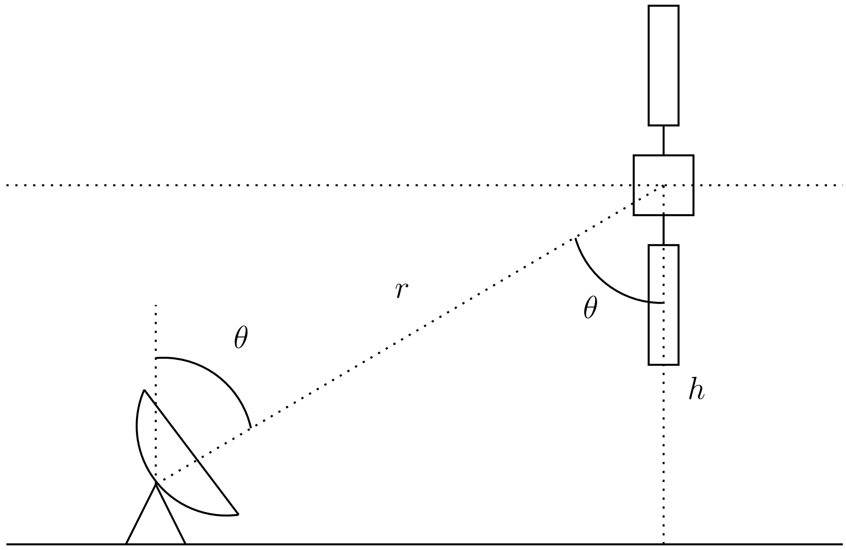

Figure 2.14: Schematic of a satellite approaching a ground station.

If the gain is designed to have the secant-squared shape $g(\theta, \phi) = \dfrac{K}{\cos^2\theta}$, where K is a constant, the power will become independent of r:

$$P_R = \frac{p_{\text{out}} g_{\text{sat}} g_{\text{ground}} g(\theta, \phi) \lambda^2}{(4\pi)^2 r^2} = \frac{p_{\text{out}} g_{\text{sat}} g_{\text{ground}} K \lambda^2}{(4\pi)^2 r^2 \cos^2\theta} = \frac{p_{\text{out}} g_{\text{sat}} g_{\text{ground}} K \lambda^2}{(4\pi)^2 h^2}. \tag{2.69}$$

The secant behavior is valid over the range $0 \leq \theta \leq \theta_{\max}$, where $\theta_{\max}$ is the desired maximum range of the ground station $r_{\max} = \dfrac{h}{\cos\theta_{\max}}$.

2.5 Optimization Algorithms

This section explores some optimization techniques, which are used in this work.

First it is presented the convex optimization, that performs minimization of convex functions over convex sets. The main advantage of this method is its capability to find the global minimum. It is widely used in engineering and data analysis.

Then, it is presented the sequential quadratic programming, used for nonlinear programming problems. It is very useful to approach constrained optimization problems.

Lastly, it is presented the differential evolution algorithm. It is a heuristic approach to solve highly nonlinear problems.

2.5.1 Convex Optimization

A convex optimization problem is defined as [7]:

$$\begin{aligned} &\min \|f_0(x)\| \quad \text{subject to} \\ &\|f_i(x)\| \leq b_i, i = 1, \ldots, m, \end{aligned} \tag{2.70}$$

where the functions $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ are convex, that is:

$$f_m(\alpha x + \beta y) <= \alpha f_m(x) + \beta f_m(y). \tag{2.71}$$

A problem modeled as a convex one will achieve the global minimum, it the minimum exists.

2.5.2 Sequential Least Squares

Sequential quadratic programming is a efficient computational method to solve the general nonlinear problem:

$$\begin{aligned} &\min_{x \in \mathbb{R}} f(x) \quad \text{subject to} \\ &g_j(x) = 0, j = 1, \ldots, m_e, \\ &g_j(x) \geq 0, j = m_e + 1, \ldots, m, \text{ and} \\ &x_0 \leq x \leq x_n, \end{aligned} \tag{2.72}$$

for a local minimum, where the problem functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are assumed to be continuously differentiable.

These problems are solved by many open source libraries, like Python SciPy.

2.5.3 Differential Evolution

Differential Evolution (DE) is a heuristic parallel direct search method which utilizes N_P vectors y_i^g and $i \in \mathbb{Z}$ with $0 \leq i < N_P$ as a population for each generation g of a total G [4].

For each vector y_i^g, an offspring vector v_i^{g+1} is generated according to:

$$v_i^{g+1} = y_{r_1}^g + F_\beta \left(y_{best} - y_{r_1}^g \right) + F_\alpha \left(y_{r_2}^g - y_{r_3}^g \right), \tag{2.73}$$

where r_1, r_2, r_3 are random different integers with $0 \leq (r_i)_{i=1,2,3} < N_P$.

The parameters F_α and F_β are control variables. F_α controls the amplification of the differential variation $\left(y_{r_3}^g - y_{r_4}^g \right)$. F_β provides a mean to enhance the greediness of the scheme by incorporating the current best vector y_{best}. This is known as mutation.

The mutated vector v_i^{g+1} is then combined with its parent y_i^g to generate u_i^{g+1} according to:

$$u_{i,j}^{g+1} = \begin{cases} v_{i,j}^{g+1}, \text{if } \left(r_j \leq C_R \vee j = j_{rand} \right), \\ y_{i,j}^g, \text{otherwise}, \end{cases} \tag{2.74}$$

which is known as crossover or recombination. This is the main differential of the method.

Finally, $u_{i,j}^{g+1}$ is evaluated by the cost function. If it has a lower cost than $y_{i,j}^g$, it is selected as the next generation $y_{i,j}^{g+1}$. Otherwise, the vector $y_{i,j}^g$ survives for the next generation.

Mathematically:

$$y_{i,j}^{g+1} = \begin{cases} u_{i,j}^{g+1}, \text{if } f\left(u_{i,j}^{g+1} \right) < f\left(y_{i,j}^g \right), \\ y_{i,j}^g, \text{otherwise}, \end{cases} \tag{2.75}$$

where $f(y)$ is the cost function.

2.6 Spherical Modes

Spherical harmonics are special functions defined on the surface of a sphere. They are useful to solve differential equations in many fields, including decomposing electromagnetic fields as the spherical modes method is particularly useful in antenna modeling due to its ability to account for the geometry and boundary conditions inherent to spherical structures.

The electric fields are represented in spherical modes by [19]:

$$E(\theta, \phi, k) = \sum_{l \in \mathbb{N}} \sum_{|m| \leq l} \boldsymbol{T}_{lm}(\theta, \phi) \boldsymbol{q}_{lm}(k), \tag{2.76}$$

where q_{lm} are the mode coefficients and can be determined if E is analytically available through the expression:

$$\boldsymbol{q}_{lm} = \frac{1}{\eta_0} \oiint \boldsymbol{T}_{lm}^{H} E d\Omega. \tag{2.77}$$

In these equations, $\boldsymbol{T}_{lm}$ is derived from the eigenfunctions $\boldsymbol{Y}_{lm}\boldsymbol{Z}_{ls}$, l is the degree and m is the order of the corresponding spherical harmonics $\boldsymbol{Y}$ and $\boldsymbol{Z}$ as defined in [19].

This study uses the software developed by [19], which estimates antennas in spherical modes based on their analytical electric fields. The software, called AFTK, models the antennas used in the design of the proposed arrays of this work.

Part III

Ground Stations Distribution

Chapter 3

Ground Stations Distribution

This chapter, inspired by the studies studies [9] and [10], compares a ground station distribution inside the United Stated with a parabola placed near the pole, proposes an algorithm to find a ground station distribution inside Brazilian territory that: maximizes the link coverage for a given satellite and minimizes the number of employed stations.

The considered satellites for this scenario are the SAC-C (with link budget parameters described in Tables 2.3 and 2.1) and VCUB1 (with link budget parameters described in Tables 3.1 and 3.2.

The ground stations are considered to track the satellites. That is, the maximum gain is used in the link budget evaluation. Also, the atmospheric losses are considered constant and equal to $P_{atm} = -1.6\,\mathrm{dB}$.

The problem is initially relaxed to be modeled as a linear convex one and goes through a Convex (CVX) Optimization. To refine this initial solution, two more algorithms are executed: a Sequential Least Squares (SQLQ), which improves the first solution, but still does not consider all constraints, and Differential Evolution (DE), that considers all constraints and refines even more the solution.

3.1 Problem Analysis

This section provides some notations used in algorithm descriptions.

Table 3.1: VCUB1 parameters.

Parameter	**Description**	**Value**
P_{EIRP}	Satellite Antenna EIRP	2 dB
f	Carrier frequency	2244 MHz
R_{spec}	Output data rate	10 Mbps
B	Bandwidth	6 MHz

Table 3.2: Ground antenna parameters for VCUB1.

Parameter	**Description**	**Value**
D	Antenna diameter	2.6 m
f	Carrier frequency	2244 MHz
e_a	Antenna aperture efficiency	0.5
G_{ground}	Maximum gain using Eq. (2.10)	32.7 dB
T_{sys}	Receiver noise temperature	24.94 dBK

3.1.1 Brazil Map

The Brazil map is obtained as a shapefile from [22]. The developed functions works for any territory. It is defined a rectangular grid of $M \times P$ points within the latitude and longitude bounds of the considered shapefile.

Two different grids are chosen for each of the used satellites. For VCUB1, it was used $M = P = 40$ and for SAC-C, $M = P = 50$. Increasing this grid would lead to higher computational costs, as a lot of RAM is necessary to represent all possible positions coverage.

Each of the $N_{grid} = MP$ points has an associated index i, which represents the possible positions for ground stations placement. This represents a convex set.

The script that produces this result is shown in Appendix B.6.

3.1.2 Array Notations

The link budget is evaluated for every possible antenna i in the grid and the resulting E_b/N_0 is stored in a matrix $a_i \in \mathbb{R}^{M \times P}$. If the antenna i is not inside Brazil, the associated a_i matrix is $0_{M \times P}$. Any element of a_i that corresponds to a point outside Brazil is also set to zero.

This matrix is then parsed into a binary matrix with 1 meaning that the value E_b/N_0 is above a threshold $(E_b/N_0)_{\text{min}}$ to establish the link and 0 meaning that is not. Therefore, the matrix a_i represents at which points inside Brazil the satellite can establish a communication link with the antenna located in position i. In other words, it represents the coverage pattern of the antenna i.

Then, each matrix a_i is reshaped in a vector of size $\mathbb{R}^{MP}$ and they are concatenated in a matrix $A \in \mathbb{R}^{N \times MP}$, where $N_{\text{grid}} = MP$ is the size of the grid. In this way, the new matrix A contains individual coverage patterns as its columns, with the The script that generates this matrix is shown in Appendix B.7.

It is also defined a binary vector $x \in \mathbb{R}^{N_{\text{grid}}}$, with 1 representing that there is an antenna in position i and 0 that there is not. Defining the problem with x as binary leads to better results than using the approach in [14], which employs a combination of l_1-norm and l_∞-norm to promote a binary sparse solution.

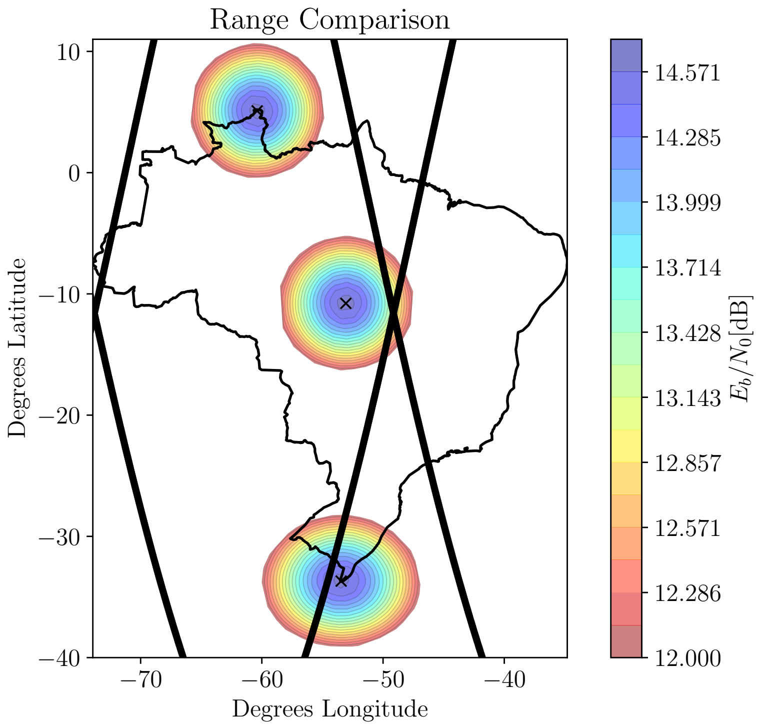

Figure 3.1: Ratio E_b/N_0 varying with longitude. From this it is possible to see the distortion of the antenna coverage in latitude and longitude as the ground stations are positioned in lower latitudes. The black lines represents the satellite trajectory.

The ratio E_b/N_0 varies with r, that is the distance between the transmitter and the receptor. From this, it is observed a distortion when projecting the coverage pattern on a latitude versus longitude graph. This is represented in Figure 3.1.

3.2 Convex Optimization

The objective is to optimize the antenna distribution while maximizing the Brazilian covered area using the minimum number of antennas.

Using the adopted notation, that means that the vector x must be sparse with few elements equal to one. These elements represent the chosen antenna positions. Hence, it is necessary to minimize the l_1-norm of x, that is $\|x\|_1 = \sum_{n=1}^{N_{\text{grid}}} |x_n|$. This leads to a sparser solution with the minimum of elements set to one [14]. Therefore, from this algorithm, both the number of antennas and the positions are provided as solutions.

The coverage of the antenna i is represented by a_i. The linear operation $x^T A$ gives the resulting coverage of the chosen antennas, which is the sum of the coverage matrices a_i.

Thus, the problem can be modeled as:

$$\begin{aligned} &\min \|x\|_1 \quad \text{subject to} \\ &\left\|x^T A - f_d\right\|_2 \leq \epsilon_u, \end{aligned} \tag{3.1}$$

where $\|y\|_2 = \sum_{n=1}^{N} |y_n|^2$ and $f_d \in \mathbb{R}^{MP}$ shares the same physical meaning as reshaped a_i and represents the desired overall pattern. That is:

$$f_d(i) = \begin{cases} 1, \text{if } i \text{ is inside Brazil} \\ 0, \text{otherwise} \end{cases} \tag{3.2}$$

This represents that is possible to establish a link in any point of the grid inside the Brazilian territory. In this scenario, ϵ_u is the acceptable error, that is, the positions not covered by any antenna pattern.

The problem formulated like this is linear with positive semi-definite matrices $x^T A$ and f_d. Therefore, it is convex.

To solve these kind of problems there are a lot of open source solvers available like the ones implemented in CVXPY [13] [17]. The one used in this work is SCIP [18], that is an open source mixed-integer nonlinear solver.

The solution of Eq. (3.1) only provides the initial solution of the problem. This modeling of the problem restrains the ground stations positions to the proposed grid. This represents nothing else but the relaxation of the problem. Thus, the problem has to be tuned to overcome this limitation.

3.3 Sequential Least Squares Optimization

The output obtained from the CVX algorithm serves as input for a SQLQ Optimization.

The solution of the convex optimization problem provides both the number and the position of the antennas. The SQLQ takes an initial antenna configuration with a fixed number of antennas and moves them around to get the optimal solution with minimized intersections.

The input is a matrix $y \in \mathbb{R}^{2\times Q}$, which contains the longitude and latitude of Q antennas.

Given a matrix y it is possible to determine how much area these Q antennas cover. This is represented by $S_{cov}(y)$ and is evaluated by getting the union of every antenna coverage area and subtracting its intersections.

It is also defined the quantity $S_{best}(y)$, which is the maximum potential coverage area achievable when all Q antennas are placed without intersections.

To achieve the proposed goal of maximizing the covered area, it is necessary to minimize the difference between $S_{best}(y)$ and $S_{cov}(y)$. In this way, the maximum area is covered with minimum intersections. In an ideal scenario, $S_{best}(y) = S_{cov}(y)$, which means that there are no intersections.

There is still one constraint to consider: the covered area must be the Brazilian territory, S_{Brazil}. This is achieved by considering that the intersection between $S_{cov}(y)$ and Brazil, $S_{\text{cov}}(x) \cap S_{\text{ Brazil}}$, must be higher than an acceptable parameter, S_d. This represents the percentage of Brazil that is covered.

Therefore, the problem can be modeled as:

$$\begin{aligned} &\min\left(S_{best}(y) - S_{cov}(y)\right) \quad \text{subject to} \\ &\frac{S_{\text{cov}}(y) \cap S_{\text{ Brazil}}}{S_{\text{Brazil}}} - S_d \geq 0. \end{aligned} \tag{3.3}$$

This problem is solved by the open source library SciPy [24].

An improvement over the last approach is that the antennas are not restrained to a grid. In this implementation, they can assume any position, which includes places outside the considered territory. Limiting the antennas to be inside the territory transforms the problem into a non-linear one. The next step, Differential Evolution, can include this constraint into the model.

Another observation is that the antenna coverage pattern projected into a latitude versus longitude grid is approximated to be a fixed circumference. This represents another drawback of this algorithm. However, this fact is also modeled by the Differential Evolution.

3.4 Differential Evolution

The result obtained from SQLQ serves as input to a DE algorithm. The main advantage of this algorithm is that it accommodates all problem constraints. It can also consider the real antenna coverage, instead of approximating them as circles. The disadvantage is that the solution obtained may not be optimal.

The method described in Section 2.5.3 must be adapted. The matrix $y_i^g \in \mathbb{R}^{2\times Q}$ is identically defined as the one in SQLQ. This means that as SQLQ, the DE generates position values for a fixed number Q of antennas inside the territory while maximizing the coverage.

The initial solution is usually randomly generated. In this case, the vector obtained by the SQLQ algorithm is included as one element of the first population and assigned as y_{best}, which is the best solution. The other $N_P - 1$ elements are random.

The mutation process is done as described in Section 2.5.3.

The crossover process is adapted as follows:

The mutated vector v_i^{g+1} is then combined with its parent y_i^g to generate u_i^{g+1} according to:

$$u_{i,j}^{g+1} = \begin{cases} v_{i,j}^{g+1}, \text{if two conditions are fulfilled} \\ y_{i,j}^{g}, \text{otherwise} \end{cases} \tag{3.4}$$

where the conditions are:

1. $v_{i,j}^{g+1}$ must be inside Brazil AND
2. A randomly generated number r_j must be less than the specified crossover probability (C_R) OR j is equal to j_{rand}, which is a random generated integer between 0 and Q.

Table 3.3: Results for the CVX Optimization.

${E_b}/{N_0}_{\min} = 10$								
ϵ_u	9	10	11	12	13	14	15	20
# Antennas	5	5	5	4	4	4	3	3
Coverage (%)	92.63	90.31	91.88	86.62	85.15	84.23	78.39	68.69
${E_b}/{N_0}_{\min} = 11$								
ϵ_u	9	10	11	12	13	14	15	20
# Antennas	7	6	6	6	5	5	5	4
Coverage (%)	94.07	90.98	91.72	86.39	84.43	85.80	85.66	67.31
${E_b}/{N_0}_{\min} = 12$								
ϵ_u	9	10	11	12	13	14	15	20
# Antennas	10	9	9	8	8	7	7	5
Coverage (%)	95.34	93.23	92.89	87.04	88.20	81.40	81.79	64.27

Mathematically, this can be expressed as:

$$\left(v_{i,j}^{g+1} \in \text{Brazil}\right) \wedge \left(r_j \leq C_R \vee j = j_{rand}\right) \tag{3.5}$$

This is how the constraint of keeping the antennas inside the territory is considered. An offspring will only pass to the next generation if it is inside the territory.

Finally, to analyze if the generated offspring are better than its parents, the cost function gives the percentage of Brazil area that is not covered.

It was tested a method of multiple offspring generation as described in [11]. However, the algorithm time has highly increased and it was not observed a better performance than the usual implementation. Therefore, the last one was chosen.

3.5 Simulations and Results for SAC-C

The simulation considers 3 scenarios for SAC-C, one for each value of the threshold $(E_b/N_0)_{\min}$: 10 , 11 and 12 dB. This implies antennas with 3 different coverage patterns.

3.5.1 Convex Optimization

To solve this problem it is necessary to select the parameter ϵ_u, presented in Section 3.2. The choice of the acceptable error ϵ_u depends on the dimension of the grid and affects how much area is covered by antennas. Ideally, this parameter should be zero. However, the lower the value of ϵ_u, the more time the algorithm takes to converge and if it is too small, the algorithm becomes unfeasible. For the proposed 50×50 coarse grid, a value of $\epsilon_u = 9$ is used, which is approximately equivalent to 1% of the points within the Brazilian territory. Simulations results are shown in Table 3.3.

The problem is very sensitive to the control variable ϵ_u, as it is highly non linear.

Table 3.4: Results for the SQLQ Optimization.

$E_b/N_{0\min}$	Antennas	CVX Coverage	SQLQ Coverage
10	5	92.63%	98.03%
10	6	95.68%	95.80%
11	7	94.07%	99.01%
11	8	96.14%	98.99%
12	10	95.34%	98.48%
12	11	96.26%	99.00%

As it is desired to maximize the coverage, in addition to the CVX optimal results, one variation is also considered as input for SQLQ. For each set of antennas, one antenna is added. This additional antenna is placed in the middle point of two existing ones. As the territory geometry is not convex, if the middle point is outside Brazil it is snapped into the nearest inside point. The antennas are combined two by two, the one that provides the higher increase in coverage is chosen.

This algorithm is implement by the script in Appendix B.8.

3.5.2 Sequential Least Squares

To solve this problem it is necessary to choose the parameter S_d, which represents the desired territory coverage, and provide an initial value x_0. The algorithm is very dependant on the initial condition. However, the one provided is from the CVX, which is the optimal solution considering that the antennas are in a grid and the chosen parameter ϵ_u. It is considered $S_d = 0.99$, i.e., the target is 99% of territory coverage. The results are shown in Table 3.4. These will serve as input for DE. The implementation of this algorithm is displayed in Appendix B.9.

3.5.3 Differential Evolution

Before executing DE, it is necessary to parse the solution obtained by SQLQ. If there is any antenna that is placed outside the territory, this antenna is moved to the nearest point that is inside Brazil.

The algorithm is initially executed considering that the antennas range is fixed, that is, the coverage pattern does not depends on the geodetic coordinates of the ground stations. This simplification is used because computing the actual coverage for every iteration is computational costly. After this first run, the algorithm is executed computing the real coverage each iteration. However, in this second run, it runs for fewer generations as the solution is already acceptable and this final step is for refining purposes.

The algorithm uses the parameters shown in Table 3.5, that are commonly used values for

Table 3.5: Parameters for DE Algorithm.

	F_α	F_β	C_R	N_P	G
Fixed Range	0.3	0.8	0.9	500	300
Variable Range	1	0.6	0.9	500	50

Table 3.6: Results for the DE Algorithm.

$E_b/N_{0\text{min}}$	Antennas	Fixed Range	Variable Range
10	5	98.04%	98.36%
10	6	99.90%	99.85%
11	7	99.56%	99.42%
11	8	99.97%	99.95%
12	10	98.54%	98.38%
12	11	99.27%	99.13%

DE implementations. The results are shown in Table 3.6. The implementation of this algorithm is displayed in Appendix B.10.

3.5.4 Overall Results for SAC-C

Figure 3.2 shows the results obtained from CVX are close to the final one despite the grid-restrained positions. It is possible to see the refinement of the solution as the algorithms are executed.

The addition of one antenna in the CVX's optimal result shows minimal improvement, as presented in Figure 3.3. The solution found by CVX is a good compromise between covered area and number of antennas, as the highest improvement found is in case of $(E_b/N_0)_{\text{min}} = 10$, which only provides around 1% more coverage, while also increasing the intersection percentage by more or less 10%.

3.6 Simulations and Results for VCUB1

The value used in simulation is $(E_b/N_0)_{\text{min}} = 7.5\,\text{dB}$, because this is the minimum value necessary for establishing a link connection with the VCUB1.

3.6.1 Convex Optimization

For this algorithm, the followed procedure is the same as in Section 3.2. The results are shown in Table 3.7 and the covered for $\epsilon_u = 8$ is shown in Figure 3.4.

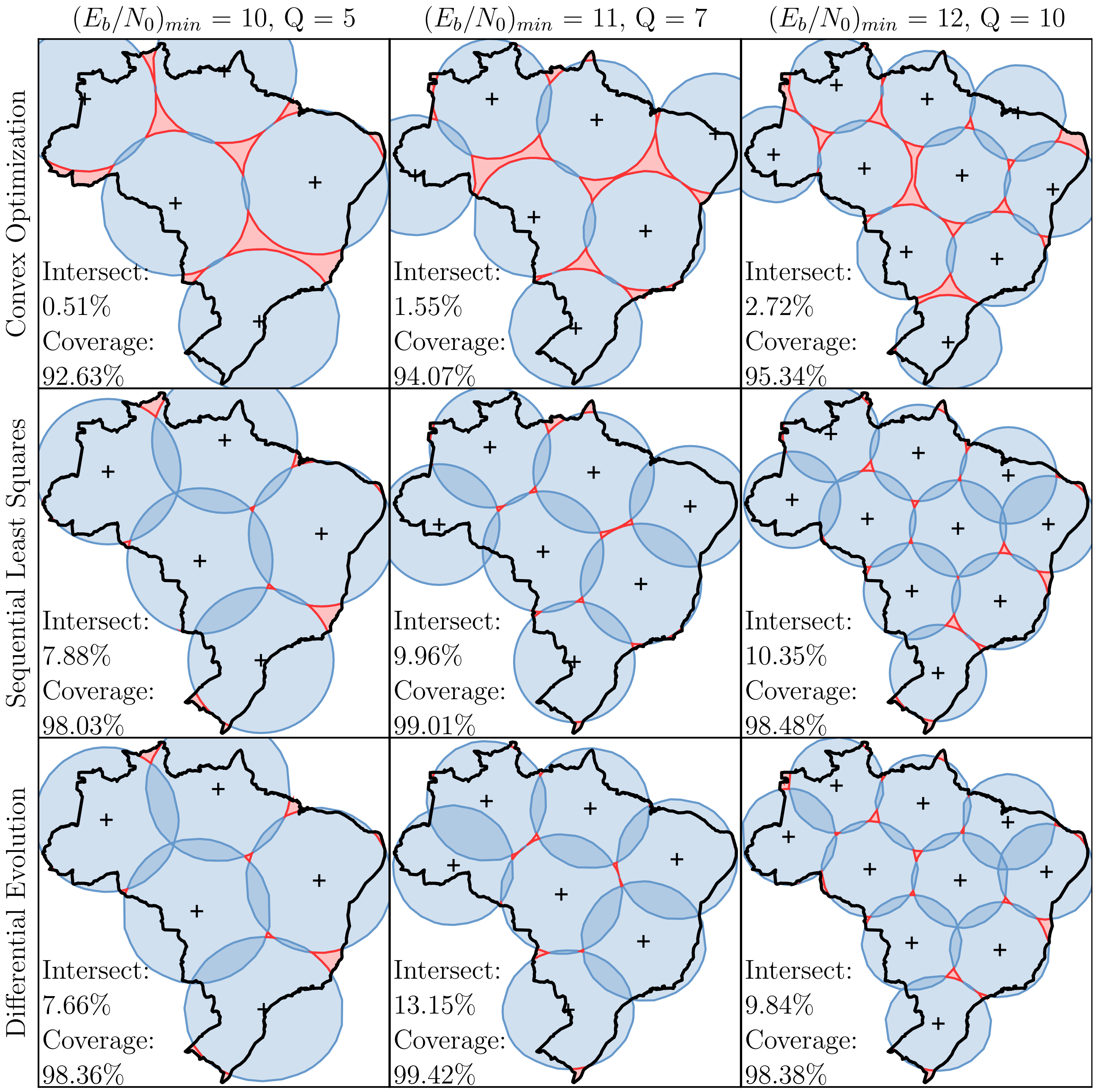

Figure 3.2: Simulation results using from Convex Optimization result of for SAC-C. For each scenario, the same number of ground stations found in the convex optimization was used in the other two steps of the algorithm to improve the final coverage. This shows that the initial solution proposed by the convex optimization is close to the final obtained despite of the initial problem relaxation.

Table 3.7: Results for the CVX Optimization for the VCUB1.

$^{E_b}/_{N_0}{}_{\min} = 7.5$					
ϵ_u	7	8	9	10	20
# Antennas	6	6	5	5	2
Coverage (%)	94.71	96.04	89.79	91.10	48.74

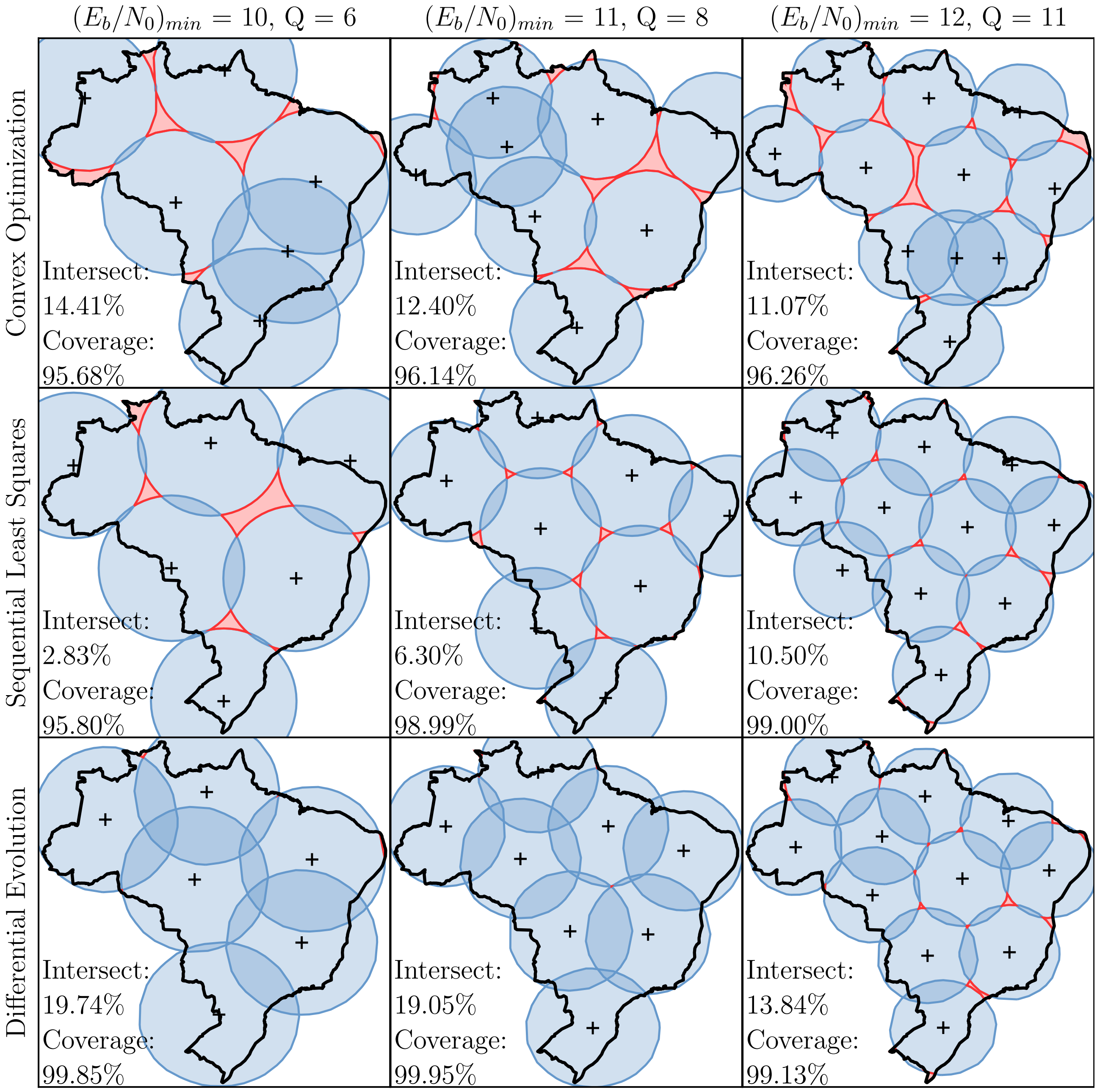

Figure 3.3: Simulation results using one additional an antenna to CVX optimal result for SAC-C. For each scenario, one antenna was added to the number of ground stations found in the convex optimization and then they are used in the following steps. From this, it is possible to see that the initial solution proposed by the convex optimization, presented in Figure 3.2, is a great compromise between the covered area and the intersections.

Table 3.8: Results for the SQLQ Optimization for VCUB1.

$E_b/N_{0\min}$	Antennas	CVX Coverage	SQLQ Coverage
7.500000	6	96.01%	98.28%
7.500000	7	97.55%	99.87%

Table 3.9: Parameters for DE Algorithm for the VCUB1 ground station positions.

	F_α	F_β	C_R	N_P	G
Fixed Range	0.05	0.1	0.9	500	300
Variable Range	0.05	0.1	0.9	100	25

3.6.2 Sequential Least Squares

For this algorithm, the followed procedure is the same as in Section 3.5.2. The results are shown in Table 3.8 and in Figures 3.4 and 3.5.

3.6.3 Differential Evolution

For this algorithm, the followed procedure is the same as in Section 3.4. The parameters for the algorithm are shown in Table 3.9 and results are shown in Table 3.10 and in Figures 3.4 and 3.5.

3.6.4 Overall Results for VCUB1

In a similar way of the results of SAC-C, Figure 3.4 shows that the results obtained from the convex optimization are close to the final one despite the grid-restrained positions. The SQLQ tuned these results leading to a better coverage and then the DE refined it even more.

The addition of one antenna in the convex optimization result shows minimal improvement, as presented in Figure 3.5. Again, the solution found by convex optimization is proven to be the optimal compromise between covered area and number of antennas, as the improvement provides only around 2% more coverage, while also increasing the intersection percentage by more or less 5%.

Table 3.10: Results for the DE Algorithm for the VCUB1 ground station positions.

$E_b/N_{0\min}$	Antennas	Fixed Range	Variable Range
7.5	6	98.67%	98.64%
7.5	7	99.99%	100%

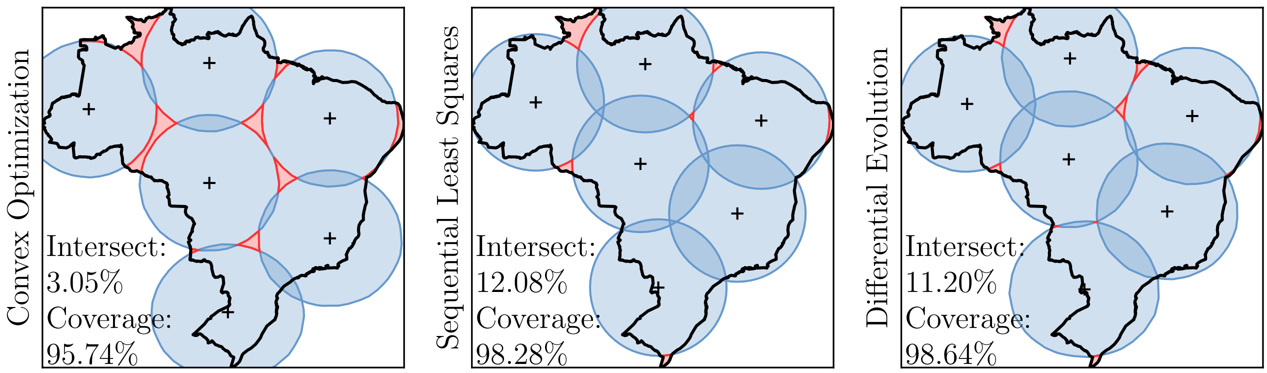

Figure 3.4: Simulation results using from CVX optimal result for VCUB1. The same number of ground stations found in the convex optimization was used in the other two steps of the algorithm to improve the final coverage. This shows that the initial solution proposed by the convex optimization is close to the final obtained despite of the initial problem relaxation.

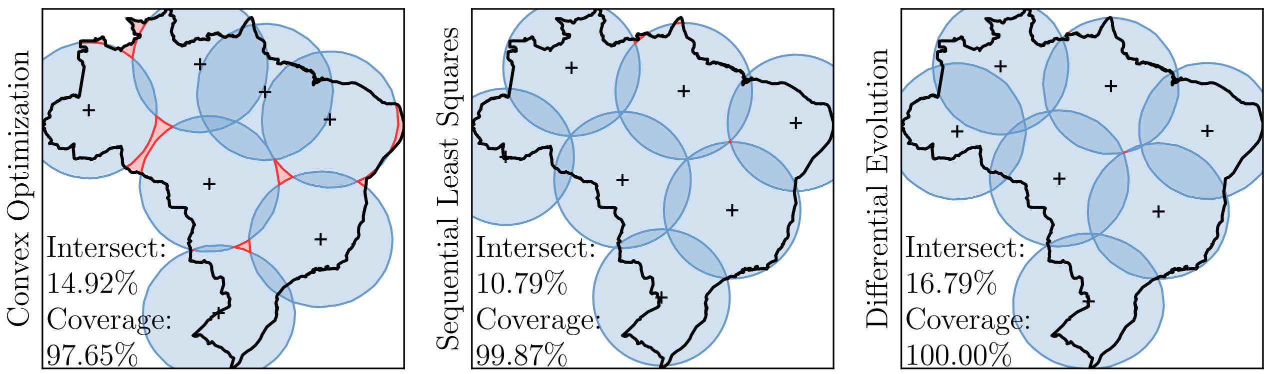

Figure 3.5: Simulation results using one additional antenna to CVX optimal result for VCUB1. One antenna was added to the number of ground stations found in the convex optimization and then they are used in the following steps. This shows that the initial solution proposed by the convex optimization, shown in Figure 3.4, represents a good compromise between the number of antennas and the intersections.

3.7 Parabolic Reflectors with Different Diameters

Another approach to the problem of ground station distribution is to consider that the antennas cannot be placed anywhere. They should be in specific areas due to a legacy structure, for instance. In this scenario, the possible ground station sites are fixed and the antennas themselves are variables.

3.7.1 Sequential Least Squares

For this scenario, it was used an adaptation of the previous algorithm discussed in Section 3.3.

The SQLQ takes an initial antenna configuration with a fixed maximum number of antennas, Q, and modifies its coverage pattern to get the optimal solution with minimized intersections.

The coverage pattern of the antennas are circles with a range that is dependant on the antenna diameter. The minimum and maximum antenna diameter, $D_{\min}$ and $D_{\max}$, leads to the minimum and maximum coverage patterns range. This works as boundaries for the the SQLQ.

The input is a vector $r \in \mathbb{R}^Q$, which contains the range of the Q antennas.

Given a vector r and the antenna locations it is possible to determine how much area these Q antennas cover, as well as their intersections. This is represented by $S_\cup(r)$ and is evaluated by getting the union of every antenna coverage area and subtracting its intersections, $S_\cap(r)$.

It is also defined the quantity S_{best}, which is the maximum potential coverage area achievable when all Q antennas have the maximum possible diameter.

To achieve the proposed goal of maximizing the covered area, it is necessary to minimize the difference between S_{best} and $S_\cup(r)$. It is also desired to minimize intersections.

There is still one constraint to consider: the covered area must be the Brazilian territory, S_{Brazil}. This is achieved by considering that the intersection between $S_\cup(r)$ and Brazil, $S_\cup(r) \cap S_{\text{ Brazil}}$, must be higher than an acceptable parameter, S_d. This represents the percentage of Brazil that is covered.

Therefore, the problem can be modeled as:

$$\begin{aligned} &\min \left[C_1 \left(S_{best} - S_\cup(r)\right) + C_2 \left(S_\cap\right)\right] \quad \text{subject to} \\ &\frac{S_\cup(r) \cap S_{\text{ Brazil}}}{S_{\text{Brazil}}} - S_d \geq 0. \end{aligned} \tag{3.6}$$

There is another variation of the problem, instead of using the coverage as a circle, it uses the real coverage of each station. In this scenario, the vector r has the possible parabolic antenna diameters. In the same way, from the vector r it is possible to evaluate the real coverage of each station, their union and intersection. The problem is modeled in the same way as Eq. (3.6), but $S_\cup(r)$ and $S_\cap(r)$ are evaluated considering the actual coverage of the ground stations.

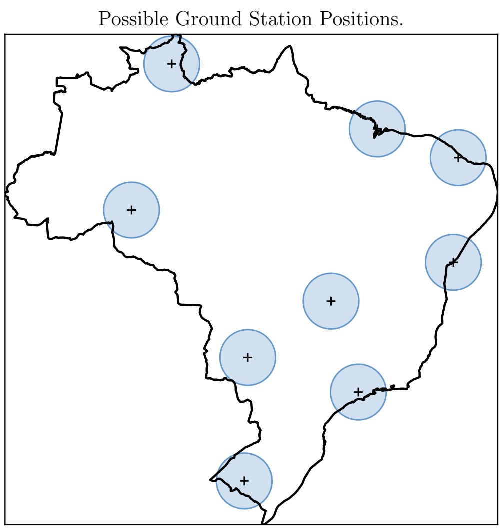

Figure 3.6: Positions for the $Q = 9$ possible ground stations that are used in the algorithm described in Section 3.7.1. The chosen criteria for these locations are capitals or cities with enough infrastructure to receive an antenna site.

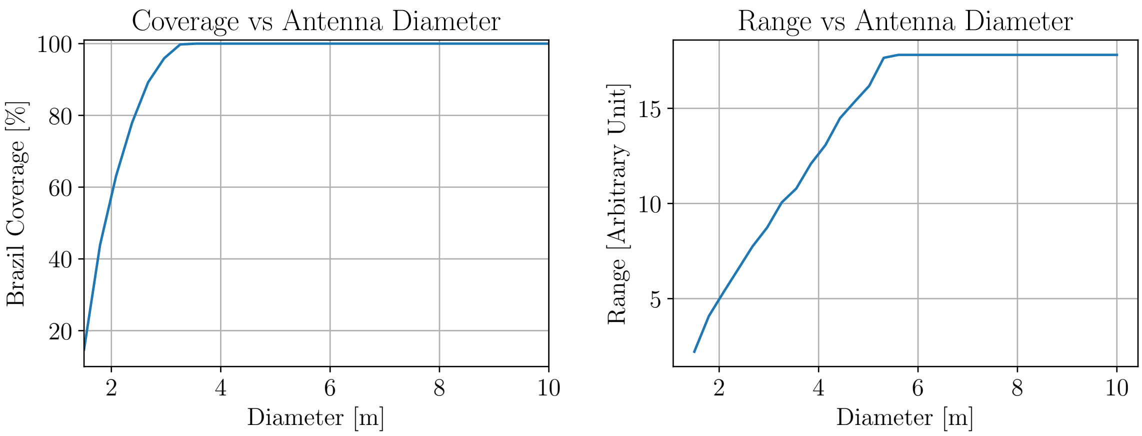

Figure 3.7: Antenna diameter impact on range and on Brazil coverage. From this, it is possible to choose a maximum diameter of $D_{\max} = 6\,\mathrm{m}$, as a higher diameter would not increase neither the coverage nor the range.

3.7.2 Simulation Results

For this scenario, it is considered $Q = 9$ ground stations in the positions shown in Figure 3.6. The link budget is analyzed considering the VCUB1 characteristics. The variable parameters are only the ground antenna diameters. The control variables for the problem are $C_1 = 0.5$, $C_2 = 0.5$ and $S_d = 0.99$.

To evaluate the impact of antennas with different diameters, it is possible to see how the diameters impact their individual coverage and the total coverage. This result is shown in Figure 3.7.

Based on this, the maximum is $D_{\max} = 6\,\mathrm{m}$. The simulation result is shown in Figure 3.8. Only 3 out of the initial 9 antennas were used. The antennas that are not used are set to $D_{\min}$

Table 3.11: Results for the Parabolic Reflectors with Different Diameters for the VCUB1. The used ground stations are in bold.

Latitude [°]	Longitude [°]	Diameter [m]
-48.05	-15.99	0
-45.88	-23.21	0
-54.94	**-30.27**	**6.00**
-44.37	-2.32	2.19
-37.94	**-4.60**	**6.00**
-60.70	**2.85**	**6.00**
-63.90	-8.76	0
-54.66	-20.46	0
-38.33	-12.91	0

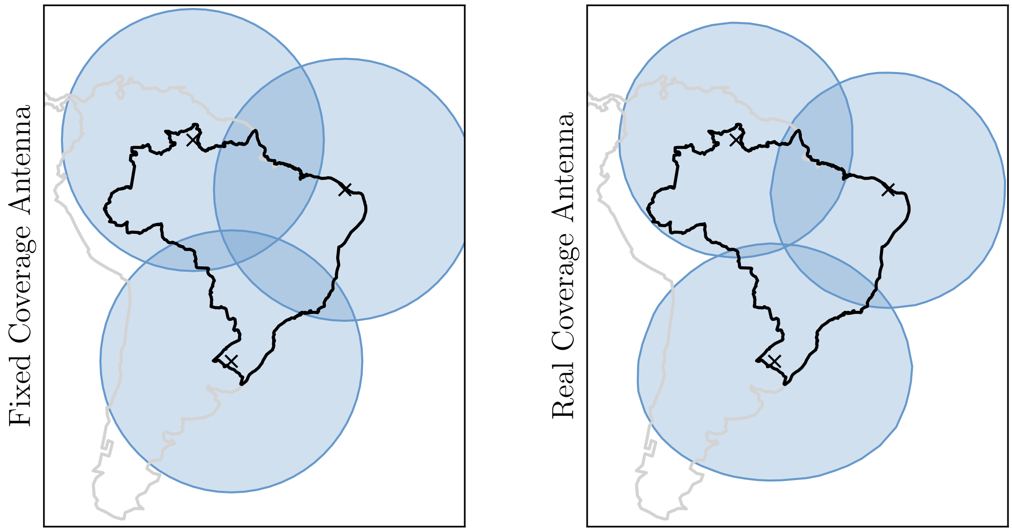

Figure 3.8: Results for the algorithm described in Section 3.7.1. On the left, it is the result considering that the antenna coverages are circular. On the right, it is the result considering the deformation in the coverages.

by the algorithm. In addition, the ground stations with more than 50% of its area overlapping other antenna coverage are not considered.

The results found considering the antenna coverage approximated by a circle are used as input for the algorithm that considers the real coverage of each antenna. However, when trying to find a better solution, the algorithm does not find any direction towards the gradient is negative. Thus, it converges to the same solution as the simpler scenario.

The script that implements this algorithm is found in Appendix B.11.

3.7.3 Downlink Capabilities of the Proposed Stations

Concluding this analysis of ground station positioning, it is evaluated how much data is possible to be downloaded from the satellite in comparison with a ground station near the pole.

To this calculation, it is considered how much time the satellite is in contact with the ground station with the ratio $E_b/N_0 > 7.5\,\mathrm{dB}$.

It is considered an antenna in Comandante Ferraz Antarctic Station, which is the Brazilian station on the south pole. The results are shown in Figure 3.9.

The ground station located on the Brazilian mainland has virtually the same capability of downlink as the station in Comandante Ferraz, as shown in Figure 3.9, with significantly fewer resources necessary for the maintenance of these stations.

The station outside of the Brazilian territory is useful for other reasons than just downlink capabilities. Considering the case of an Earth Observation satellite that cannot receive commands while imaging, and that the main necessities of imaging are inside Brazil, it is interesting to send commands to the satellite outside of the area of interest. However, the proposed scenario is still useful considering that the satellite payload is an optical imaging system, as in this case the night passes can be used for downlink and imaging planning, while the day passes can be used for imaging purposes.

Also as a matter of comparison, a station placed in Svalbard, Norway, is capable of downloading 48.55 Gbits/day. The proposed system has approximately 65% of the download capability of Svalbard.

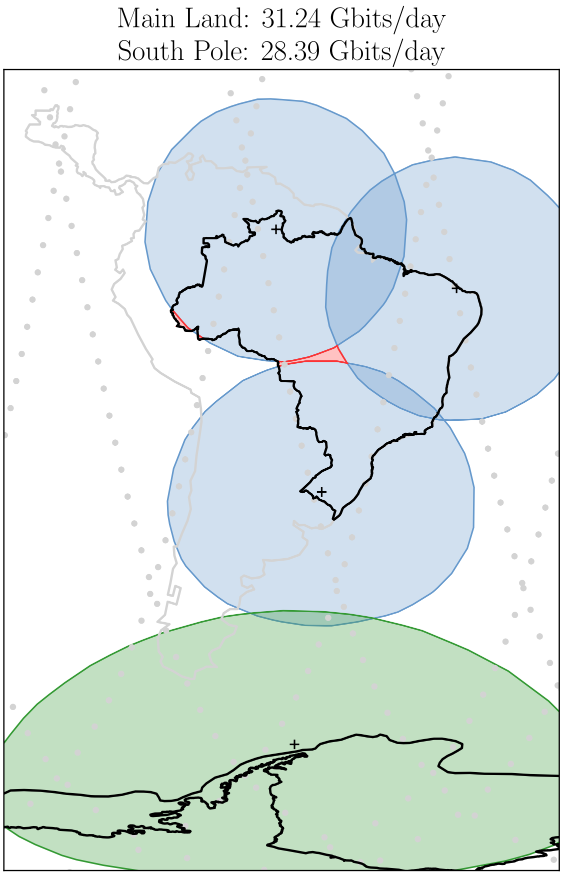

Figure 3.9: Comparison between the obtained ground station on mainland Brazil configuration against a station placed near the South Pole. It has virtually the same capability to download data. The gray points represent some of the satellite coordinates.

Part IV

Antenna Array Design

Chapter 4

Antenna Array Design

The Appendix A presents the classical methods of designing an array, which, in general, is useful to project uniform arrays varying only their input magnitude and phase.

This chapter proposes a design process of a static antenna array with the antennas physically steered and in any 3D position. This introduces more degrees of freedom to the problem when compared to more classical approaches.

4.1 Physically Steered Array

First, it is proposed a method to combine the field of one element into an array.

A generic array with N elements can be described by its global Cartesian positions $(d_0, d_1, \ldots, d_{N-1})$, its feed coefficients $(a_0, a_1, \ldots, a_{N-1})$, its local rotations around x-axis $(\alpha_0, \alpha_1, \ldots, \alpha_{N-1})$, around y-axis $(\beta_0, \beta_1, \ldots, \beta_{N-1})$ and around z-axis $(\gamma_0, \gamma_1, \ldots, \gamma_{N-1})$.

For this generic array, the resultant fields E_θ and E_ϕ are given by:

$$
\begin{aligned}
E_\theta &= \sum_{n=0}^{N-1} a_n E_\theta[n] e^{j\boldsymbol{k}\cdot\boldsymbol{d}_n}, \text{ and} \\
E_\phi &= \sum_{n=0}^{N-1} a_n E_\phi[n] e^{j\boldsymbol{k}\cdot\boldsymbol{d}_n},
\end{aligned}
\tag{4.1}
$$

where $\boldsymbol{k} = \frac{2\pi}{\lambda}\hat{r} = \frac{2\pi}{\lambda}\begin{bmatrix}\cos\phi\sin\theta \\ \sin\phi\sin\theta \\ \cos\theta\end{bmatrix}$ and $E_\theta[n], E_\phi[n]$ are evaluated using the local angles α_n, β_n, γ_n and then rotated to the global axis using the method presented in Section 2.2.

To validate the method, it is first considered a non rotated array with $N = 3$ elements placed along the x-axis: $d_0 = [-\lambda/2, 0, 0]$, $d_1 = [0, 0, 0]$ and $d_2 = [\lambda/2, 0, 0]$. This array is compared with a simulation in HFSS representing the same array. The results are shown in Figure 4.1.

Also as a form of validation, it is proposed another array with $N = 3$ elements placed along the x-axis: $d_0 = [-\lambda/2, 0, 0]$, $d_1 = [0, 0, 0]$ and $d_2 = [\lambda/2, 0, 0]$. But in this scenario, they are rotated around the y-axis: $\beta_0 = \beta_1 = \beta_2 = 45°$. Again, this array is compared with a simulation in HFSS representing the same array. The results are shown in Figure 4.2.

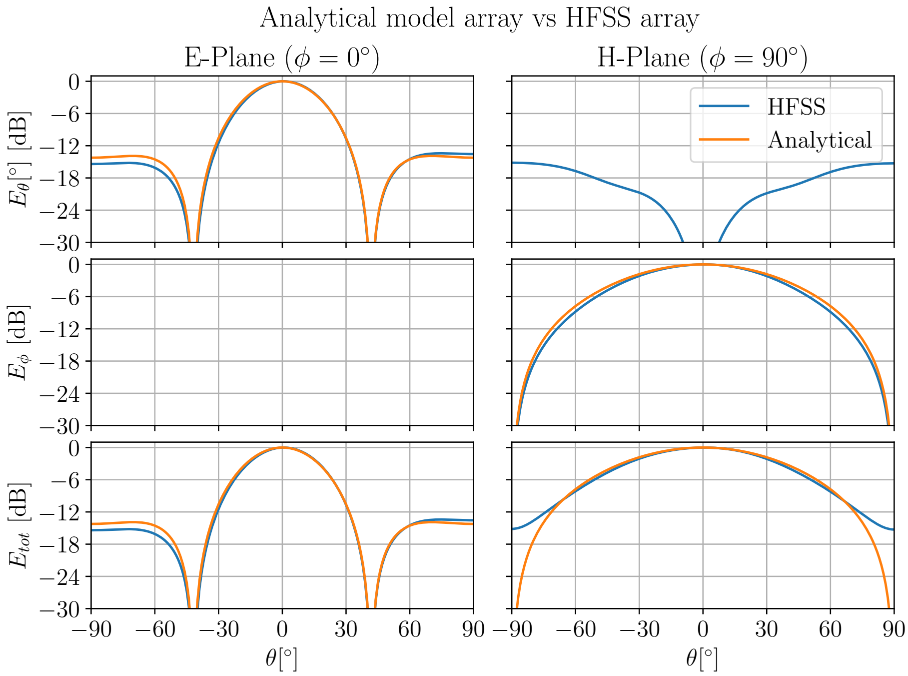

Figure 4.1: E-plane and H-plane for the microstrip antenna array equations compared with a simulation in HFSS. This shows that the implemented array equations are close enough to the HFSS array simulation. The differences are justifiable by the simplification of the microstrip fields model.

In both cases, the results of the proposed model for the MicroStrip antenna are very similar, with the differences being justifiable by the simplification of the equations presented in Section 2.1.3. To overcome this, another scenario is considered. Instead of using the proposed equations, the fields of a MicroStrip antenna are extracted from HFSS and these fields are combined and rotated into arrays by the proposed equations.

To replicate the first result, it is considered the field pattern of the non-rotated patch obtained from HFSS and this pattern is combined to form an array, which is compared against the same array simulated in HFSS. The result is in Figure 4.3.

Then, to replicate the second result, it is proposed another scenario: the patch is analytically rotated of $\beta = 45°$ around the y-axis and combined into an array. The result is compared against the rotated array patch simulated on HFSS and is shown in Figure 4.4. These figures are obtained using the script in Appendix B.12.

4.2 Validation Model

For the validation model, the considered array element is the one presented in Figure 4.5, which is a Yagi Uda antenna with 4 elements designed by [20]. This antenna is then modeled with spherical modes using AFTK.

To validate the algorithm, it is proposed the following objective array:

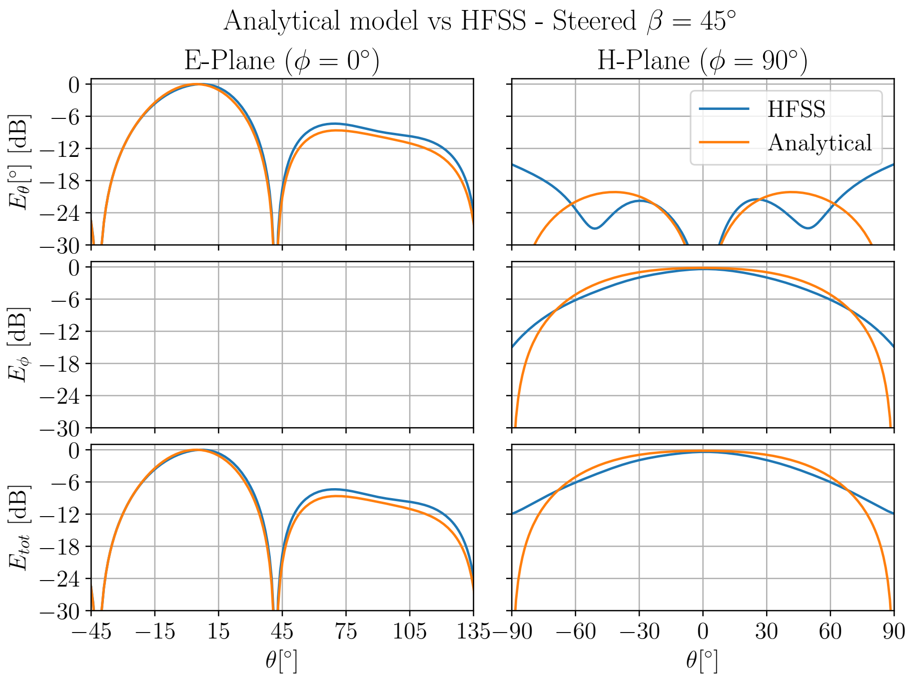

Figure 4.2: E-plane and H-plane for the microstrip antenna array steered of $\beta = 45°$ around the y-axis using the presented equations compared with a simulation in HFSS. This shows that the implemented steered array equations are close enough to the HFSS steered array simulation. The differences are justifiable by the simplification of the microstrip fields model.

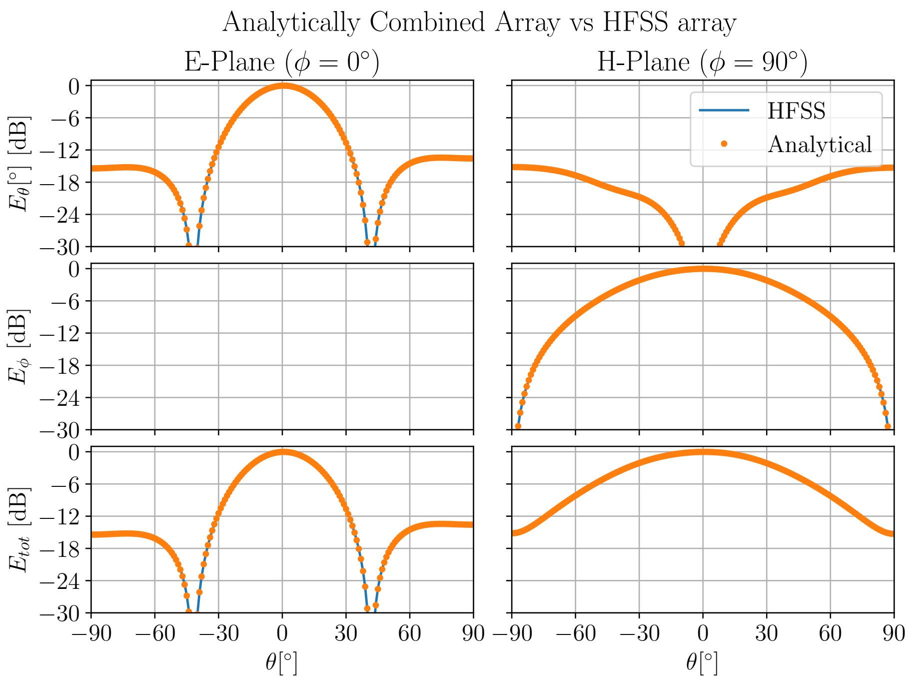

Figure 4.3: E-plane and H-plane for the microstrip antenna using field patterns from HFSS and then analytically combined into an array compared with a simulation in HFSS. This scenario eliminates the simplifications on the microstrip model and evaluates only the equations that implements the array. As the result is virtually the same as the HFSS simulation, the proposed model is validated.

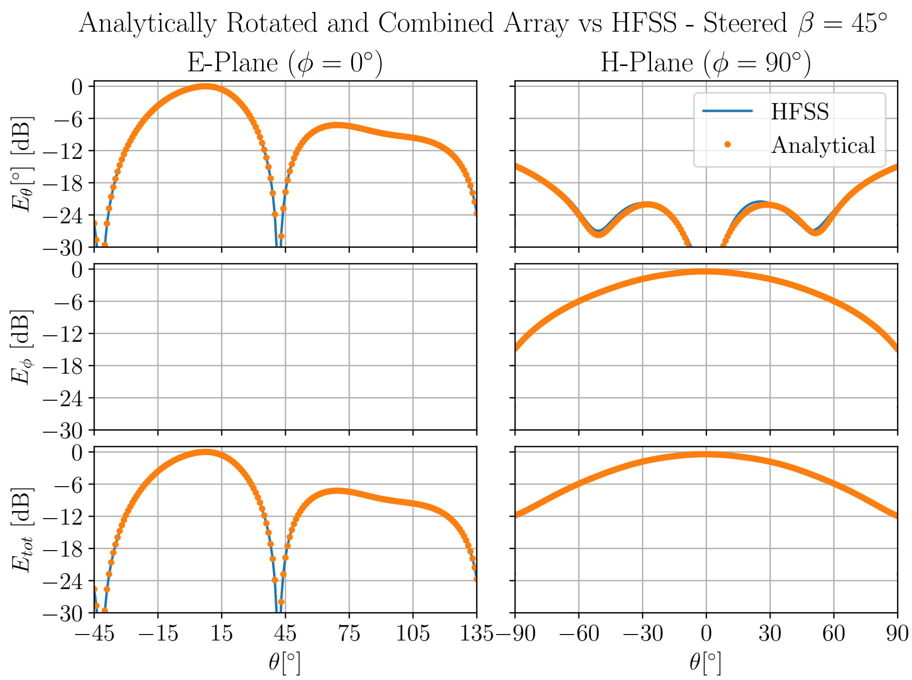

Figure 4.4: E-plane and H-plane for the microstrip antenna analytically rotated of $\beta = 45°$ around the y-axis using field patterns from HFSS analytically combined into an array compared with a simulation in HFSS. This scenario eliminates the simplifications on the microstrip model and evaluates only the equations that implements the steered array. As the result is virtually the same as the HFSS simulation, the proposed model is validated.

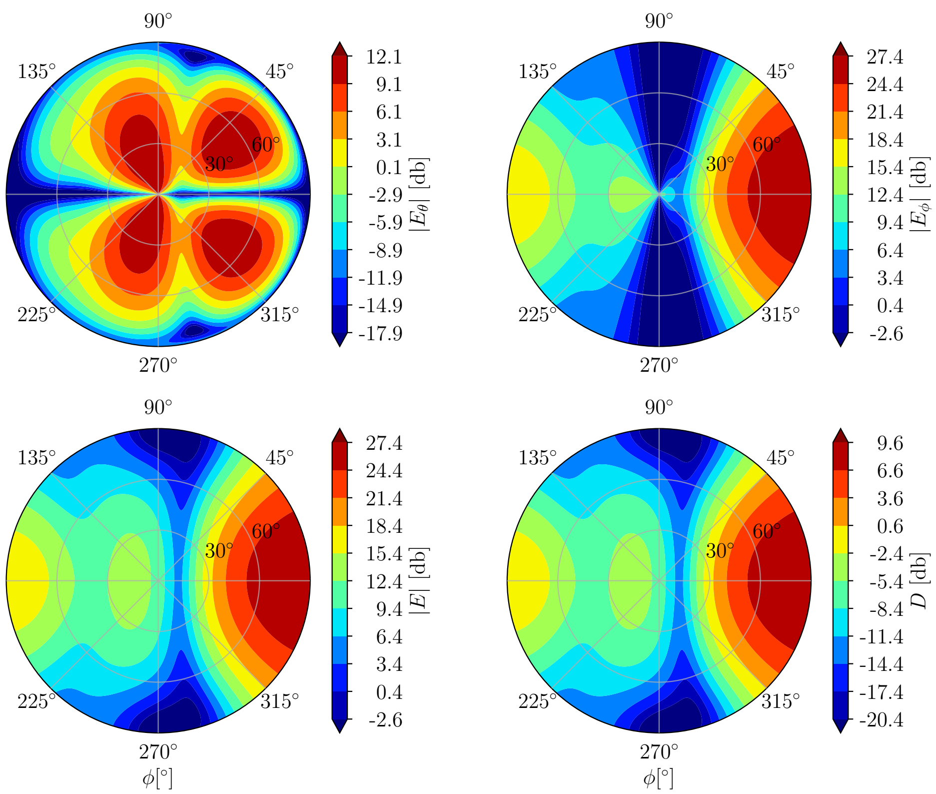

Figure 4.5: Electric fields and directivity for one array element, which is an Yagi Uda antenna with 4 elements designed by [20] and modeled with spherical modes using AFTK.

$$\mathbf{d} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda/2 & \lambda \\ 0 & 0 & 0 \end{bmatrix}, \tag{4.2}$$

where each column represents the coordinates $d_n = (x_n, y_n, z_n)$ of the n-th element.

The elements are rotated of:

$$\boldsymbol{\psi} = \begin{bmatrix} 0^\circ & 0^\circ & 0^\circ \\ -90^\circ & -90^\circ & -90^\circ \\ 55^\circ & 55^\circ & 55^\circ \end{bmatrix}, \tag{4.3}$$

where each column represents the angles $\psi_n = (\alpha_n, \beta_n, \gamma_n)$ of the n-th element.

The input array for the algorithm is slightly offset from the objective:

$$\mathbf{d} = \begin{bmatrix} 0 & 0.07\lambda & 0.14\lambda \\ 0 & 0.8\lambda & 1.6\lambda \\ 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{\psi} = \begin{bmatrix} 0^\circ & 0^\circ & 0^\circ \\ -90^\circ & -90^\circ & -90^\circ \\ 90^\circ & 30^\circ & 110^\circ \end{bmatrix}. \tag{4.4}$$

The field pattern is analyzed in $0 \leq \theta \leq 180^\circ$ and for $\phi = 90^\circ$.

The objective is to optimize the positions and rotations of N antennas to match the objective array pattern of far-field E_θ and E_ϕ.

It is possible to write this problem as:

$$\begin{gathered} \min \sum_{\theta=0}^{\pi} \left[\| E_\theta(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi}) - E_{\theta_d}(\theta, \phi) \|_2 + \| E_\phi(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi}) - E_{\phi_d}(\theta, \phi) \|_2 \right] \quad \text{subject to} \\ \delta - \sum_{\theta=-\pi}^{\pi} \left[\| E_\theta(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi}) + E_{\theta_d}(\theta, \phi) \|_2 - \| E_\phi(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi}) - E_{\phi_d}(\theta, \phi) \|_2 \right] \geq 0, \end{gathered} \tag{4.5}$$

where δ is the acceptable error. This problem is solved using the Sequential Least Squares algorithm.

The obtained positions and rotations are displayed in Table 4.1. As the problem is non-convex, the algorithm converged to a solution that is not equal to the initially proposed array. However, it is possible to see a successfully convergence, as the cost profile shown in Figure 4.6 converges. Also, as shown in Figure 4.7, the projected array fields match the obtained fields. This algorithm is implemented by the script displayed at Appendix B.13.

4.3 Field Pattern Design

For this section, it is also considered the element array presented in Figure 4.5.

The objective of this section is to propose an array capable of maintaining a approximately constant field strength over a desired range of θ.

Table 4.1: Results of the Sequential Least Squares algorithm. The values of x, y and z are in multiple of λ.

	$x[\lambda]$	$y[\lambda]$	$z[\lambda]$	$\alpha[°]$	$\beta[°]$	$\gamma[°]$
0	0.16	-0.50	0.00	-18.49	-90.00	36.51
1	0.16	0.00	0.00	-14.63	-90.00	40.37
2	0.16	0.50	0.00	-4.12	-90.00	50.88

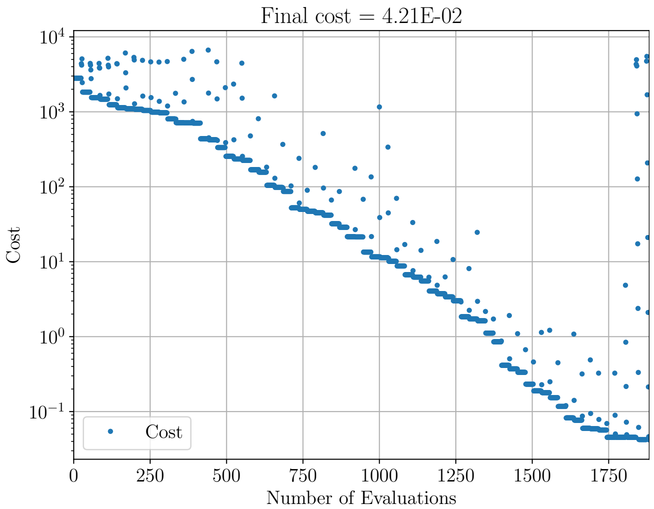

Figure 4.6: Sequential Least Squares cost profile for the validation model algorithm. This shows a convergence, as the cost is reducing as the iterations increases.

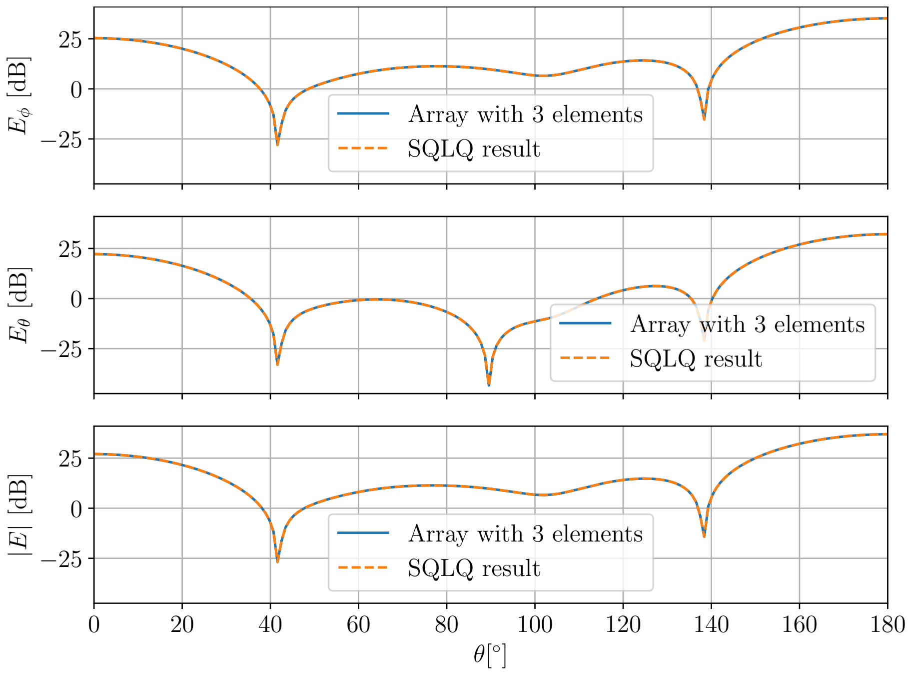

Figure 4.7: Field comparison between the projected array and the obtained from the Sequential Least Squares optimization. The field profile is virtually the same as the projected array, even though the found optimal array is not the same as the target one in terms of position and rotation of the array elements. This shows that the problem has multiple solutions.

Table 4.2: Obtained array for the plateau design. The values of x, y and z are in multiple of λ.

	$x[\lambda]$	$y[\lambda]$	$z[\lambda]$	$\alpha[°]$	$\beta[°]$	$\gamma[°]$
0	0.00	2.95	0.00	90.00	-89.82	-90.00
1	0.00	2.81	0.51	64.32	-90.00	-64.32
2	0.00	-2.81	0.19	-2.73	-90.00	86.84

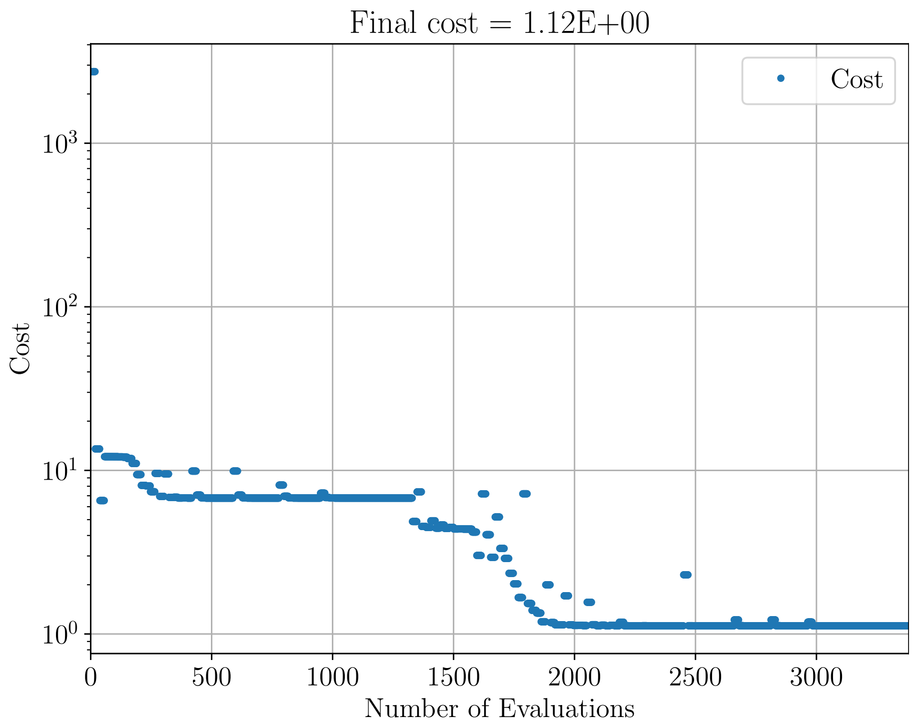

Figure 4.8: BFGS cost profile for the plateau pattern design with $\theta_{\max} = 70°$. It can be seen the algorithm convergence, as the cost reduces with the iterations.

It is possible to model this problem by adapting the algorithm presented in Section 4.2 as follows:

$$\min \frac{\max \|E_\theta(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi})\|}{\min \|E_\theta(\theta, \phi, \boldsymbol{d}, \boldsymbol{\psi})\|}. \tag{4.6}$$

For this problem, the chosen algorithm was BFGS.

For testing purposes it is proposed to find an array that has a plateau from $\theta = 0°$ to $\theta = 70°$. The obtained array is shown in Table 4.2.

Figure 4.8 shows the cost profile and Figure 4.9 shows that the initial objective was accomplished.

When repeating the same procedure considering more antennas, the found result is more oscillatory, as shown in Figures 4.10 and 4.11. However, the power level increases as there are more elements.

This algorithm is implemented by the script displayed in Appendix B.14.

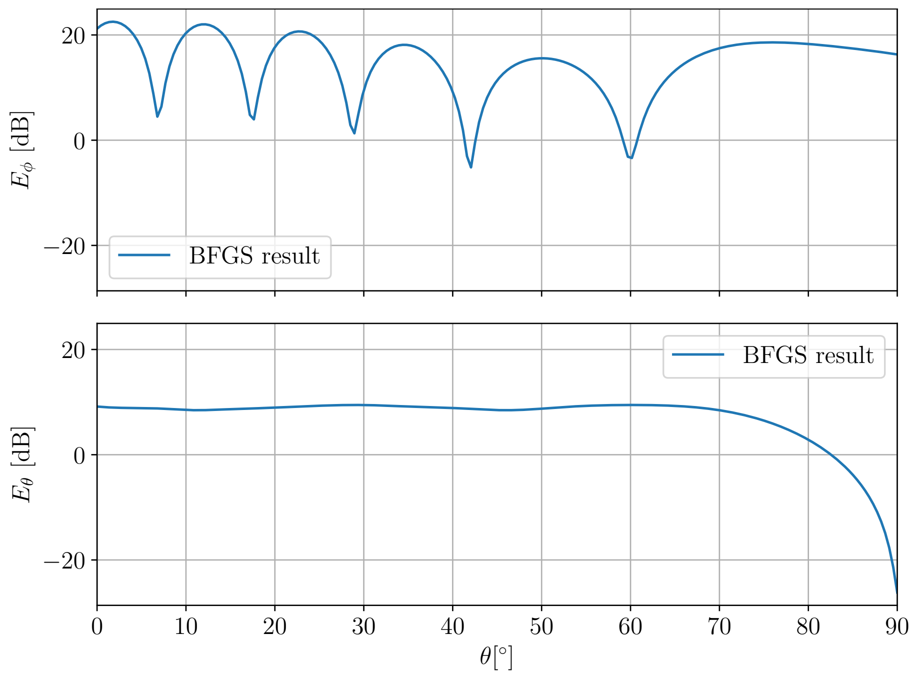

Figure 4.9: Fields obtained by the BFGS algorithm with 3 elements in array for the plateau pattern design in $|E_\theta|$ with $\theta_{\text{max}} = 70°$. The algorithm successfully converged to a solution with a plateau over the desired values of θ.

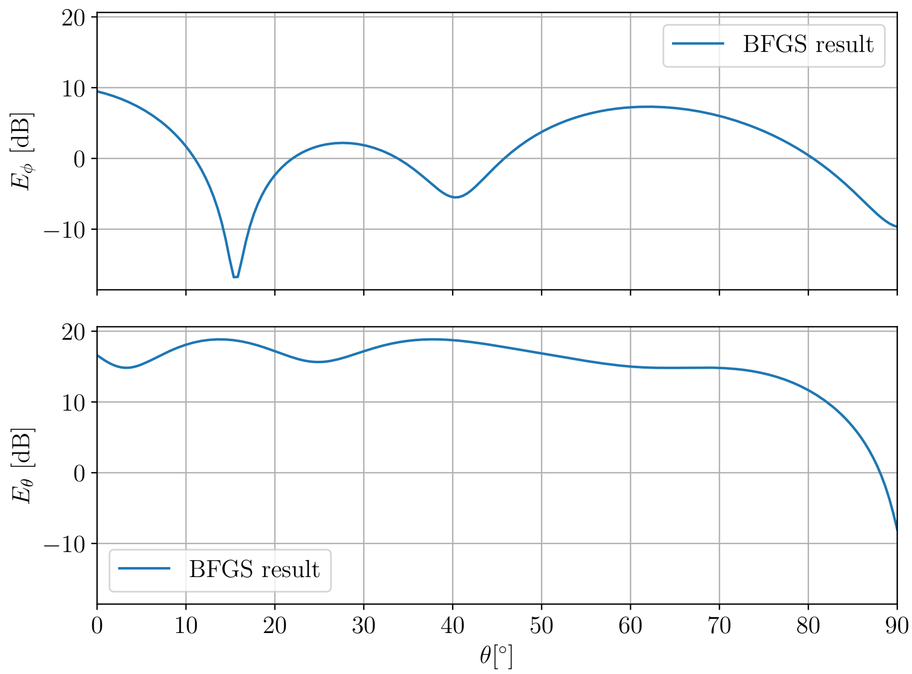

Figure 4.10: Fields obtained by the BFGS algorithm with 5 elements in array for the plateau pattern design in $|E_\theta|$ with $\theta_{\text{max}} = 70°$. Compared with the solution array using 3 elements, shown in Figure 4.9, the result is more oscillatory, but with higher plateau values as there are more elements.

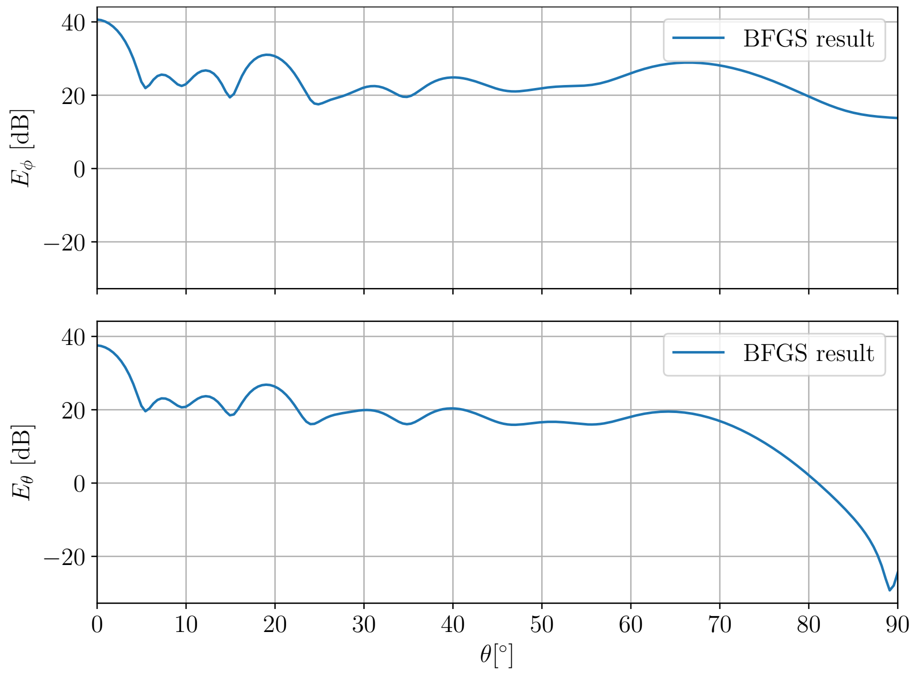

Figure 4.11: Fields obtained by the BFGS algorithm with 21 elements in array for the plateau pattern design in $|E_\theta|$ with $\theta_{\max} = 70°$. This solution is even more oscillatory than the one with 5 elements, shown in Figure 4.10, and has a higher plateau value.

4.4 Filter Pattern Design for Satellite Passes

In this section is presented an array design considering given satellite passes.

For this section, the considered array element is the one presented in Figure 4.12, which is designed with AFTK considering the maximum directive antenna with $l_{\max} = 5$.

The considered satellite for this scenario is VCUB1, NORAD 56215, passing over the centroid point of Brazil.

To achieve the goal of projecting fixed-position arrays capable of tracking a specific satellite, the passes are categorized, as illustrated in Figure 4.13. The time interval considered is 60 days. For the algorithm validation, the descending east passes are considered.

In this scenario, it is possible to model the optimization problem as:

$$\begin{aligned} \min \sum_{\boldsymbol{x}} |f_d(\boldsymbol{x}) - E_{tot}(\boldsymbol{x})|&, \text{subject to} \\ \frac{\min E_{tot}}{\max E_{tot}} - \epsilon_E >= 0&, \text{and} \\ d_{\min} - \epsilon_d >= 0&, \end{aligned} \tag{4.7}$$

where f_d is the desired pattern, which in this case is a constant vector with all elements equal one. E_{tot} is the normalized field absolute value, $\sqrt{|E_\theta|^2 + |E_\phi|^2}$. $d_{\min}$ is the minimum acceptable distance between two elements in multiple of λ. At last, ϵ_E is the goal ratio between the minimum and maximum E_{tot} inside the region of interest and ϵ_d is the minimum acceptable distance between two array elements. Ideally, $\epsilon_E = 1$ and ϵ_d are big enough to prevent a

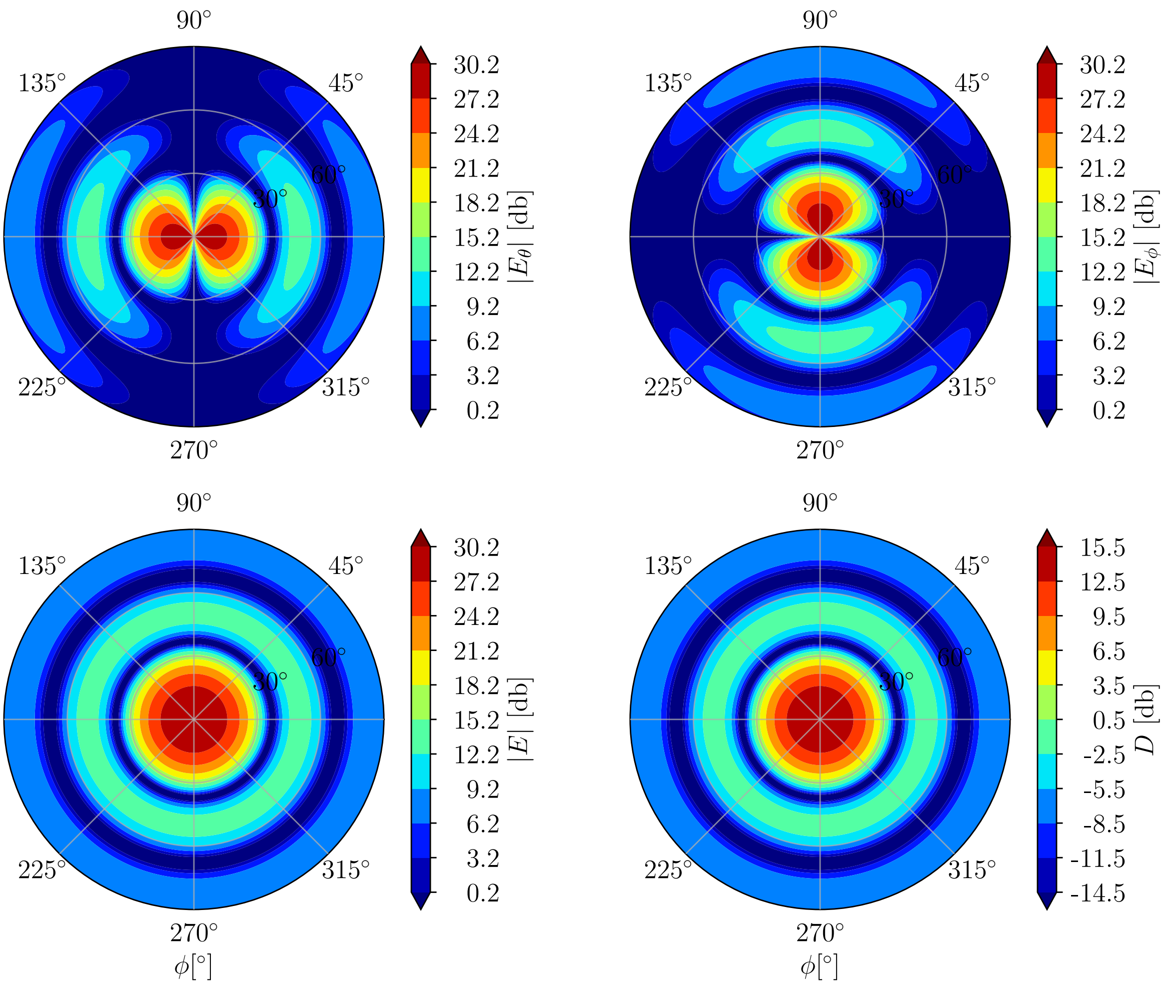

Figure 4.12: Electric fields and directivity for one element used in the design of the array that points towards the satellite path. It is designed with AFTK, considering the maximum directive antenna with $l_{\max} = 5$.

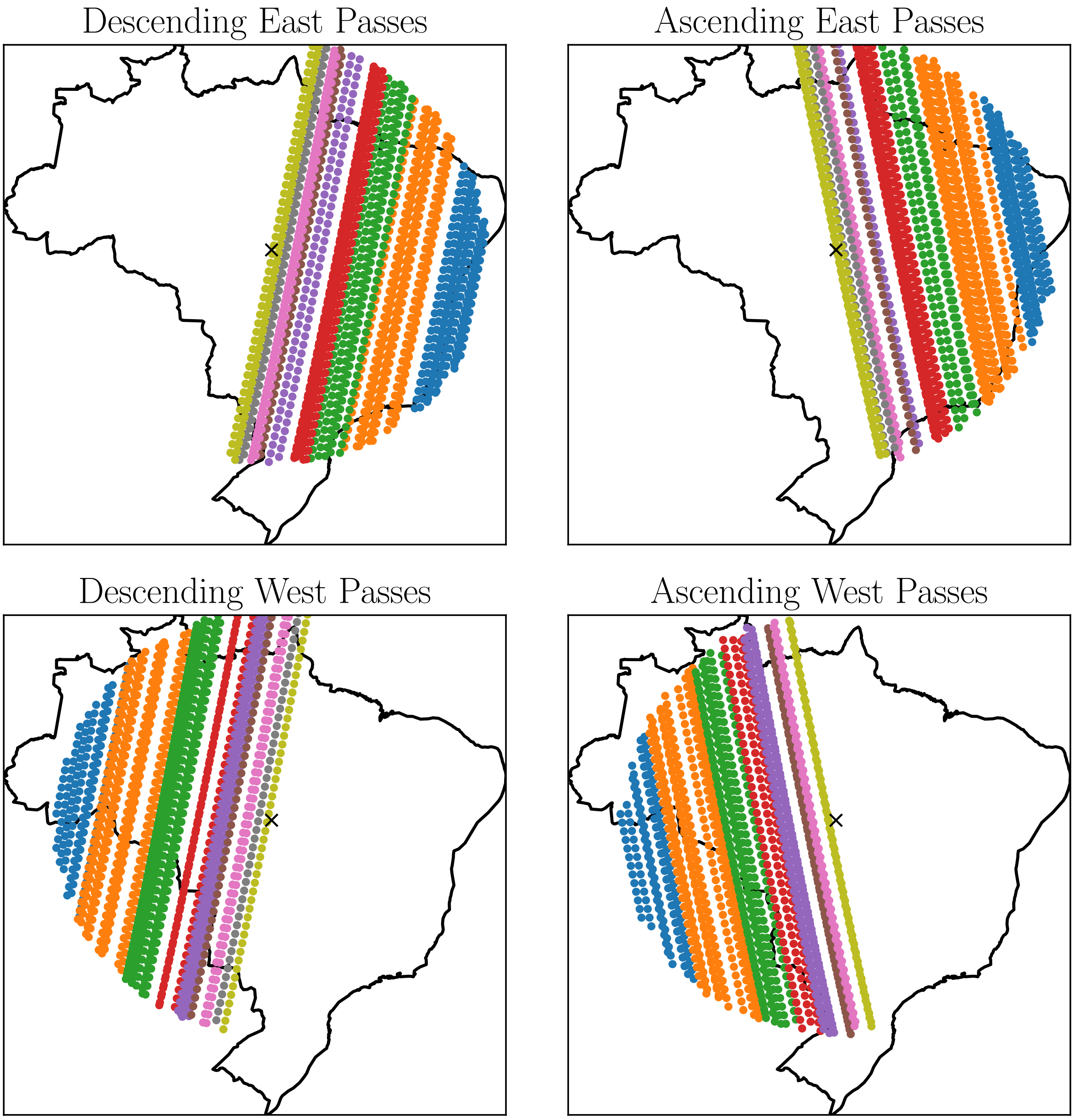

Figure 4.13: This figure shows how the passes are categorized before the optimization runs. First, they are divided into descending and ascending passes. Then, into east or west of the considered ground station. Finally, each group is divided considering the maximum elevation of the passes into nine categories. Each color represents passes in the same maximum elevation group.

physical intersection between two elements. Two possibilities are considered for $\boldsymbol{x}$:

$$\boldsymbol{x} = \begin{bmatrix} | & | & | & | & | \\ \alpha_i & \beta_i & \gamma_i & |a|_i & \angle a_i \\ | & | & | & | & | \end{bmatrix}_{N\times 5}, \text{or} \tag{4.8}$$

$$\boldsymbol{x} = \begin{bmatrix} | & | & | & | & | & | \\ x_i & y_i & z_i & \alpha_i & \beta_i & \gamma_i \\ | & | & | & | & | & | \end{bmatrix}_{N\times 6}, \tag{4.9}$$

where N is the number of elements composing the array, x_i, y_i, z_i are the Cartesian position for the i-th antenna and $\alpha_i, \beta_i, \gamma_i$ are the rotation around x-axis, y-axis and z-axis, respectively, for the i-th antenna. Finally, $|a_i|$ and $\angle a_i$ are the input magnitude and phase for the i-th antenna.

This problem is highly nonlinear, and there are multiple possible local solutions. To prevent the algorithm from converge to one of these local solutions, two runs are executed.

The first run considers the $\boldsymbol{x}$ in (4.8), which is a more conventional approach for designing arrays and provides a good start point for the next run.

The second run considers the $\boldsymbol{x}$ in (4.9), which provides the final solution for the problem.

For the proposed solutions, the algorithm runs for arrays with a minimum of $N = 3$ and a maximum of $N = 9$ elements, with $\epsilon_E = 1$ and $\epsilon_d = 0.3$. The most directive arrays are presented in Figure 4.14, with the details about the position in Table 4.3. This algorithm is implemented by the script in Appendix B.15.

The arrays found are not able to establish a link with a satellite like VCUB1, because they are not very directive and the output power of the satellite is too low.

These total cost of these arrays is around $1 million. Which is 25% of the cost of a state-of-the-art parabolic reflector.

Table 4.3: Final array configurations for the satellite descending east pass optimization. The values of x, y and z are in multiple of λ.

$x[\lambda]$	$y[\lambda]$	$z[\lambda]$	$\alpha[°]$	$\beta[°]$	$\gamma[°]$
0° to 10° - 5 elements					
0.26	-1.34	2.23	-66.15	-73.68	240.61
0.75	0.39	2.53	-61.47	-74.17	208.79
0.88	0.92	2.40	10.26	82.23	81.88
0.51	-0.85	1.80	65.53	73.45	62.94
0.57	1.78	2.70	47.77	81.02	20.57
10° to 20° - 8 elements					
0.69	-0.92	4.38	69.78	-48.25	3.45
1.68	-0.81	4.28	74.85	-50.55	316.94
1.58	-0.22	3.87	-74.54	47.52	140.83
0.87	-0.08	5.04	62.75	-71.99	31.88
1.49	0.92	4.58	-5.07	32.02	142.86

Continued on next page

Table 4.3 – continued from previous page

1.69	0.49	5.07	10.01	-79.18	330.37
1.21	1.43	4.85	-67.96	49.46	167.73
0.73	2.32	2.36	-73.23	-70.09	318.36
20° to 30° - 4 elements					
-1.75	0.89	1.71	-9.43	78.94	54.33
-1.12	1.14	1.88	16.90	65.22	65.64
-1.32	1.46	2.25	79.69	-33.94	66.78
-1.78	2.16	2.36	66.86	4.69	39.15
30° to 40° - 4 elements					
1.25	-0.04	1.94	-39.28	76.83	80.71
0.68	-0.23	1.69	12.21	59.19	63.40
0.98	0.11	1.49	76.32	35.22	82.85
0.38	-0.07	3.11	56.98	12.33	45.24
40° to 50° - 4 elements					
1.24	-0.79	1.69	-29.44	71.19	59.90
0.96	-0.68	1.21	21.14	44.13	44.90
0.57	-0.37	1.72	73.30	35.44	93.55
1.69	-0.34	2.02	29.11	41.09	95.43
50° to 60° - 5 elements					
2.22	-0.91	3.75	-71.95	77.71	91.51
2.92	0.73	3.43	-45.88	22.08	109.22
2.35	0.96	3.03	-14.01	26.72	131.14
1.93	2.11	3.29	41.74	-27.47	41.07
1.92	2.95	3.20	58.24	-77.14	66.11
60° to 70° - 7 elements					
1.99	-1.40	3.61	-18.26	67.65	53.95
2.13	-1.38	3.25	-21.61	76.21	49.59
2.15	-0.60	3.63	-35.75	-11.28	252.61
2.33	0.39	3.66	-11.03	32.74	108.05
1.60	-1.32	4.45	-6.26	39.92	60.59
2.40	1.21	3.63	59.60	-3.88	67.94
2.60	3.63	2.80	77.60	-48.43	83.96
70° to 80° - 9 elements					
-1.98	-3.75	5.68	-8.41	52.94	34.38
-2.30	-4.15	5.55	-44.84	85.19	58.53
-2.30	-3.62	5.54	-78.52	46.86	101.57
-2.04	-4.03	5.49	6.35	13.70	45.82
-2.23	-3.36	5.13	-2.89	30.86	29.83
-1.76	-4.71	5.62	24.54	0.07	66.96
-1.79	-4.16	5.58	54.22	1.80	85.09
-1.89	-2.14	5.82	74.28	80.31	107.56
-2.15	-3.85	6.07	72.82	19.32	103.72
80° to 90° - 8 elements					
0.99	0.98	5.42	-71.28	37.90	94.53
1.17	1.31	5.23	-53.91	41.46	68.79
0.50	1.26	4.69	-27.00	28.15	65.71

Continued on next page

Table 4.3 – continued from previous page

0.23	1.09	4.62	4.96	14.29	31.33
0.93	1.00	4.61	6.03	-9.40	34.32
0.64	0.60	4.76	29.80	-23.79	60.78
0.12	1.27	5.32	65.96	-14.30	92.55
0.31	1.46	5.36	72.22	-33.85	82.64

As a measure to increase the directivity in the desired region, the arrays found for each group were considered as a substation in a new uniform linear array separated by λ that is in the direction perpendicular to the satellite trajectory. The resulting array is also electronically steered towards θ_{mean} of each group of passes. The resulting directivity if shown in Figure 4.15, from where it is possible to see that this approach worked well for the satellite groups with maximum elevation lower than 20° and maximum elevation higher than 50°, in which the directivity is greater than the original array found by the optimization. Unexpectedly, it does not work well for the passes with a maximum elevation between 20° and 50°.

Finally, this simulation is repeated considering that the array is composed of an element that is equivalent to a $1m$ dish parabola. The results are shown in Figure 4.16.

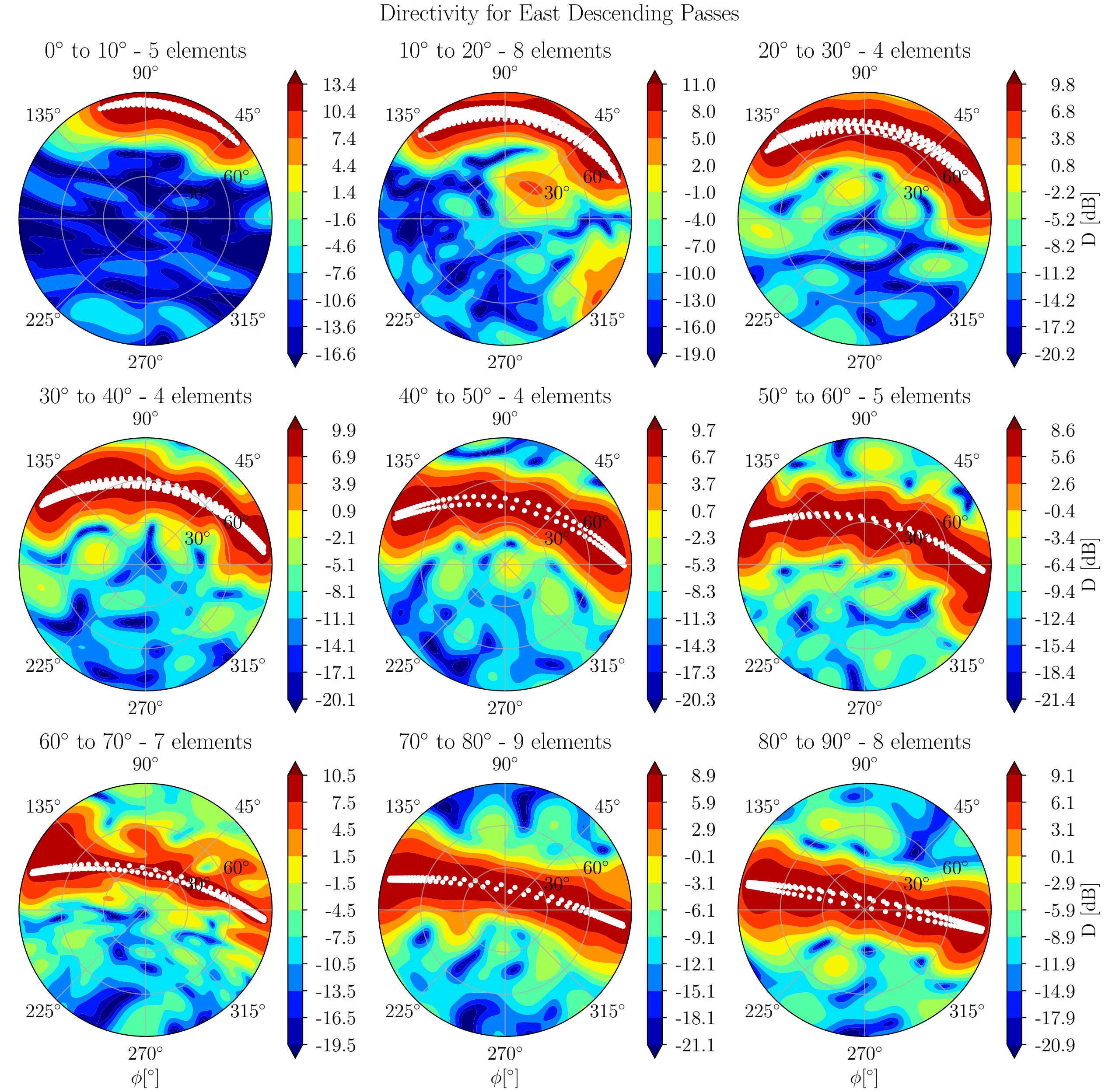

Figure 4.14: This figure shows the directivity for the arrays of all groups of descending passes that are located east from the ground station. The objective is to maintain an approximately constant power level for each satellite pass, which is represented by the white dots. The element composing the arrays in this scenario is equivalent to a 30 cm dish parabola.

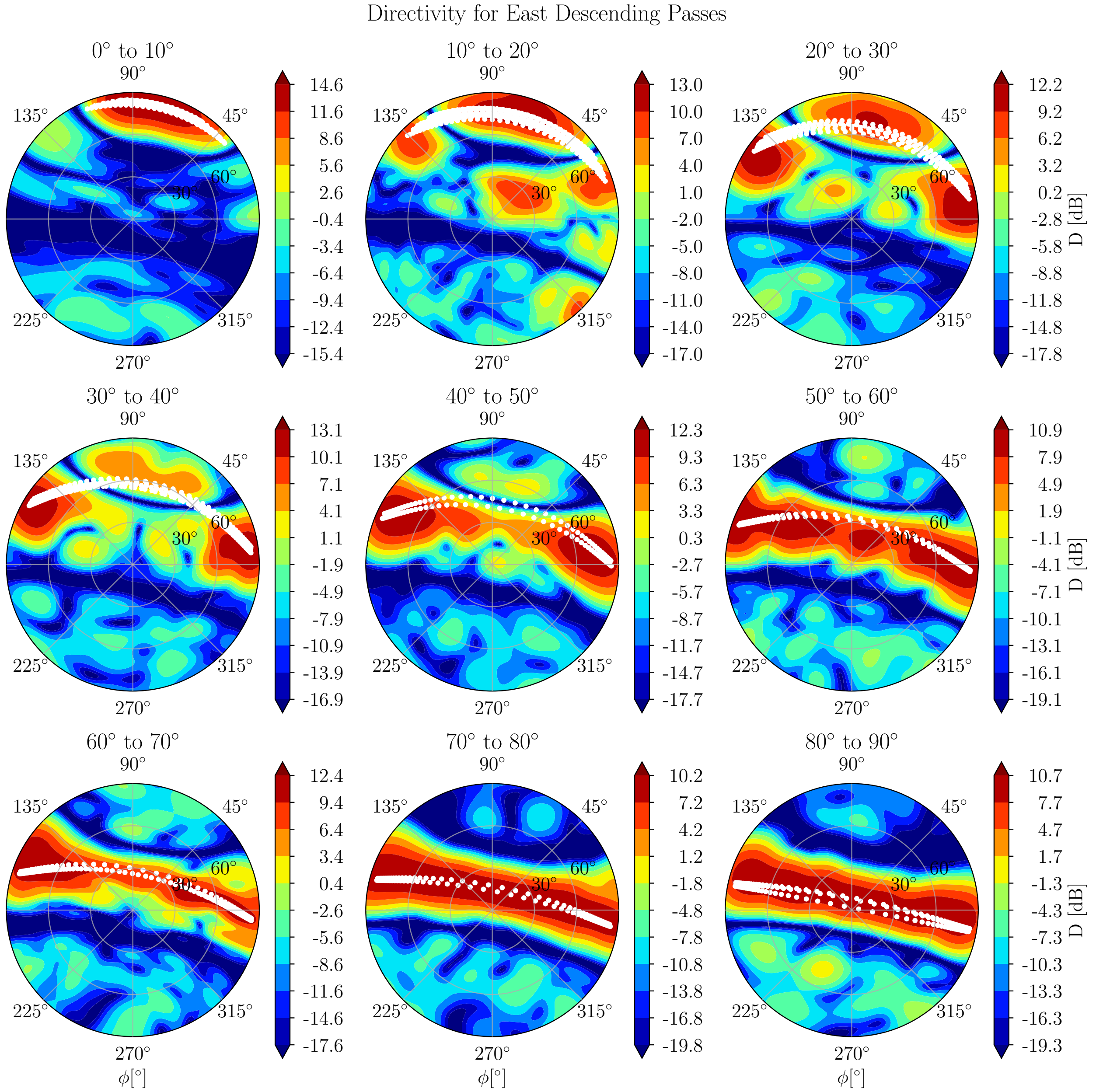

Figure 4.15: This figure shows the arrays with 2 substations for the descending satellite passes that are located east from the ground station. It is possible to see that this approach worked well for the satellite groups with maximum elevation lower than 20° and with maximum elevations higher than 50°, in which the directivity is greater than the original array found by the optimization. The satellite trajectory is depicted by the white dots on each graph.

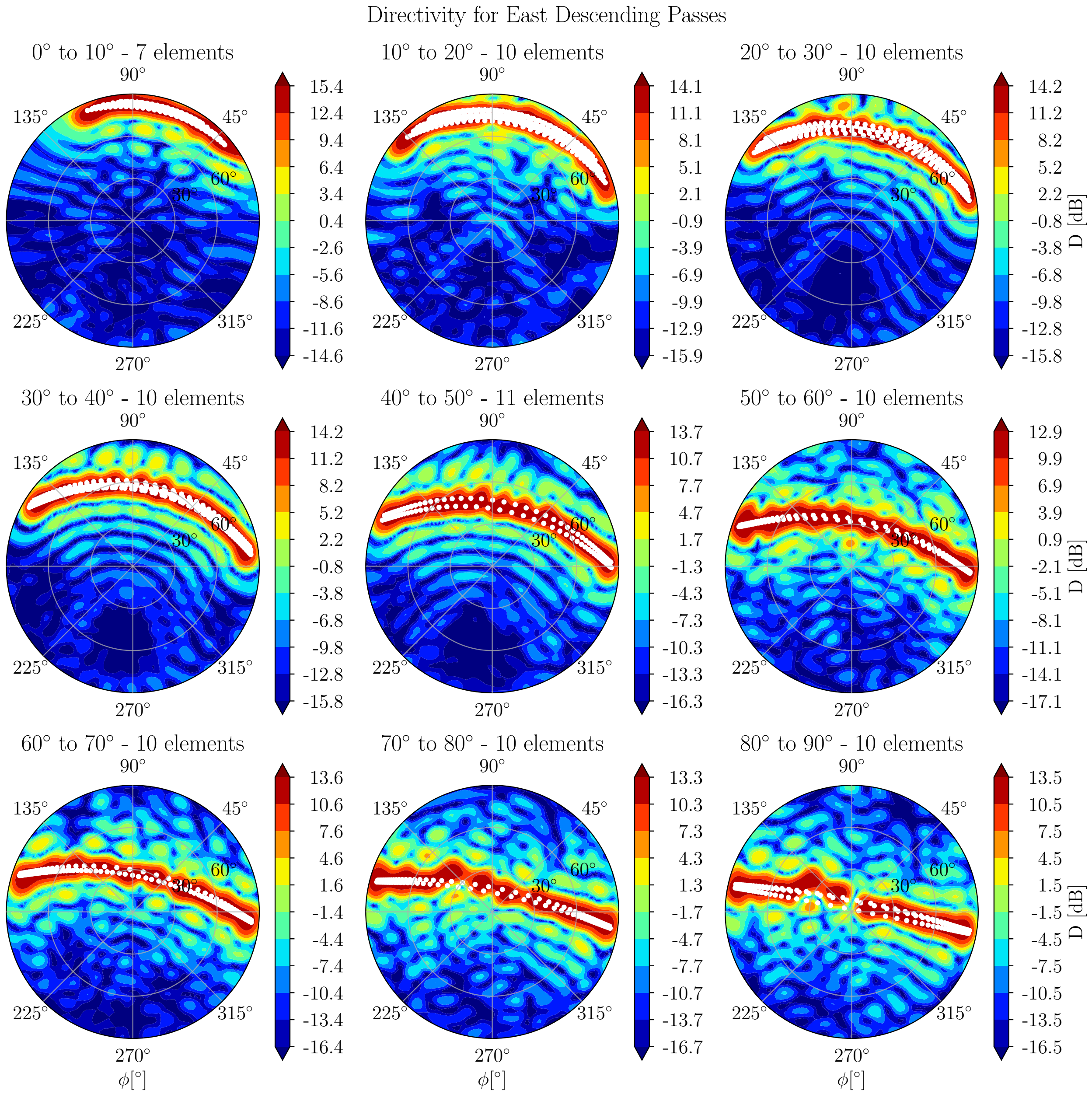

Figure 4.16: This figure shows the directivity for the arrays of all groups of descending passes that are located east from the ground station. The objective is to maintain an approximately constant power level for each satellite pass, which is represented by the white dots. The element composing the arrays in this scenario is equivalent to a 1 m dish parabola.

Part V

Results and Conclusion

Chapter 5

Results and Conclusion

This chapter reviews and summarizes the obtained results of this work and suggests future works to continue the research.

This work presented many optimization techniques involving satellite communication, like proposing a ground station distribution to maximize a territory coverage, also proposing a distribution inside the Brazilian territory that has virtually the same downlink capability of a station placed near the South Pole and, finally, proposing an static array that can maintain an approximately constant power level during a satellite pass.

5.1 Ground Station Distribution Maximizing Coverage

The proposed technique to solve the problem of optimal placement of ground stations over a region to establish communication links with specific satellites involved a multi-step process utilizing three optimization techniques.

First, the problem was linearly relaxed so it could be convexly modeled and solved by Convex Optimization. The limitation of this model is that the antenna positions were restrained to a coarse grid.

This solution was then fed to a Sequential Least Squares model, where the positions are no longer restrained. In this scenario, antennas can even be placed outside the region of interest, which is a drawback of this model. The antenna coverage pattern depends on the geodetic coordinates of the ground stations. However, it was considered fixed in this model.

The result from SQLQ was then the input for a Differential Evolution algorithm. This one, at last, made a fine adjustment in the previous solution while accommodating all problem constraints. This combination of techniques was able to successfully place ground stations inside Brazilian territory considering communication links with SAC-C and VCUB1 satellites with acceptable percentages of coverage.

Furthermore, it was observed that despite the problem relaxation, the number of antennas obtained by the Convex Optimization model was an optimal trade-off between the number of antennas and the covered area. That is, adding one more antenna to this solution before feeding it to the other algorithms did not lead to a significant increase in the coverage area while increasing the intersection among the antennas.

5.2 Ground Station Parameters Optimization

Adapting the used Sequential Least Squares model, it was analyzed a different scenario. In this case, the ground station must be placed in specific locations, due to some legacy infrastructure, for instance. Given these possible positions, the proposed algorithm finds which are the best antenna to employ in order to maximize the link coverage. In the specific solved case, were considered parabolic antennas with variable diameters and the algorithm found where to put the stations and which parabolic diameter to use.

In this new scenario, it was proposed a two-step process. The first step is a simplified version of the problem, which considers that each antenna coverage is circular. The solution of this step is then fed into the same optimization, with one difference: considering the real coverage instead of circular approximations. The second algorithm was not able to find a better solution as it could not find any direction where the gradient was negative.

The proposed solution was compared with antennas near the poles, in Comandante Ferraz Antarctic Station, that is the Brazilian South Pole Station, and in Svalbard, Norway. The proposed solution has virtually the same downlink capability as the antenna placed in Comandante Ferraz and approximately 65% of the capability of Svalbard.

5.3 Antenna Array Design

There are methods of designing arrays considering the electronic steer of the elements. That is, it varies only the feed input magnitude and phase to achieve some desired pattern. It is proposed a method to vary the positions and physical rotation of the array elements, increasing the degrees of freedom of the problem.

The main objective of this section is to propose an static array that can maintain an approximately constant level of power during a satellite pass. To achieve this, first the possible passes were categorized, to reduce the area in which is necessary to irradiate power. It were used two elements for the design, one equivalent to a 30 cm dish parabola and other equivalent to a 1 m dish parabola. In both cases, the maximum directivity of the obtained arrays were not greater than 15 dB, which is not enough to establish a link with the considered satellite.

A more directive element could be used. However, it would require a huge computational capability, as the number of elements necessary in each array would increase considerably. Also, as the software AFTK was used for modeling the element arrays, for more directive elements it requires more spherical modes to be able to estimate the antenna, which also increases the computational cost of the problem.

For these reasons, the conclusion was that this approach to the problem is not the best. It would be best to use a design method considering Space-Fed Lens, which is composed of a feed array and a radiating array with each corresponding element pair interconnected by transmission lines of different lengths to radiate a plane wave in the forward direction, as described in [10] and [12].

5.4 Other applications

The developed algorithms can also be used to solve problems as:

- How to distribute air defense radars to fully cover a given territory;
- How to distribute surveillance telescopes to get every space object that passes over a given territory; and
- Integration with ANSYS STK through easy tweaks if necessary.

5.5 Future Works

This section presents some directions for future research, showing identified gaps observed during the development of this work.

First, the inclusion of the Doppler effect in satellite link calculations. The Doppler effect must be compensated in order to establish a link connection with a satellite, so this must be integrated into the software that is being developed in the context of the project.

Another possible work is to consider dynamic elements to compose the array instead of static ones. This would compensate the problems encountered with the low directivity.

It would also be relevant a research regarding the impacts of ionospheric scintillation in satellite communication, as the Brazilian territory is affected by the South Atlantic Anomaly.

Finally, the electronic steering of the found arrays in Chapter 4 needs to be investigated in more detail, finding a way to make the arrays more directive and, therefore, being able to establish a communication link with more satellites.

Part VI

Appendices

Appendix A

Antenna Array Electrical Design

A.1 Translational Phase Shift

The fundamental characteristic of an array is that the spatial displacements between its antenna elements result in relative phase shifts within the radiation vectors. These shifts can either combine constructively in certain directions or cancel each other out in others. This phenomenon directly derives from the translational phase-shift property inherent in Fourier transforms, where a spatial or temporal translation corresponds to a phase shift in the Fourier domain.

The current density of the translated antenna is $\mathbf{J_d}(\mathbf{r}) = \mathbf{J}(\mathbf{r} - \mathbf{d})$. By definition, the radiation vector is the three-dimensional Fourier transform of the current density. Thus, the radiation vector of the translated current is:

$$\begin{aligned} \mathbf{F_d} &= \int e^{j\mathbf{k}\cdot\mathbf{r}} \mathbf{J_d}(\mathbf{r}) d^3\mathbf{r} = \int e^{j\mathbf{k}\cdot\mathbf{r}} \mathbf{J}(\mathbf{r} - \boldsymbol{d}) d^3\mathbf{r} = \int e^{j\mathbf{k}\cdot(\mathbf{r}'+\boldsymbol{d})} \mathbf{J}(\mathbf{r}') d^3\mathbf{r}' \\ &= e^{j\mathbf{k}\cdot\boldsymbol{d}} \int e^{j\mathbf{k}\cdot\mathbf{r}'} \mathbf{J}(\mathbf{r}') d^3\mathbf{r}' = e^{j\mathbf{k}\cdot\boldsymbol{d}} \boldsymbol{F}, \end{aligned} \tag{A.1}$$

with $\boldsymbol{r}' = \boldsymbol{r} - \boldsymbol{d}$.

A.2 Array Pattern Multiplication

Consider N antennas in positions $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{N-1}$ with relative feed coefficients $a_0, a_1, \ldots, a_{N-1}$. The current density of the nth antenna will be $\mathbf{J_n}(\mathbf{r}) = a_n \mathbf{J}(\mathbf{r} - \mathbf{d})$ and the corresponding radiation vector:

$$\mathbf{F_n}(\mathbf{k}) = a_n e^{j\mathbf{k}\cdot\boldsymbol{d}_n} \boldsymbol{F}(\boldsymbol{k}). \tag{A.2}$$

The total current density of the array is:

$$\boldsymbol{J}_{tot}(\boldsymbol{r}) = a_0 \boldsymbol{J}(\boldsymbol{r} - \boldsymbol{d}_0) + a_1 \boldsymbol{J}(\boldsymbol{r} - \boldsymbol{d}_1) + \cdots + a_{N-1} \boldsymbol{J}(\boldsymbol{r} - \boldsymbol{d}_{N-1}). \tag{A.3}$$

And the total radiation vector is:

$$\boldsymbol{F}_{tot}(\boldsymbol{k}) = \sum_n \boldsymbol{F}_n = \sum_n a_n e^{j\mathbf{k}\cdot\boldsymbol{d}_n} \boldsymbol{F}(\boldsymbol{k}) = A(\boldsymbol{k})\boldsymbol{F}(\boldsymbol{k}), \tag{A.4}$$

where $A(\boldsymbol{k})$ is the array factor:

$$A(\boldsymbol{k}) = \sum_n a_n e^{j\mathbf{k}\cdot\boldsymbol{d}_n}. \tag{A.5}$$

Since $\boldsymbol{k} = k\hat{\mathbf{r}}$, it is also possible to denote the array factor as $A(\hat{\mathbf{r}})$ or $A(\theta, \phi)$.

A.3 One-Dimensional Arrays

Consider a uniformly-spaced one-dimensional array. An array along the x-axis with elements positioned at locations x_n, $n = 0, 1, 2 \ldots$, will have displacement vectors $\vec{d_n} = x_n\hat{\mathbf{x}}$ and array factor:

$$A(\theta, \phi) = \sum_n a_n e^{j\vec{k}\cdot\vec{d}_n} = \sum_n a_n e^{jk_n x_n} = \sum_n a_n e^{jkx_n \sin\theta\cos\phi}, \tag{A.6}$$

where $k_x = k\sin\theta\cos\phi$. In this case, the array factor is:

$$A(\theta, \phi) = \sum_n a_n e^{jnkd\sin\theta\cos\phi} = \sum_n a_n e^{j\Psi n}, \tag{A.7}$$

where $\Psi = k_x d = kd\sin\theta\cos\phi$ is the digital wavenumber, which is a normalized version of the wavenumber k_x and is measured in units of radians per (space) sample.

The wavenumber Ψ is defined similarly for arrays along the y- or z- directions:

$$\begin{aligned} \Psi &= k_x d = kd\sin\theta\cos\phi && \text{for an array along } x\text{-axis,} \\ \Psi &= k_y d = kd\sin\theta\sin\phi && \text{for an array along } y\text{-axis, and} \\ \Psi &= k_z d = kd\cos\theta && \text{for an array along } z\text{-axis.} \end{aligned} \tag{A.8}$$

A.3.1 Analogy with Time-Domain Digital Signal Processing (DSP)

The array factor $A(\Psi)$ is the wavenumber version of the frequency response of a digital filter defined by:

$$A(\omega) = \sum_n a_n e^{-j\omega n}. \tag{A.9}$$

The distinction in the exponent sign between (A.9) and (A.7) is rooted in the contrast between defining time-domain and space-domain Fourier transforms. This discrepancy can also be attributed to the variation in the sign for a plane wave, specifically expressed as $e^{j\omega t - j\boldsymbol{k}\cdot\boldsymbol{r}}$.

It is also possible to define the spatial analog of the z-plane by defining the variable $z = e^{j\Psi}$ and the corresponding z-transform:

$$A(z) = \sum_n a_n z^n. \tag{A.10}$$

The difference between the space-domain and time-domain definitions is also evident in this equation, where the expansion is in powers of z^n instead of z^{-n}.

The array factor $A(\Psi)$ can be referred to as the discrete-space Fourier transform (DSFT) of the array weighting sequence a_n, like the discrete-time Fourier transform (DTFT) in the time-domain scenario. The corresponding inverse DSFT is obtained by:

$$a_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} A(\Psi) e^{-j\Psi n} d\Psi. \tag{A.11}$$

The inverse transform forms the basis of most design methods for the array coefficients. These methods are identical to the methods of designing finite impulse response filters in DSP.

A.4 Visible Region

The array factor $A(\Psi)$ is periodic in Ψ with period 2π, which means it is enough to know it within one Nyquist interval, that is, $-\pi \leq \Psi \leq \pi$.

However, the actual range of variation of Ψ depends on $kd = 2\pi d/\lambda$. The overall range of variation of Ψ is called the visible region is defined as:

$$-kd \leq \Psi \leq kd. \tag{A.12}$$

The visible region can also be viewed as that part of the unit circle covered by the angle range.

Depending on the value of kd, the visible region can be less, equal or more than one Nyquist interval:

$$\begin{aligned} d < \lambda/2 \Rightarrow kd < \pi \Rightarrow \Psi_{\text{vis}} < 2\pi \quad & \text{less than Nyquist,} \\ d = \lambda/2 \Rightarrow kd = \pi \Rightarrow \Psi_{\text{vis}} = 2\pi \quad & \text{full Nyquist, or} \\ d > \lambda/2 \Rightarrow kd > \pi \Rightarrow \Psi_{\text{vis}} > 2\pi \quad & \text{more than Nyquist.} \end{aligned} \tag{A.13}$$

A.5 Grating Lobes

When $kd > \pi$, the values of $A(\Psi)$ become redundant and cyclically repeat across the visible region. This redundancy can lead to the emergence of grating lobes or fringes, representing mainbeam lobes in directions other than the intended one.

The quantity of grating lobes within an array pattern corresponds to the number of complete Nyquist intervals that fit within the width of the visible region, expressed as:

$$m = \Psi_{\text{vis}}/2\pi = kd/\pi = 2d/\lambda. \tag{A.14}$$

A.6 Electronically Steered Array

An array is typically designed to achieve maximum directive gain at broadside, specifically at $\phi = \frac{\pi}{2}$ (assuming an array along the x-axis). The objective is to electronically steer the array pattern towards a different direction, denoted as ϕ_0, without the need for physical rotation.

This steering process can be accomplished through wavenumber translation in Ψ-space, where the broadside pattern $A(\Psi)$ is replaced by the translated pattern $A(\Psi - \Psi_0)$. Thus, is defined the steered array factor:

$$A'(\Psi) = A(\Psi - \Psi_0), \tag{A.15}$$

and the translated wavenumber variable:

$$\Psi' = \Psi - \Psi_0. \tag{A.16}$$

The concept of visible region translates with minor modifications to the case of a steered array:

$$-kd(1 + \cos\psi_0) \leq \Psi' \leq kd(1 + \cos\psi_0). \tag{A.17}$$

A.7 Array Design Methods

The array design problem is essentially equivalent to designing finite impulse response (FIR) digital filters in DSP.

A.7.1 Notation

One-dimensional equally-spaced arrays are commonly analyzed with symmetry concerning the origin of the array axis. However, when dealing with an even number of array elements, a slight adjustment to the definition of the array factor is necessary.

Consider an array of N elements at locations $\{x_m\}$ along the x-axis with element spacing d. The array factor is:

$$A(\theta, \phi) = \sum_m a_m e^{jk_x x_m} = \sum_m a_m e^{jkx_m \sin\theta\cos\phi}. \tag{A.18}$$

If $N = 2M + 1$ (odd), the element locations $\{x_m\}$ are:

$$x_m = md, \quad m = 0, \pm 1, \pm 2, \ldots, \pm M. \tag{A.19}$$

Writing the array factor as a discrete-space Fourier transform and as a spatial z-transform:

$$\begin{aligned} A(\Psi) &= \sum_{m=-M}^{M} a_m e^{jm\Psi} = a_0 + \sum_{m=1}^{M} \left[a_m e^{jm\Psi} + a_{-m} e^{-jm\Psi}\right], \text{ and} \\ A(z) &= \sum_{m=-M}^{M} a_m z^m = a_0 + \sum_{m=1}^{M} \left[a_m z^m + a_{-m} z^{-m}\right]. \end{aligned} \tag{A.20}$$

On the other hand, if $N = 2M$ (even), in order to have symmetry with respect to the origin, the elements x_m must be placed in half-integer locations:

$$x_{\pm m} = \pm\left(md - \frac{d}{2}\right) = \pm\left(m - \frac{1}{2}\right) d, \quad m = 1, 2, \ldots, M. \tag{A.21}$$

Writing the array factor as a discrete-space Fourier transform and as a spatial z-transform:

$$\begin{aligned} A(\Psi) &= \sum_{m=1}^{M} \left[a_m e^{j(m-1/2)\Psi} + a_{-m} e^{-j(m-1/2)\Psi}\right], \text{ and} \\ A(z) &= \sum_{m=1}^{M} \left[a_m z^{m-1/2} + a_{-m} z^{-(m-1/2)}\right]. \end{aligned} \tag{A.22}$$

In most design methods, the weights array a_m is symmetric with respect to the origin, that is, $a_m = a_{-m}$. In this case, the array factor can be simplified:

$$\begin{aligned} A(\Psi) &= a_0 + 2\sum_{m=1}^{M} a_m \cos(m\psi), \text{ for} N = 2M+1, \text{ and} \\ A(\Psi) &= 2\sum_{m=1}^{M} a_m \cos\left[(m-1/2)\psi\right], \text{ for} N = 2M. \end{aligned} \tag{A.23}$$

In both even and odd cases, the spatial z-transform can be expressed as the left-shifted version of a right-sided z-transform:

$$A(z) = z^{-(N-1)/2} \tilde{A}(z) = z^{-(N-1)/2} \sum_{n=0}^{N-1} \tilde{a}_n z^n, \tag{A.24}$$

where $\boldsymbol{a} = [\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_{N-1}]$ is the vector of array weights reindexed to be right-sided. In terms of the original symmetric:

$$\begin{aligned} \boldsymbol{a} &= [\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_{N-1}] = [a_{-M}, \ldots, a_{-1}, a_0, a_1, \ldots, a_M], \quad \text{for } N = 2M+1, \text{ and} \\ \boldsymbol{a} &= [\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_{N-1}] = [a_{-M}, \ldots, a_{-1}, a_1, \ldots, a_M], \quad \text{for } N = 2M. \end{aligned} \tag{A.25}$$

The corresponding array factors in Ψ-space are, setting $z = e^{j\Psi}$:

$$A(\Psi) = e^{-j\Psi(N-1)/2} \tilde{A}(\Psi) = e^{-j\Psi(N-1)/2} \sum_{n=0}^{N-1} \tilde{a}_n e^{jn\Psi}. \tag{A.26}$$

The steered version of $\tilde{A}(\Psi)$ is:

$$\tilde{A}'(\Psi) = e^{j\Psi_0(N-1)/2}\tilde{A}(\Psi - \Psi_0), \tag{A.27}$$

which implies for the weights:

$$\tilde{a_n}' = \tilde{a}_n e^{-j\Psi_0[n-(N-1)/2]}, \quad n = 0, 1, \ldots, N-1. \tag{A.28}$$

A.8 Woodward-Lawson Frequency-Sampling Design

Equations (A.20) and (A.22) represents truncated versions of the corresponding infinite Fourier series. Considering the case where the inverse transform integrals cannot be done exactly, the frequency-sampling design method of DSP is used [3] [5]. Assuming an infinite and convergent series, it is possible to write, for the odd case:

$$A(\psi) = a_0 + \sum_{m=1}^{\infty}\left[a_m e^{jm\Psi} + a_{-m}e^{-jm\Psi}\right] \overset{FT}{\rightleftharpoons} a_m = \frac{1}{2\pi}\int_{-\pi}^{\pi} A(\psi)e^{-jm\psi}d\psi. \tag{A.29}$$

Similarly, for the even case:

$$A(\psi) = \sum_{m=1}^{\infty}\left[a_m e^{j(m-1/2)\Psi} + a_{-m}e^{-j(m-1/2)\Psi}\right] \overset{FT}{\rightleftharpoons} a_{\pm m} = \frac{1}{2\pi}\int_{-\pi}^{\pi} A(\psi)e^{\mp j(m-1/2)\psi}d\psi. \tag{A.30}$$

Therefore, given a desired response, $A_d(\Psi)$, it is possible to choose a window length, N, and calculate the N ideal weights $a_d(m)$ by evaluating the inverse integrals of (A.29) or (A.30). Then, the final weights are obtained by windowing with a length-N window $w(m)$:

$$a(m) = w(m)a_d(m). \tag{A.31}$$

This method is convenient when it is possible to evaluate the integrals analytically, when $A_d(\psi)$ has a simple shape, such as an ideal lowpass filter. For arbitrary shaped $A_d(\psi)$, the integrals must be approximated by an inverse DFT. Also, the method requires that $A_d(\psi)$ be specified over one complete Nyquist interval, $-\pi \leq \psi \leq \pi$.

When it's not possible to analytically evaluate $A(\psi)$, it is necessary to have the array factor sampled at N points, named DFT frequencies, $\psi_i, i = 0, 1, \cdots, N-1$:

$$\psi_i = \frac{2\pi i}{N}. \tag{A.32}$$

The frequency samples $A(\psi_i)$ are related to the array weights by the forward N-point DFT's obtained by:

$$\begin{aligned} A(\psi_i) &= a_0 + \sum_{m=1}^{M}\left[a_m e^{jm\Psi} + a_{-m}e^{-jm\Psi}\right], \qquad N = 2M+1, \text{or} \\ A(\psi_i) &= \sum_{m=1}^{M}\left[a_m e^{j(m-1/2)\Psi} + a_{-m}e^{-j(m-1/2)\Psi}\right], \qquad N = 2M. \end{aligned} \tag{A.33}$$

The corresponding inverse N-point DFT's are as follows:

$$\begin{aligned} a_m &= \frac{1}{N}\sum_{i=0}^{N-1} A(\psi_i)e^{-jm\psi_i}, \quad N = 2M+1, \quad M = 0, \pm 1, \pm 2, \ldots, \pm M, \text{or} \\ a_{\pm m} &= \frac{1}{N}\sum_{i=0}^{N-1} A(\psi_i)e^{\mp j(m-1)\psi_i}, \quad N = 2M, \quad M = 1, 2, \ldots, M. \end{aligned} \tag{A.34}$$

There is an alternative definition of the N DFT frequencies ψ_i that is usually preferred in array processing and maintains the presented equations for the inverse DFT's:

$$\psi_i = \frac{2\pi(i-K)}{N}, \qquad K = \frac{N-1}{2}. \tag{A.35}$$

As an example, consider the design of a sector beam with edges θ_1 and θ_2. Thus, the beam is centered at $\theta_c = \dfrac{\theta_1 + \theta_2}{2}$. As θ ranges over $[\theta_1, \theta_2]$, the wavenumber for an array along the z-axis, $\psi = kd\cos\theta$, ranges over $kd\cos\theta_2 \le \psi \le kd\cos\theta_1$. Assuming the alternative definition for ψ_i:

$$A(\psi_i) = \begin{cases} 1, \text{ if } kd\cos\theta_2 \le \dfrac{2\pi(i-K)}{N} \le kd\cos\theta_1, \text{or} \\ 0, \text{ otherwise.} \end{cases} \tag{A.36}$$

Substituting $kd = 2\pi d/\lambda$, the DFT index $i - K$ becomes:

$$j_1 \le i - K \le j_2, \qquad \text{with } j_1 = \frac{Nd}{\lambda}\cos\theta_2, \quad j_2 = \frac{Nd}{\lambda}\cos\theta_1. \tag{A.37}$$

These are the indices i that $A(\psi_i) = 1$.

The method works well for half-wavelength spacing $d = \lambda/2$, because all N DFT frequencies ψ_i fall within the visible region, which, in this scenario, aligns with the complete Nyquist interval, defined as $-\pi \le \psi \le \pi$. An example of design is shown in Figure A.1.

This method can be used to any desired $A(\psi)$. Consider one that has a secant-squared gain pattern, as discussed in Section 2.4.7.

Consider an array of N elements along the z-direction with half-wavelength spacing $\lambda/2$. In this scenario, $\psi = kd\cos\theta$ and the desired array factor, $g_d(\theta)$, is:

$$g_d(\theta) = |A(\psi)|^2 = \frac{K}{\cos^2\theta} \to |A(\psi)| = \frac{\sqrt{K}}{|\cos\theta|}. \tag{A.38}$$

As the secant pattern is only desired to a maximum angle θ_{max}, the normalized theoretical array factor is:

$$A_{\text{norm}}(\theta) = \begin{cases} \dfrac{\cos\theta_{\text{max}}}{\cos\theta}, & \text{if } 0 \le \theta \le \theta_{\text{max}}, \text{ or} \\ 1, & \text{if } \theta_{\text{max}} \le \theta \le \dfrac{\pi}{2}. \end{cases} \tag{A.39}$$

As θ varies over $[0, \theta_{\text{max}}]$, the wavenumber ψ ranges over $[\psi_{\text{max}}, kd] = [\psi_{\text{max}}, \pi]$.

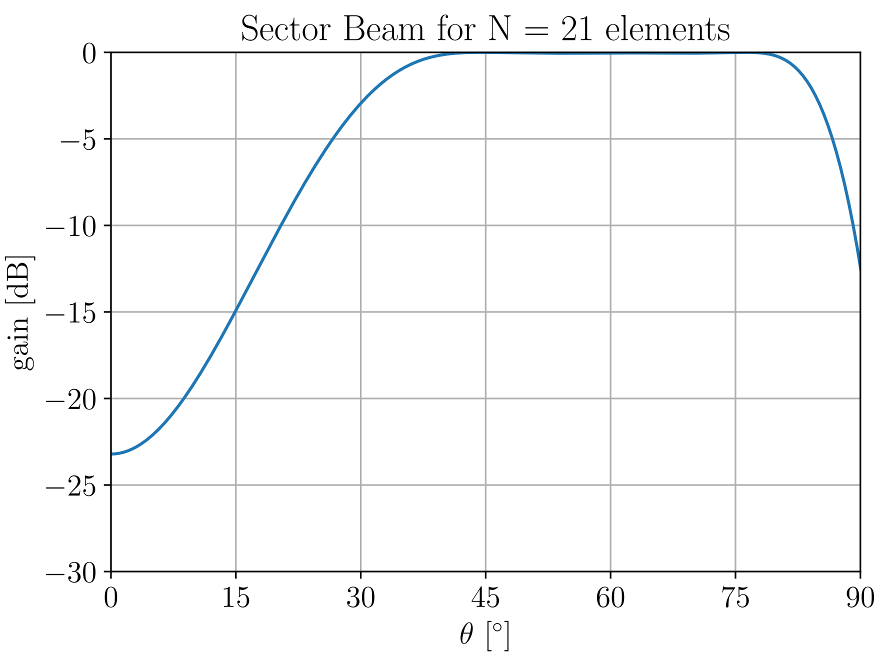

Figure A.1: Designed array factor of a sector beam with edges $\theta_1 = 20°$ and $\theta_2 = 90°$ for $N = 21$ elements spaced of $d = \lambda/2$ placed along the z-axis.

As $\cos\theta_{\mathrm{max}}/\cos\theta = \psi_{\mathrm{max}}/\psi$:

$$A_{\mathrm{norm}}(\psi) = \begin{cases} \dfrac{\psi_{\mathrm{max}}}{\psi}, & \text{if } \psi_{\mathrm{max}} \le \psi \le \pi, \text{ or} \\ 1, & \text{if } 0 \le \psi \le \psi_{\mathrm{max}}. \end{cases} \tag{A.40}$$

An example of this design is shown in Figure A.2.

An adaptation of this method is to include the gain pattern of one array element, $g_{\mathrm{elem}}(\theta)$. This means that the antenna array will not be considered as omnidirectional. In this case, the equivalent array factor (not normalized) is:

$$A(\theta) = \begin{cases} \dfrac{\cos\theta_{\mathrm{max}}}{g_{\mathrm{elem}}(\theta)\cos\theta}, & \text{if } 0 \le \theta \le \theta_{\mathrm{max}}, \text{ or} \\ \dfrac{1}{g_{\mathrm{elem}}(\psi)}, & \text{if } \theta_{\mathrm{max}} \le \theta \le \dfrac{\pi}{2}, \text{ and} \end{cases} \tag{A.41}$$

$$A(\psi) = \begin{cases} \dfrac{\psi_{\mathrm{max}}}{g_{\mathrm{elem}}(\theta)\psi}, & \text{if } \psi_{\mathrm{max}} \le \psi \le \pi, \text{ or} \\ \dfrac{1}{g_{\mathrm{elem}}(\theta)}, & \text{if } 0 \le \psi \le \psi_{\mathrm{max}}, \end{cases} \tag{A.42}$$

where $\psi = 2\pi d \cos\theta$.

An example of this design is shown in Figure A.3.

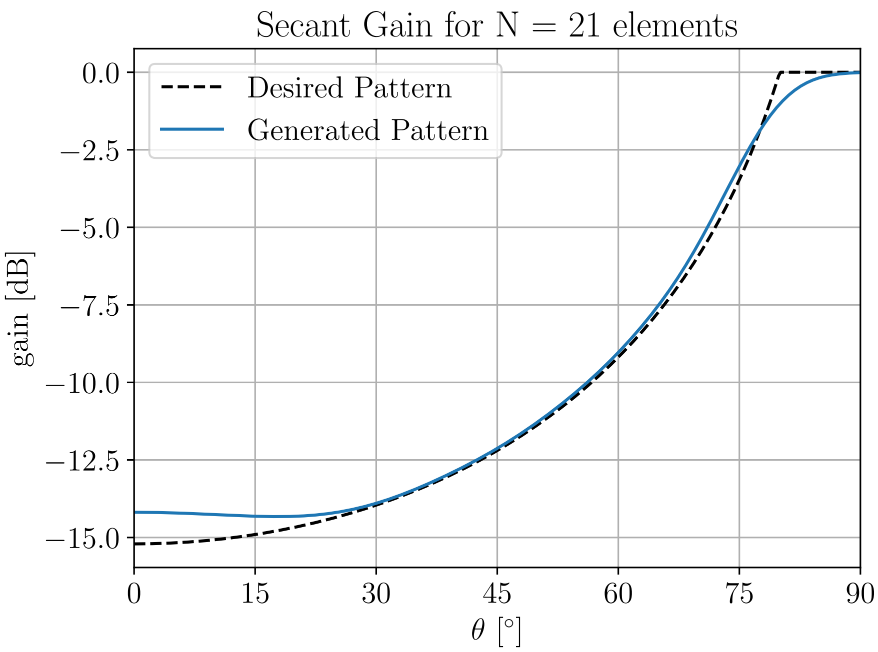

Figure A.2: Designed array factor of a secant gain with $\theta_{\text{max}} = 80°$ for $N = 21$ elements spaced of $d = \lambda/2$ placed along the z-axis.

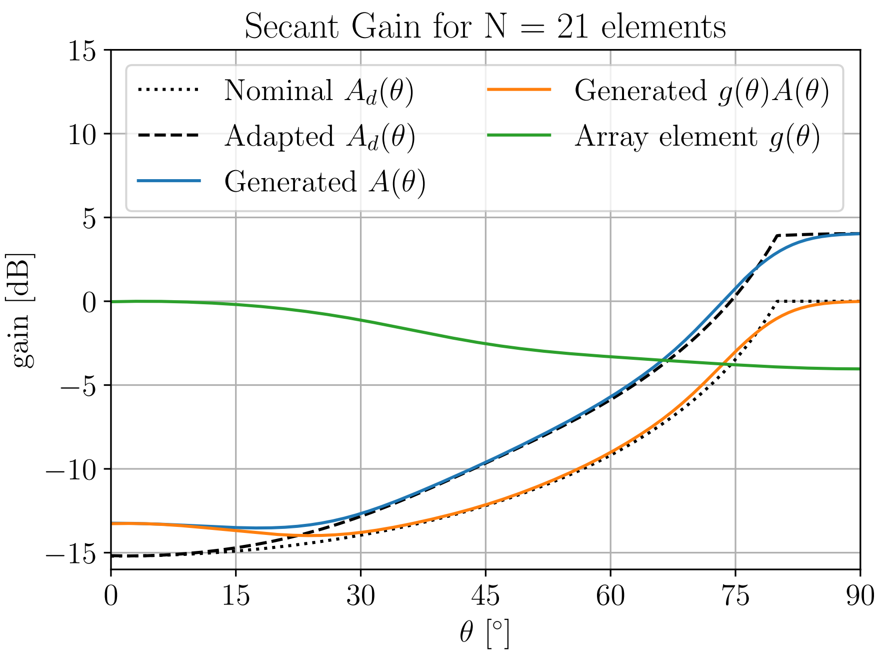

Figure A.3: Designed array factor of a secant gain with $\theta_{\text{max}} = 80°$ for $N = 21$ elements spaced of $d = \lambda/2$ placed along the z-axis.

A.9 Taylor One-Parameter Source

This section employs the Kaiser window for designing a narrow beam array, a problem analogous to the spectral analysis of windowed sinusoids [3] [5] [2].

Taylor's one-parameter continuous line source has current $I(x)$ flowing along the x-axis and corresponding radiation pattern $F(u)$ given by the Fourier transform pair [1]:

$$F(u) = \frac{\sinh\left(\pi\sqrt{B^2 - u^2}\right)}{\pi\sqrt{B^2 - u^2}} \overset{FT}{\Longleftrightarrow} I(x) = I_0\left(\pi B\sqrt{1 - (2x/l)^2}\right), \tag{A.43}$$

where x is the space region limiting the current, $-l/2 \leq x \leq l/2$, $I_0(\cdot)$ is the modified Bessel function of first kind and zeroth order, B is a positive parameter that controls the sidelobe level and u is the normalized wavenumber defined by:

$$u = \frac{lk_x}{2\pi} \iff k_x = \frac{2\pi u}{l} \iff u = \frac{l}{\lambda}\sin\theta\cos\phi. \tag{A.44}$$

For $u > B$, the pattern becomes a sinc-pattern in the variable $\sqrt{u^2 - B^2}$, and for large u, it tends to the pattern of the uniform line source.

This method of design uses array weights equal to the window coefficients that is obtained from using $x_m = md$ with $d = l/(2M)$ in (A.43). This way, the parameter B or $\alpha = \pi B$ controls the sidelobe level:

$$a(m) = w(m) = I_0\left(\alpha\sqrt{1 - m^2/M^2}\right), \tag{A.45}$$

where

$$m = \begin{cases} \pm 1, \pm 2, \ldots, \pm M, & \text{for even array elements, } N = 2M, \text{or} \\ 0, \pm 1, \pm 2, \ldots, \pm M, & \text{for odd array elements, } N = 2M + 1. \end{cases} \tag{A.46}$$

The continuous line pattern of (A.43),

$$F(u) = \frac{\sinh\left(\pi\sqrt{B^2 - u^2}\right)}{\pi\sqrt{B^2 - u^2}} = \frac{\sin\left(\pi\sqrt{u^2 - B^2}\right)}{\pi\sqrt{u^2 - B^2}}, \tag{A.47}$$

has a first null at $u_0 = \sqrt{B^2 + 1}$, and, therefore, the first sidelobe will occur for $u > u_0$. For this range, it must be used the sinc-form of $F(u)$ and it is possible to find the peak sidelobe of $\text{sinc}(x)$, $r_0 = 0.2172$. This value corresponds, in db, to $R_0 = 13.26\,\text{dB}$. For $R \leq R_0$, $w(m)$ becomes the rectangular window, and, therefore, $B = 0$.

The sidelobe level R_a is defined as the ratio of pattern at $u = 0$ to the maximum sidelobe level r_0:

$$R_a = \frac{1}{r_0}\frac{\sinh(\pi B)}{\pi B}, \tag{A.48}$$

and, in dB, $R = 20\log_{10}(R_a)$. For $R \geq R_0$, it is possible to solve numerically (A.48) and find the parameter B.

The 3-dB angle is calculated by finding it in u-space, then transforming it to ψ-space and them to the ϕ-space.

The width u is given by the solution of the half-power condition:

$$|F(u)|^2 = \frac{1}{2}|F(0)|^2 \Rightarrow \frac{\sinh\left(\pi\sqrt{B^2-u^2}\right)}{\pi\sqrt{B^2-u^2}} = \frac{1}{\sqrt{2}}\frac{\sinh(\pi B)}{\pi B}. \tag{A.49}$$

Then, transforming to the ψ-space:

$$\psi_n = \frac{2\pi u}{N}, \tag{A.50}$$

where N is the number of elements in the array.

And, finally, to the ϕ-space:

$$\phi_{3\text{dB}} = \begin{cases} \dfrac{\psi_{3\text{dB}}}{kd\sin\phi_0}, & \text{for } 0<\phi_0<\pi, \text{or} \\ 2\sqrt{\dfrac{\psi_{3\text{dB}}}{kd}}, & \text{for } \phi_0 = 0, \phi_0 = \pi. \end{cases} \tag{A.51}$$

Once the B-parameter is determined, the array weights $w(m)$ can be computed from (A.45) and then steered towards an angle ϕ_0 using:

$$\begin{aligned} a(\pm m) &= e^{\mp j(m-1/2)\psi_0} w(\pm m), \qquad \text{for } N=2M, \quad m=1,2,\ldots,M, \text{ or} \\ a(m) &= e^{-jm\psi_0} w(m), \qquad \text{for } N=2M+1, \quad m=0,\pm1,\pm2,\ldots,\pm M, \end{aligned} \tag{A.52}$$

with $\psi_0 = kd\cos\phi_0$.

To avoid grating lobes, the element spacing must be less then the maximum:

$$d_0 = \frac{\lambda}{1+|\cos\phi_0|}. \tag{A.53}$$

And, in order for the visible region in ψ-space to cover at least one Nyquist period, the element spacing d must be in the range:

$$\frac{d_0}{2} \le d \le d_0. \tag{A.54}$$

A.10 Multibeam Array

An array has the capability to generate multiple narrow beams directed towards different angles. Generally, for a odd number of array elements, $N = 2M+1$, it is possible to form L beams towards the angles $\phi_i, i = 1,2,\ldots,L$ by superimposing the steered beams:

$$a(m) = \sum_{i=1}^{L} A_i e^{-jm\psi_i} w(m), \qquad m = 0,\pm1,\pm2,\ldots,\pm M, \tag{A.55}$$

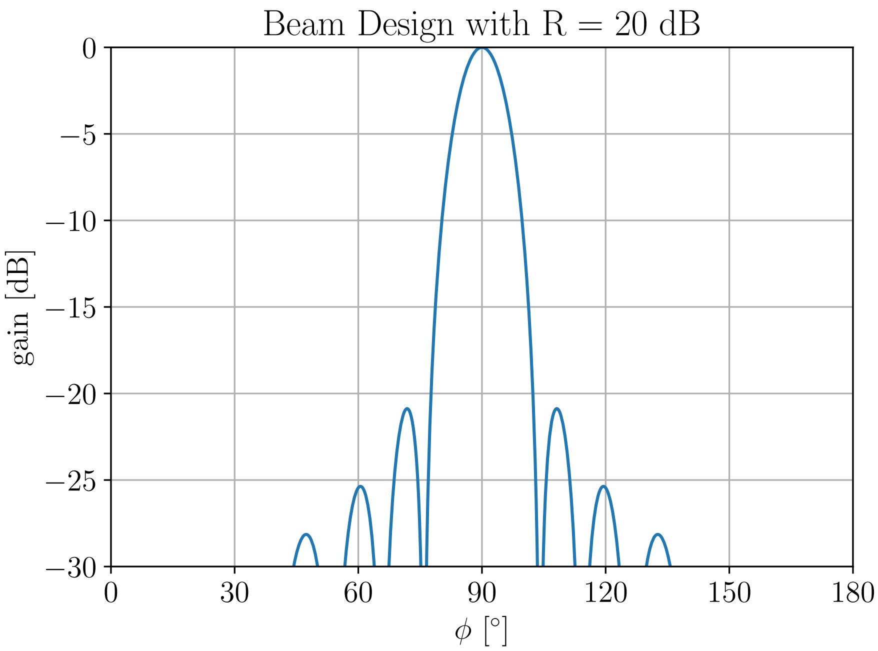

Figure A.4: Array factor designed with Taylor method. It has $N = 21$ elements spaced of $d = \lambda/4$ placed along the x-axis with sidelobe atennuation of $R = 20\,\text{dB}$.

where $\psi_i = kd\cos\phi_i, i = 1, 2, \ldots, L$ and A_i are the complex amplitudes that represents the relative importance of the beams.

For an even number of array elements, $N = 2M$, the equation becomes:

$$a(\pm m) = \sum_{i=1}^{L} A_i e^{\mp j(m-1/2)\psi_i} w(\pm m), \qquad m = 0, \pm 1, \pm 2, \ldots, \pm M. \tag{A.56}$$

For both cases, the corresponding array factor will be the superposition:

$$A(\psi) = \sum_{i=i}^{L} A_i W(\psi - \psi_i). \tag{A.57}$$

As an example, consider a beam designed with the Taylor method presented in Section A.9 with $N = 21$ elements spaced of $d = \lambda/4$ and $R = 20\,\text{dB}$. The result is presented in Figure A.4.

Then, consider $L = 8$ beams each one steered of $\phi_{i+1} = \phi_i + 2\phi_{3\text{dB}}$. This way, the patterns merge with each other. The result is shown in Figure A.5.

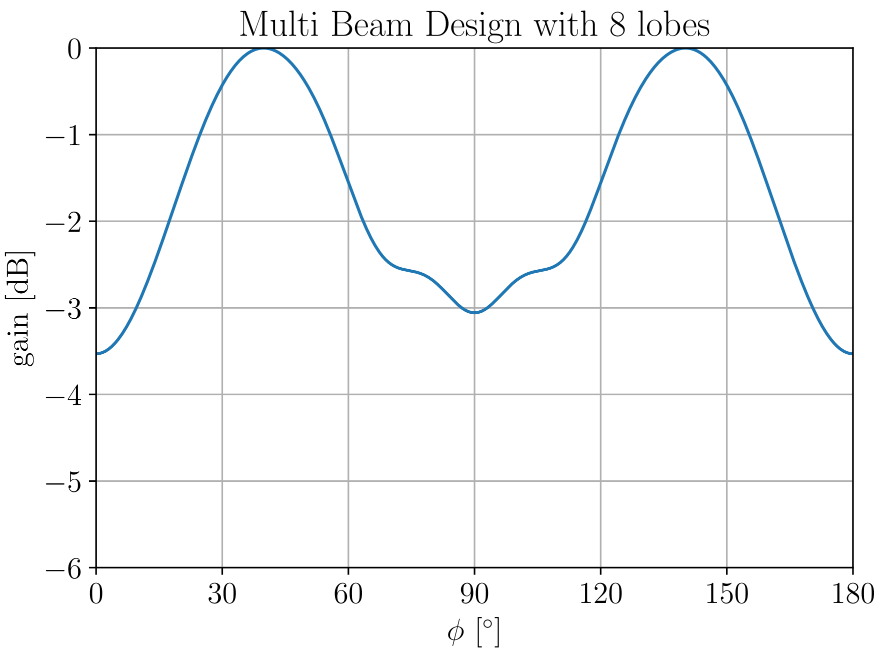

Figure A.5: Multibeam array designed with $L = 8$ lobes with steering angles close to each other about one 3-dB beamwidth.

Appendix B

Scripts

B.1 Parabolic Reflector Gain

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
font_size = 15
plt.rcParams.update({'font.size': font_size})
pandas.set_option('expand_frame_repr', False)

if __name__ == '__main__':
    f = 8363e6
    lamb = c0 / f
    theta = numpy.linspace(0, numpy.pi, 2000)
    phi = numpy.full_like(theta, 0)
    parabola = Parabola(f=f, D=1 / lamb, eff=0.75, theta=theta, phi=phi,
                        use_parallel=True)
    parabola.calc_gain_pattern_sym()
    fig_sym, axes_sym = parabola.plot_gain_pattern_sym()
    parabola.calc_gain_pattern()
    fig_comparison, axes_comparison = parabola.plot_gain_pattern_comparison()
```

B.2 MicroStrip Design

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', 500)
```

```


if __name__ == '__main__':
    # This method of project is following Balanis Chapter 14.2.C
    f = 10e9
    lamb = c0 / f
    eps_r = 2.2
    h = 0.1588e-2
    parameters = {
        "h": h,
        "eps_r": eps_r,
        "f": f
    }
    strip = MicroStrip(**parameters, hfss_path='hfss_data/E.csv')

    step_theta_hfss = 1
    step_phi_hfss = 1
    theta_hfss = numpy.arange(-180, 180 + step_theta_hfss, step_theta_hfss)
    phi_hfss = numpy.arange(-180, 180 + step_phi_hfss, step_phi_hfss)
    phi_hfss[phi_hfss < 0] += 360
    theta_grid = deg2rad(theta_hfss)
    n_theta_hfss = len(theta_hfss)
    n_phi_hfss = len(phi_hfss)

    hfss_e_db = pandas.read_csv('hfss_data/E_db_full_theta.csv', skiprows=1,
        header=None, index_col=0)
    hfss_e_tot_db, hfss_e_phi_db, hfss_e_theta_db = parse_hfss_data(hfss_e_db,
        phi_hfss)

    e_plane_phi = 0
    h_plane_phi = 90
    y_min = -30

    # Electric fields before rotation
    e_theta_nominal, e_phi_nominal = strip.e_analytical(theta_grid, deg2rad(
        e_plane_phi))
    e_tot_nominal = numpy.sqrt(numpy.abs(e_theta_nominal) ** 2 + numpy.abs(
        e_phi_nominal) ** 2)

    h_theta_nominal, h_phi_nominal = strip.e_analytical(theta_grid, deg2rad(
        h_plane_phi))
    h_tot_nominal = numpy.sqrt(numpy.abs(h_theta_nominal) ** 2 + numpy.abs(
        h_phi_nominal) ** 2)

    fig_size = 3
    fig, axes = plt.subplots(nrows=3, ncols=2, sharex=True, sharey=True,
        figsize=(3 * fig_size, 2 * fig_size))
    fig.suptitle(f"Analytical model vs HFSS")
    axes[0][0].set_title(f"E-Plane ($\phi = 0^\circ$)")
    axes[0][1].set_title(f"H-Plane ($\phi = 90^\circ$)")
    axes[0][0].set_ylabel(r"$E_\theta [^\circ]$ [dB]")
    axes[1][0].set_ylabel(r"$E_\phi$ [dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$ [dB]")
    axes[2][0].set_xlabel(r"$\theta [^\circ]$")
    axes[2][1].set_xlabel(r"$\theta [^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db[e_plane_phi])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db[e_plane_phi])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db[e_plane_phi])
```

```
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_nominal), db=20))
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_nominal), db=20))
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_nominal), db=20))

    axes[0][1].plot(theta_hfss, hfss_e_theta_db[h_plane_phi], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db[h_plane_phi])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db[h_plane_phi])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_nominal), db=20), label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_nominal), db=20))
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_nominal), db=20))
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 1, step=6))
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
            _ax.set_xlim([-90, 90])
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Antenna Array/hfss_vs_analytical.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)

    filtered_theta_deg = numpy.linspace(0, 180, 181, dtype=int)
    filtered_theta = deg2rad(filtered_theta_deg)
    filtered_phi_deg = numpy.linspace(0, 360, 361, dtype=int)
    filtered_phi = deg2rad(filtered_phi_deg)
    Theta, Phi = numpy.meshgrid(filtered_theta, filtered_phi, indexing='ij')
    e_theta_nominal, e_phi_nominal = strip.e_analytical(Theta.flatten(), Phi.flatten())
    e_theta_nominal = numpy.abs(e_theta_nominal)
    e_phi_nominal = numpy.abs(e_phi_nominal)
    e_tot_nominal = numpy.sqrt(numpy.abs(e_theta_nominal) ** 2 + numpy.abs(e_phi_nominal) ** 2).reshape(Theta.shape)

    e_theta_hfss = to_db(numpy.abs(strip.e_theta[filtered_phi_deg].loc[filtered_theta_deg]).to_numpy(), db=20)
    e_phi_hfss = to_db(numpy.abs(strip.e_phi[filtered_phi_deg].loc[filtered_theta_deg]).to_numpy(), db=20)
    e_tot_hfss = to_db(strip.e_tot[filtered_phi_deg].loc[filtered_theta_deg].to_numpy(), db=20)



    fig_fields, axes_fields = plt.subplots(nrows=3, ncols=2, sharex=True, sharey=True, figsize=(10, 12),
                                           subplot_kw={'projection': 'polar'})

    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[0][0], filtered_theta, filtered_phi, e_theta_hfss,
                                      theta_max_deg=90, theta_step=30)
    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[0][1], Theta, Phi,
                                      to_db(e_theta_nominal.reshape(Theta.shape), db=20),
                                      theta_max_deg=90, theta_step=30)

```

```
    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[1][0],
       filtered_theta, filtered_phi, e_phi_hfss,
                                      theta_max_deg=90, theta_step=30)
    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[1][1], Theta, Phi
       ,
                                      to_db(e_phi_nominal.reshape(Theta.shape),
                                          db=20),
                                      theta_max_deg=90, theta_step=30)

    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[2][0],
       filtered_theta, filtered_phi, e_tot_hfss,
                                      theta_max_deg=90, theta_step=30)
    plot_polar_contour_mag_db_in_axis(fig_fields, axes_fields[2][1], Theta, Phi
       , to_db(e_tot_nominal, db=20),
                                      theta_max_deg=90, theta_step=30)
    axes_fields[0][0].set_title("HFSS")
    axes_fields[0][1].set_title("Analytical")
    axes_fields[0][0].set_ylabel("$|E_\\theta|$")
    axes_fields[1][0].set_ylabel("$|E_\\phi|$")
    axes_fields[2][0].set_ylabel("$|E_{tot}|$")
    axes_fields[0][1].set_ylabel("$|E_\\theta|$")
    axes_fields[1][1].set_ylabel("$|E_\\phi|$")
    axes_fields[2][1].set_ylabel("$|E_{tot}|$")
    plt.subplots_adjust(wspace=0.2, hspace=0.45)
    fig_fields.set_tight_layout(True)
    fig_fields.savefig(f'../parts/Antenna Array/hfss_vs_analytical_polar.pdf',
                       transparent=True, bbox_inches='tight', pad_inches=0)
```

B.3 MicroStrip Analytical Rotation

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', 500)


if __name__ == '__main__':
    # This method of project is following Balanis Chapter 14.2.C
    f = 10e9
    lamb = c0 / f
    eps_r = 2.2
    h = 0.1588e-2
    parameters = {
        "h": h,
        "eps_r": eps_r,
        "f": f
    }
    strip = MicroStrip(**parameters, hfss_path='hfss_data/E.csv')
```

```

    beta = deg2rad(45)

    step_theta_hfss = 1
    step_phi_hfss = 1
    theta_hfss = numpy.arange(-180, 180 + step_theta_hfss, step_theta_hfss)
    phi_hfss = numpy.arange(-180, 180 + step_phi_hfss, step_phi_hfss)
    phi_hfss[phi_hfss < 0] += 360
    theta_grid = deg2rad(theta_hfss)
    n_theta_hfss = len(theta_hfss)
    n_phi_hfss = len(phi_hfss)

    filtered_theta_deg = numpy.linspace(0, 180, 181, dtype=int)
    filtered_theta = deg2rad(filtered_theta_deg)
    filtered_phi_deg = numpy.linspace(0, 360, 361, dtype=int)
    filtered_phi = deg2rad(filtered_phi_deg)
    Theta, Phi = numpy.meshgrid(filtered_theta, filtered_phi, indexing='ij')

    hfss_e_db_steered = pandas.read_csv('hfss_data/E_db_beta_45_full_theta.csv', skiprows=1, header=None, index_col=0)
    hfss_e_tot_db_steered, hfss_e_phi_db_steered, hfss_e_theta_db_steered = parse_hfss_data(hfss_e_db_steered, phi_hfss)

    e_plane_phi = 0
    h_plane_phi = 90
    y_min = -30

    # Electric fields after rotation
    e_theta_steered, e_phi_steered = strip.e_rotated(Theta.flatten(), Phi.flatten(), 0, -beta, 0)
    # e_theta_steered, e_phi_steered = general_rotated_fields_func(Theta.flatten(), Phi.flatten(), 0, -beta, 0)
    e_tot_steered = numpy.sqrt(numpy.abs(e_theta_steered) ** 2 + numpy.abs(e_phi_steered) ** 2)
    max_rotated = numpy.max(e_tot_steered)

    e_theta_steered, e_phi_steered = strip.e_rotated(theta_grid, numpy.full_like(theta_grid, deg2rad(e_plane_phi)), 0, -beta, 0)
    e_tot_steered = numpy.sqrt(numpy.abs(e_theta_steered) ** 2 + numpy.abs(e_phi_steered) ** 2)
    e_theta_steered = e_theta_steered / max_rotated
    e_phi_steered = e_phi_steered / max_rotated
    e_tot_steered = e_tot_steered / max_rotated

    h_steered = strip.e_rotated(theta_grid, numpy.full_like(theta_grid, deg2rad(h_plane_phi)), 0, -beta, 0)
    h_theta_steered = h_steered[0]
    h_phi_steered = h_steered[1]
    h_tot_steered = numpy.sqrt(numpy.abs(h_theta_steered) ** 2 + numpy.abs(h_phi_steered) ** 2)
    h_theta_steered = h_theta_steered / max_rotated
    h_phi_steered = h_phi_steered / max_rotated
    h_tot_steered = h_tot_steered / max_rotated

    fig_size = 3
    fig, axes = plt.subplots(nrows=3, ncols=2, sharey=True, sharex='col', figsize=(3 * fig_size, 2 * fig_size))
    axes[0][0].set_title(f"E-Plane␣($\phi␣=␣0^\circ$)")
    axes[0][1].set_title(f"H-Plane␣($\phi␣=␣90^\circ$)")
```

```
    fig.suptitle(f"Analytical␣model␣vs␣HFSS␣-␣Steered␣$\\beta␣=␣{rad2deg(beta):0.0f}^\circ$")
    axes[0][0].set_title(f"E-Plane␣($\phi␣=␣0^\circ$)")
    axes[0][1].set_title(f"H-Plane␣($\phi␣=␣90^\circ$)")
    axes[0][0].set_ylabel(r"$E_\theta␣[^\circ]$␣[dB]")
    axes[1][0].set_ylabel(r"$E_\phi$␣[dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$␣[dB]")
    axes[2][0].set_xlabel(r"$\theta␣[^\circ]$")
    axes[2][1].set_xlabel(r"$\theta␣[^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_steered[e_plane_phi])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_steered[e_plane_phi])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_steered[e_plane_phi])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_steered), db=20))
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_steered), db=20))
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_steered), db=20))

    axes[0][1].plot(theta_hfss, hfss_e_theta_db_steered[h_plane_phi], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_steered[h_plane_phi])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_steered[h_plane_phi])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_steered), db=20), label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_steered), db=20))
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_steered), db=20))
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    for ax in axes[:, 0]:
        ax.set_xlim([-90 + rad2deg(beta), 90 + rad2deg(beta)])
        ax.set_xticks(numpy.arange(-90 + rad2deg(beta), 91 + rad2deg(beta), step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Antenna␣Array/hfss_vs_analytical_steered.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)
```

B.4 MicroStrip HFSS Analytically Steered

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
```

```python
pandas.set_option('display.max_rows', 500)


if __name__ == '__main__':
    # This method of project is following Balanis Chapter 14.2.C
    f = 10e9
    lamb = c0 / f
    eps_r = 2.2
    h = 0.1588e-2
    parameters = {
        "h": h,
        "eps_r": eps_r,
        "f": f
    }
    strip = MicroStrip(**parameters, hfss_path='hfss_data/E.csv')
    beta = deg2rad(45)

    step_theta_hfss = 1
    step_phi_hfss = 1
    theta_hfss = numpy.arange(-180, 180 + step_theta_hfss, step_theta_hfss)
    phi_hfss = numpy.arange(-180, 180 + step_phi_hfss, step_phi_hfss)
    phi_hfss[phi_hfss < 0] += 360
    theta_grid = deg2rad(theta_hfss)

    filtered_theta_deg = numpy.linspace(0, 180, 181, dtype=int)
    filtered_theta = deg2rad(filtered_theta_deg)
    filtered_phi_deg = numpy.linspace(0, 360, 361, dtype=int)
    filtered_phi = deg2rad(filtered_phi_deg)
    Theta, Phi = numpy.meshgrid(filtered_theta, filtered_phi, indexing='ij')

    hfss_e_db_steered = pandas.read_csv('hfss_data/E_db_beta_45_full_theta.csv'
        , skiprows=1, header=None, index_col=0)
    hfss_e_tot_db_steered, hfss_e_phi_db_steered, hfss_e_theta_db_steered =
        parse_hfss_data(hfss_e_db_steered, phi_hfss)

    e_plane_phi = 0
    h_plane_phi = 90
    y_min = -30

    e_theta_steered_hfss, e_phi_steered_hfss = strip.e_rotated(theta_grid,
        numpy.full_like(theta_grid, deg2rad(e_plane_phi)),
                                       0, -beta, 0, hfss=True)
    e_tot_steered_hfss = numpy.sqrt(numpy.abs(e_theta_steered_hfss)**2 + numpy.
        abs(e_phi_steered_hfss)**2)
    e_theta_steered_hfss = e_theta_steered_hfss
    e_phi_steered_hfss = e_phi_steered_hfss
    e_tot_steered_hfss = e_tot_steered_hfss

    h_theta_steered_hfss, h_phi_steered_hfss = strip.e_rotated(theta_grid,
        numpy.full_like(theta_grid, deg2rad(h_plane_phi)),
                                       0, -beta, 0, hfss=True)
    h_tot_steered_hfss = numpy.sqrt(numpy.abs(h_theta_steered_hfss)**2 + numpy.
        abs(h_phi_steered_hfss)**2)
    h_theta_steered_hfss = h_theta_steered_hfss
    h_phi_steered_hfss = h_phi_steered_hfss
    h_tot_steered_hfss = h_tot_steered_hfss

    # Using field from hfss
    fig_size = 3
```

```
    fig, axes = plt.subplots(nrows=3, ncols=2, sharey=True, sharex='col', figsize=(3*fig_size, 2*fig_size))
    axes[0][0].set_title(f"E-Plane␣($\phi␣=␣0^\circ$)")
    axes[0][1].set_title(f"H-Plane␣($\phi␣=␣90^\circ$)")
    fig.suptitle(f"HFSS␣Analytically␣Steered␣vs␣HFSS␣Steered␣-␣$\\beta␣=␣{rad2deg(beta):0.0f}^\circ$")
    axes[0][0].set_title(f"E-Plane␣($\phi␣=␣0^\circ$)")
    axes[0][1].set_title(f"H-Plane␣($\phi␣=␣90^\circ$)")
    axes[0][0].set_ylabel(r"$E_\theta␣[^\circ]$␣[dB]")
    axes[1][0].set_ylabel(r"$E_\phi$␣[dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$␣[dB]")
    axes[2][0].set_xlabel(r"$\theta␣[^\circ]$")
    axes[2][1].set_xlabel(r"$\theta␣[^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_steered[e_plane_phi].loc[theta_hfss])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_steered[e_plane_phi].loc[theta_hfss])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_steered[e_plane_phi].loc[theta_hfss])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_steered_hfss), db=20), '-.', markersize=10)
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_steered_hfss), db=20), '-.', markersize=10)
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_steered_hfss), db=20), '-.', markersize=10)

    axes[0][1].plot(theta_hfss, hfss_e_theta_db_steered[h_plane_phi].loc[theta_hfss], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_steered[h_plane_phi].loc[theta_hfss])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_steered[h_plane_phi].loc[theta_hfss])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_steered_hfss), db=20), '-.', label="Analytical", markersize=10)
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_steered_hfss), db=20), '-.', markersize=10)
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_steered_hfss), db=20), '-.', markersize=10)
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    for ax in axes[:, 0]:
        pass
        ax.set_xlim([-90 + rad2deg(beta), 90 + rad2deg(beta)])
        ax.set_xticks(numpy.arange(-90 + rad2deg(beta), 91 + rad2deg(beta), step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Theoretical␣Foundation/hfss_analytically_rotated_vs_hfss.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)
```

B.5 Link Budget Example

```python
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
from datetime import datetime
from shapely.geometry import Point
import multiprocessing

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
# use('TkAgg')
plt.rcParams.update({'font.size': 18})
pandas.set_option('expand_frame_repr', False)

if __name__ == '__main__':
    num_cores = multiprocessing.cpu_count()
    shapefile = Path(__file__).parent / "input/brazil_Brazil_Country_Boundary.shp"
    n_lon = 40
    n_lat = 40
    brazil_points, main_land = generate_grid_from_shapefile(shapefile, n_lat=n_lat, n_lon=n_lon)
    f = 2244e6
    lamb = c0 / f
    start = datetime.now()
    vcub1 = Satellite(eirp=32 - 30, start_time='2023-06-01 00:00:00.000', end_time='2023-07-01 00:00:00.000',
                      N=5000000, line1='1 56215U 23054AP  23188.40114352  .00016899  00000+0  67546-3 0  9998',
                      line2='2 56215  97.4015  83.1853 0008227 292.9660  67.0710 15.25154711 13184', f=f)
    save_pickle(vcub1, './input/satellite_vcub1.pkl')
    # vcub1 = read_pickle('./input/satellite_vcub1.pkl')
    end = datetime.now()
    print(f"Elapsed {(end - start).total_seconds():5.2f} seconds in satellite")
    long_min, lat_min, long_max, lat_max = main_land.bounds
    station_lon, station_lat = main_land.centroid.coords[0]
    start = datetime.now()
    ground_parabola = Parabola(f=f, D=2.6 / lamb, eff=0.67)
    Eb_N0_min = 7.5
    station_parameters = {'f': f, 'eff': 0.67, 'temp': 312, 'bandwidth': 6e6}
    link_parameters = {'satellite': vcub1, 'R_spec': 10e6, 'Eb_N0_min': Eb_N0_min, 'calc_transmitted_data': True, 'G_other':-1.6}
    ground_station = Station(lon=station_lon, lat=station_lat, e_theta_e_phi_function=ground_parabola.get_fields_sym, **station_parameters,
                             G_max=ground_parabola.G_max)
    link_cope = LinkBudget(station=ground_station, **link_parameters)
    end = datetime.now()
    print(f"Elapsed {(end - start).total_seconds():5.2f} seconds")
    start = datetime.now()
    end = datetime.now()
    print(f"Elapsed {(end - start).total_seconds():5.2f} seconds in link calculation")
```

```

    longs = numpy.unique(brazil_points['lon'].values)
    lats = numpy.unique(brazil_points['lat'].values)

    fig, axes = link_cope.plot_contour_eb_n0(save=False, path="../parts/Link Budget/contour_eb_n0.png")
    axes.plot(*main_land.exterior.coords.xy, color='black')
    axes.set_xlim([long_min, long_max])
    axes.set_ylim([lat_min, lat_max])
    axes.plot(station_lon, station_lat, 'x', color='black')
    axes.set_title('Link Budget Example')
    axes.set_aspect('equal')
    plt.show()
```

B.6 Brazil Grid

```
from arraytools import *
use('Qt5Agg')
pandas.set_option('expand_frame_repr', False)
font_size = 15
plt.rcParams.update({'font.size': font_size})
rc('text', usetex=True)
rc('font', family='serif')


if __name__ == '__main__':
    shapefile = Path(__file__).parent / "input/brazil_Brazil_Country_Boundary.shp"
    n_lon = 50
    n_lat = 50
    # df, main_land = generate_grid_from_shapefile(shapefile, n_lat=n_lat, n_lon=n_lon)
    main_land = brazil_mainland()
    contains_func = main_land.contains
    long_min, lat_min, long_max, lat_max = main_land.bounds
    longs, lats = numpy.meshgrid(numpy.linspace(long_min, long_max, n_lon), numpy.linspace(lat_min, lat_max, n_lat), indexing='ij')

    df = pandas.DataFrame({
        'lon': longs.flatten(),
        'lat': lats.flatten(),
        'coord': tuple(zip(longs.flatten(), lats.flatten()))
    })

    df['point'] = df['coord'].apply(shapely.geometry.Point)

    df['inside'] = df['point'].apply(contains_func)

    # Plotting result
    fig, axes = plt.subplots()
    lon = df[~df['inside']]['lon'].values
    lat = df[~df['inside']]['lat'].values
    axes.plot(lon, lat, 'o', color='red', markersize=1)
    lon = df[df['inside']]['lon'].values
    lat = df[df['inside']]['lat'].values
    axes.plot(lon, lat, 'o', color='green', markersize=1)
```

```
    axes.plot(*main_land.exterior.coords.xy, color='black')
    axes.set_aspect('equal')
    axes.set_xlabel('Degrees Longitude')
    axes.set_ylabel('Degrees Latitude')
    axes.set_title(f'Brazil {n_lon}x{n_lat} grid')
    fig.set_size_inches(8, 6)
    axes.set_xlim([long_min, long_max])
    axes.set_ylim([lat_min, lat_max])
    fig.tight_layout()
    plt.show()

    # fig.savefig(f'./graphs/brazil_points_{n_lon}x{n_lat}_grid.png',
       transparent=True, bbox_inches='tight', pad_inches=0, dpi=300)
    df.to_pickle('./input/brazil_points_sacc.zip')
```

B.7 Creating Database and A matrix

```
from shapely.prepared import prep
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use
import shapely
from shapely.geometry import Point
import multiprocessing
from alive_progress import alive_bar

use('Qt5Agg')
plt.rcParams.update({'font.size': 20})
pandas.set_option('expand_frame_repr', False)

if __name__ == '__main__':
    num_cores = multiprocessing.cpu_count()
    f = 2244e6
    lamb = c0 / f
    # sat = Satellite(eirp=32 - 30, start_time='2023-07-01 00:00:00.000',
       end_time='2023-07-31 23:59:59.999',
    #                  N=5000000, line1='1 56215U 23054AP   23188.40114352
       .00016899  00000+0  67546-3 0  9998',
    #                  line2='2 56215  97.4015  83.1853 0008227 292.9660
       67.0710 15.25154711 13184')
    # save_pickle(sat, './input/satellite_vcub1.pkl')
    sat = read_pickle('./input/satellite_vcub1.pkl')
    brazil_points = pandas.read_pickle('input/brazil_points_vcub1.zip')
    brazil_points_inside = brazil_points[brazil_points.inside]
    main_land = brazil_mainland()
    main_land_contains = main_land.contains
    long_min, lat_min, long_max, lat_max = main_land.bounds
    longs = numpy.unique(brazil_points['lon'].values)
    lats = numpy.unique(brazil_points['lat'].values)
    brazil_points['transmitted_data'] = 0
    Eb_N0_min = 7.5
    a = numpy.zeros(shape=(len(brazil_points.index), len(lats), len(longs)))
    with alive_bar(len(brazil_points.index), force_tty=True) as bar:
        for index, row in brazil_points.iterrows():
```

```
            bar()
            a_matrix = numpy.zeros(shape=(len(lats), len(longs)))
            if row.inside:
                # Calculating antenna
                ground_parabola = Parabola(f=f, D=2.6 / lamb)
                station = Station(f=f, lat=row.lat, lon=row.lon, e_theta_e_phi_function=ground_parabola.get_fields_sym,
                                  bandwidth=6e6, eff=0.5, temp=312, G_max=ground_parabola.G_max)
                link = LinkBudget(satellite=sat, station=station, R_spec=10e6, Eb_N0_min=Eb_N0_min,
                                  calc_transmitted_data=True, G_other=-1.6)
                link_data = link.data[['lon_WGS84_deg', 'lat_WGS84_deg', 'Eb_N0']].copy()
                link_data.rename(columns={
                    'lon_WGS84_deg': 'lon',
                    'lat_WGS84_deg': 'lat'
                }, inplace=True)
                coords = tuple(zip(link_data['lon'].values, link_data['lat'].values))
                points = [Point(coord) for coord in coords]
                link_data['radius'] = shapely.distance(points, row.point)
                link_data.to_pickle(f"database/vcub1/antennas/link_data_south_america/{index}.zip")
                brazil_points.loc[index, 'transmitted_data'] = link.total_data[0]

                antenna = link.get_shapely_contour()
                prep_shape = prep(antenna)
                inside_mask = numpy.array([prep_shape.contains(point) for point in brazil_points.point.values])
                inside_points = brazil_points[inside_mask]['point'].values
                for point in inside_points:
                    try:
                        long_idx = numpy.where(longs == point.x)[0][0]
                        lat_idx = numpy.where(lats == point.y)[0][0]
                        a_matrix[lat_idx, long_idx] = 1
                    except IndexError:
                        pass
            else:
                pandas.DataFrame().to_pickle(f"database/vcub1/antennas/link_data_south_america/{index}.zip")
            a[index] = a_matrix
    brazil_points.to_pickle('input/vcub1_brazil_points_with_transmission_south_america.zip')
    save_pickle(a, f'./input/a_matrix_{int(10*Eb_N0_min):03d}_binary_vcub1_south_america.pkl')
    index_inside = brazil_points_inside.index[300]
    fig, axis = plot_contour_eb_n0_from_matrix(a[index_inside], main_land, longs, lats)
    plt.show()
```

B.8 Convex Optimization

```
from arraytools import *
import numpy
```

```python
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
import cvxpy as cp

use('Qt5Agg')
rc('text', usetex=True)
rc('font', family='serif')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)

if __name__ == '__main__':
    brazil_points = pandas.read_pickle('input/brazil_points_vcub1.zip')
    brazil_points_inside = brazil_points[brazil_points.inside].copy()
    main_land = brazil_mainland()
    long_min, lat_min, long_max, lat_max = main_land.bounds
    longs = numpy.unique(brazil_points['lon'].values)
    lats = numpy.unique(brazil_points['lat'].values)
    longs_inside = numpy.unique(brazil_points[brazil_points.inside]['lon'].
        values)
    lats_inside = numpy.unique(brazil_points[brazil_points.inside]['lat'].
        values)
    satellite = read_pickle('./input/satellite_vcub1.pkl')
    Eb_N0_min = 7.5
    warm_start = False
    for Eb_N0_min in [7.5]:
        a = read_pickle(f'./input/a_matrix_{int(10 * Eb_N0_min):03d}
            _binary_vcub1.pkl')
        N = len(a)
        M, P = a[0].shape
        A = numpy.zeros((N, M * P))
        f_d = numpy.zeros(shape=(M, P))
        for _index, _row in brazil_points[brazil_points.inside].iterrows():
            coord = _row.coord
            try:
                long_idx = numpy.searchsorted(longs, coord[0], side="left")
                lat_idx = numpy.searchsorted(lats, coord[1], side="left")
                f_d[lat_idx, long_idx] = 1
            except IndexError:
                pass
        f_d = f_d.flatten()
        inside_mask = f_d == 1
        inside_total = len(f_d[inside_mask])
        for k in range(N):
            A[k, :] = a[k].flatten()
        for epsilon in [20, 10, 9, 8, 7]:
            print("=====================================================")
            print(f"Starting for Eb_N0 = {Eb_N0_min:02f} and epsilon = {epsilon
                :02f}")
            # x = cp.Variable(N, integer=True)
            if warm_start:
                x0 = numpy.load(
                    f'arrays/vcub1/xbest_cvxpy_boolean_{int(10 * Eb_N0_min):03d
                        }_eps_{int(10 * (epsilon+1)):03d}.npy')
                x = cp.Variable(N, boolean=True, value=x0)
            else:
                x = cp.Variable(N, boolean=True)
            objective = cp.Minimize(cp.norm(x, 1))
            constraints = [cp.norm2((x @ A) - f_d) <= epsilon]
```

```
            prob = cp.Problem(objective, constraints)
            print(prob.solve(verbose=True, warm_start=warm_start, solver='SCIP',
                                 scip_params={'limits/time': 3600*4}))
            delta = 1e-2
            threshold = 0.9

            _fig, _axes = plt.subplots()
            _axes.plot(x.value, '.')
            _axes.axhline(threshold, color='red', label='threshold')
            _fig.set_size_inches(8, 6)
            _fig.savefig(
                f'graphs/vcub1/chosen_antennas_Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_cvxpy.pdf',
                transparent=True, bbox_inches='tight')
            plt.close(_fig)
            numpy.save(
                f'arrays/vcub1/xbest_cvxpy_boolean_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}.npy',
                x.value)
            _x = x.value.copy()
            _x[_x <= threshold] = 0
            _x[_x > threshold] = 1
            fig, axes = plot_contour_eb_n0_from_matrix((_x @ A).reshape(M, P), main_land, longs, lats, Eb_N0_min)
            for index, row in brazil_points[_x == 1].iterrows():
                axes.plot(row.lon, row.lat, 'x', color='black')
            fig.savefig(
                f'graphs/vcub1/coverage_Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_cvxpy.pdf',
                transparent=True, bbox_inches='tight')
            plt.show()
```

B.9 Sequential Least Squares

```
from arraytools import *
import numpy
import pandas
from matplotlib import use, rc
from shapely.geometry import Point
import matplotlib.pyplot as plt
from shapely.geometry import LineString, LinearRing, Polygon
from shapely.plotting import plot_line, plot_points, plot_polygon
from figures import SIZE, BLACK, BLUE, GRAY, YELLOW, RED, set_limits
from shapely.ops import unary_union, nearest_points
from scipy.optimize import minimize
from shapely.prepared import prep
from itertools import combinations
rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
font_size = 13
plt.rcParams.update({'font.size': font_size})
pandas.set_option('expand_frame_repr', False)


```

```python
def combine_antennas(_antennas, _radius):
    return unary_union([Point(_ant[0], _ant[1]).buffer(_radius) for _ant in
        _antennas])


def combine_antenna_xy(_x, _y, _radius):
    return unary_union([Point(point[0], point[1]).buffer(radius) for point in
        tuple(zip(_x.value, _y.value))])


def minimize_antennas(x0, region_boundary, radius, desired_coverage=0.99):
    N = len(x0)
    x0 = x0.flatten()
    brazil_area = region_boundary.area
    region_prep = prep(region_boundary)
    long_min, lat_min, long_max, lat_max = region_boundary.bounds
    best_coverage_area = N * numpy.pi * radius ** 2
    max_iterations = 10000

    def func_to_minimize(x):
        x = x.reshape(-1, 2)
        coverage = combine_antennas(x, radius)
        return (best_coverage_area - coverage.area)

    def constraint_function(x):
        x = x.reshape(-1, 2)
        coverage = combine_antennas(x, radius)
        return region_boundary.intersection(coverage).area / brazil_area -
            desired_coverage

    def inside_brasil(x):
        def sigma(x):
            return 1 / (1 + numpy.exp(-x))
        x = x.reshape(-1, 2)
        violation_count = 0
        for _x in x:
            if region_prep.contains(Point(_x)):
                pass
            else:
                violation_count += 1
        return sigma(violation_count) - 0.5

    constraints = [
        {
            'type': 'ineq',
            'fun': constraint_function
        },
        {
            'type': 'eq',
            'fun': inside_brasil
        }]

    constraints = [
        {
            'type': 'ineq',
            'fun': constraint_function
        }]

    bounds = N * [[long_min, long_max], [lat_min, lat_max]]
```

```python
    res = minimize(func_to_minimize, x0, method='SLSQP', constraints=constraints, bounds=bounds,
                   options={'maxiter': max_iterations,
                            'ftol': 5e-3,
                            'eps': 1e-5})
    print("Final constraint: ", constraint_function(res.x))
    return res


if __name__ == '__main__':
    main_land = brazil_mainland()
    brazil_area = main_land.area
    main_land_contains = prep(main_land).contains
    brazil_points = pandas.read_pickle('input/brazil_points.zip')
    brazil_points_inside = brazil_points[brazil_points.inside]
    longs = numpy.unique(brazil_points['lon'].values)
    lats = numpy.unique(brazil_points['lat'].values)
    longs_inside = numpy.unique(brazil_points_inside['lon'].values)
    lats_inside = numpy.unique(brazil_points_inside['lat'].values)
    points_x = main_land.exterior.coords.xy[0]
    points_y = main_land.exterior.coords.xy[1]
    long_min, lat_min, long_max, lat_max = main_land.bounds

    threshold = 0.2
    Eb_N0_min = 10
    epsilon = 11
    all_results = []
    for Eb_N0_min in [7.5]:
        for epsilon in [8]:
            x0 = numpy.load(f'arrays/vcub1/xbest_cvxpy_boolean_{int(10*Eb_N0_min):03d}_eps_{int(10*epsilon):03d}.npy')
            x1 = x0.copy()
            x1[x1 <= threshold] = 0
            x1[x1 > threshold] = 1
            D1 = len(numpy.where(x1 == 1)[0])  # number of active antennas
            index = numpy.where(x1 == 1)[0][0]
            link_data = pandas.read_pickle(f'database/vcub1/antennas/link_data/{index}.zip')
            radius = link_data[link_data['Eb_N0'] > Eb_N0_min].sort_values('Eb_N0').iloc[0].radius
            x1 = brazil_points.loc[numpy.where(x1 == 1)][['lon', 'lat']].values.copy()

            possible_antennas = []
            for comb in combinations(x1, 2):
                ant = (comb[0] + comb[1])/2
                if ~main_land_contains(Point(ant)):
                    near = nearest_points(main_land, Point(ant))[0]
                    ant = numpy.array([near.coords.xy[0][0], near.coords.xy[1][0]])
                possible_antennas.append(numpy.vstack((x1.copy(), ant)))

            cov = 0
            best = None
            for ant in possible_antennas:
                _cov = combine_antennas(ant, radius)
                _cov = main_land.intersection(_cov).area
                if _cov > cov:
                    cov = _cov
```

```
                    best = ant
            x2 = best.copy()
            numpy.save(f'arrays/vcub1/x0_sqlq_{int(10*Eb_N0_min):03d}_{D1+1:02d
                }_antennas.npy', x2)

            initial_values = [x1, x2]
            # Defining complex optimization problem

            res = [minimize_antennas(initial_value, main_land, 0.95*radius) for
                 initial_value in initial_values]
            print(res)

            fig_full, axes_full = plt.subplots(ncols=2, figsize=(8, 4), sharey=
                True)
            fig_counter = 0
            for ax in axes_full:
                ax.tick_params(left=False, right=False, labelleft=False,
                    labelbottom=False, bottom=False)
                ax.set_aspect('equal')
            plt.subplots_adjust(wspace=0, hspace=0)
            for r, x0 in zip(res, initial_values):
                _x = x0.reshape(-1, 2)
                D = len(_x)
                filled_polygon = combine_antennas(_x, radius)
                initial_coverage = main_land.intersection(filled_polygon).area
                    / brazil_area
                fig, axes = plt.subplots()
                axes.plot(points_x, points_y, color='black')
                axes.plot(_x[:, 0], _x[:, 1], '+', color='black')
                axes.set_title(f'Coverage of {initial_coverage * 100:3.2f}\% 
                    with {D} antennas.')
                for antenna in _x:
                    plot_polygon(Point(antenna[0], antenna[1]).buffer(radius),
                        ax=axes, add_points=False, color=BLUE)
                # plot_polygon(filled_polygon, ax=axes, add_points=False, color
                    =BLUE)
                diff = main_land.difference(filled_polygon)
                try:
                    if isinstance(diff, Polygon):
                        plot_polygon(diff, ax=axes, add_points=False, color=RED
                            )
                    else:
                        for pol in diff.geoms:
                            plot_polygon(pol, ax=axes, add_points=False, color=
                                RED)
                except IndexError:
                    pass
                axes.set_xlabel('Degrees Longitude')
                axes.set_ylabel('Degrees Latitude')
                axes.set_xlim([long_min, long_max])
                axes.set_ylim([lat_min, lat_max])
                plt.show()
                fig.savefig(
                    f'final_results/vcub1/coverage_Eb_N0_{int(10*Eb_N0_min):03d
                        }_eps_{int(10*epsilon):03d}_minimalist_{D}
                        _antennas_boolean.pdf',
                    transparent=True, bbox_inches='tight')
                plt.close(fig)

```

```
                x = r.x
                x = x.reshape(-1, 2)
                D = len(x)
                filled_polygon = combine_antennas(x, radius)
                coverage = main_land.intersection(filled_polygon).area / brazil_area

                fig, axes = plt.subplots()
                axes.plot(points_x, points_y, color='black')
                axes.plot(x[:, 0], x[:, 1], '+', color='black')
                axes.set_aspect('equal')
                axes.set_title(f'Coverage of {coverage * 100:3.2f}\% with {D} antennas.')
                axes_full[fig_counter].plot(points_x, points_y, color='black')
                axes_full[fig_counter].plot(x[:, 0], x[:, 1], '+', color='black')
                axes_full[fig_counter].set_aspect('equal')
                axes_full[fig_counter].set_title(f'D = {D} antennas')
                for antenna in x:
                    plot_polygon(Point(antenna[0], antenna[1]).buffer(radius), ax=axes, add_points=False, color=BLUE)
                    plot_polygon(Point(antenna[0], antenna[1]).buffer(radius), ax=axes_full[fig_counter], add_points=False, color=BLUE)
                # plot_polygon(filled_polygon, ax=axes, add_points=False, color=BLUE)
                diff = main_land.difference(filled_polygon)
                try:
                    if isinstance(diff, Polygon):
                        plot_polygon(diff, ax=axes, add_points=False, color=RED)
                        plot_polygon(diff, ax=axes_full[fig_counter], add_points=False, color=RED)
                    else:
                        for pol in diff.geoms:
                            plot_polygon(pol, ax=axes, add_points=False, color=RED)
                            plot_polygon(pol, ax=axes_full[fig_counter], add_points=False, color=RED)
                except IndexError:
                    pass
                axes.set_xlabel('Degrees Longitude')
                axes.set_ylabel('Degrees Latitude')
                axes.set_xlim([long_min, long_max])
                axes.set_ylim([lat_min, lat_max])
                axes_full[fig_counter].set_xlim([long_min, long_max])
                axes_full[fig_counter].set_ylim([lat_min, lat_max])
                plt.show()

                numpy.save(f'arrays/vcub1/xbest_scipy_{int(10*Eb_N0_min):03d}_eps_{int(10*epsilon):03d}_{D}_antennas.npy', x)
                numpy.save(f'arrays/vcub1/radius_scipy_{int(10*Eb_N0_min):03d}_eps_{int(10*epsilon):03d}_{D}_antennas.npy', radius)
                fig.savefig(f'final_results/vcub1/coverage_Eb_N0_{int(10*Eb_N0_min):03d}_eps_{int(10*epsilon):03d}_scipy_{D}_antennas.pdf',
                            transparent=True, bbox_inches='tight')
                # plt.close(fig)
                all_results.append({
```

```
                    "$(E_b/N_0)_{\min}$": Eb_N0_min,
                    "Active Antennas": D,
                    "Initial Coverage": f"{initial_coverage * 100:3.2f}\%",
                    "Final Coverage": f"{coverage * 100:3.2f}\%"
                })
                axes_full[fig_counter].text(0.01, 0.11, f'Coverage:', transform=axes_full[fig_counter].transAxes,
                                            fontsize=font_size)
                axes_full[fig_counter].text(0.01, 0.01, f'{coverage * 100:3.2f}\%', transform=axes_full[fig_counter].transAxes,
                                            fontsize=font_size)
                fig_counter += 1
    df = pandas.DataFrame(all_results)
    print(df.to_latex(index=False))
    df.to_pickle('arrays/vcub1/scipy_summary.zip')
    fig_full.savefig(f'../parts/Ground Stations Distribution/vcub1/coverage_Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_scipy_combined.pdf',
                     transparent=True, bbox_inches='tight')
```

B.10 Differential Evolution

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
import shapely
from shapely.geometry import Point, Polygon, MultiPoint
from shapely.ops import unary_union
from shapely.plotting import plot_polygon
from figures import SIZE, BLACK, BLUE, GRAY, YELLOW, RED, set_limits
from shapely.ops import nearest_points
from itertools import combinations

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)

if __name__ == '__main__':
    Eb_N0_min = 7.5
    epsilon = 8
    threshold = 0.2
    f = 2244e6
    lamb = c0 / f
    brazil_points = pandas.read_pickle('input/brazil_points.zip')
    brazil_points_inside = brazil_points[brazil_points.inside].copy()
    main_land = brazil_mainland()
    main_land_contains = main_land.contains
    brazil_area = main_land.area
    long_min, lat_min, long_max, lat_max = main_land.bounds
    points_x = main_land.exterior.coords.xy[0]
    points_y = main_land.exterior.coords.xy[1]
    longs = numpy.unique(brazil_points['lon'].values)
```

```
    lats = numpy.unique(brazil_points['lat'].values)
    longs_inside = numpy.unique(brazil_points_inside['lon'].values)
    lats_inside = numpy.unique(brazil_points_inside['lat'].values)
    a = read_pickle(f'./input/a_matrix_{int(10*Eb_N0_min):03d}_binary_vcub1.pkl
        ')
    index = brazil_points_inside.iloc[100].name
    link_data = pandas.read_pickle(f'database/vcub1/antennas/link_data/{index}.
        zip')
    fixed_radius = link_data[link_data['Eb_N0'] > Eb_N0_min].sort_values('Eb_N0
        ').iloc[0].radius
    use_fixed_radius = False
    satellite = read_pickle('./input/satellite_vcub1.pkl')
    N = len(a)
    M, P = a[0].shape
    A = numpy.zeros((N, M * P), dtype=int)
    for k in range(N):
        A[k, :] = a[k].flatten()
    f_d = numpy.zeros(shape=(M, P))
    for _index, _row in brazil_points[brazil_points.inside].iterrows():
        coord = _row.coord
        try:
            long_idx = numpy.searchsorted(longs, coord[0], side="left")
            lat_idx = numpy.searchsorted(lats, coord[1], side="left")
            f_d[lat_idx, long_idx] = 1
        except IndexError:
            pass
    f_d = f_d.flatten()
    inside_mask = f_d == 1
    inside_total = len(f_d[inside_mask])

    # Differential Evolution
    NP = 100  # size of population
    CR = 0.9
    F_a = 0.05
    F_b = 0.1
    activate_antenna = 0.01
    deactivate_antenna = 0.5
    generations = 25
    possible_combinations = {
        7.5: [6]
    }
    all_results = []

    ground_parabola = Parabola(f=f, D=2.6 / lamb)
    station_parameters = {
        "f": f,
        "temp": 312,
        "eff": 0.5,
        "bandwidth": 6e6,
        "e_theta_e_phi_function": ground_parabola.get_fields_sym,
        "G_max": ground_parabola.G_max
    }
    link_parameters = {
        "satellite": satellite,
        "R_spec": 10e6,
        "Eb_N0_min": Eb_N0_min,
        "calc_transmitted_data": False,
        "G_other": -1.6
    }
```

```python

    def mutation(g):
        r1, r2, r3, r4 = numpy.random.default_rng().integers(low=0, high=NP, size=4)
        v_i_gp1 = X[g, r1] + F_b * (x_best - X[g, r2]) + F_a * (X[g, r3] - X[g, r4])
        return v_i_gp1


    def recombination(g, i, v_i_gp1):
        u_i_gp1 = numpy.zeros_like(v_i_gp1)
        for j in range(D):
            if ~main_land_contains(Point(v_i_gp1[j])):
                near = nearest_points(main_land, Point(v_i_gp1[j]))[0]
                # print(f"Snipping antenna {v_i_gp1[j]}")
                v_i_gp1[j] = numpy.array([near.coords.xy[0][0], near.coords.xy[1][0]])
            if main_land_contains(Point(v_i_gp1[j])) and (
                    numpy.random.uniform() <= CR or j == numpy.random.default_rng().integers(low=0, high=D, size=1)):
                u_i_gp1[j] = v_i_gp1[j]
            else:
                u_i_gp1[j] = X[g, i, j]
        return u_i_gp1


    def fit(x_i):
        radius = numpy.zeros(D)
        circles = []
        for _i in range(D):
            lon = x_i[_i][0]
            lat = x_i[_i][1]
            if use_fixed_radius:
                radius[_i] = fixed_radius
                circles.append(Point(lon, lat).buffer(radius[_i]))
            else:
                station = Station(lat=lat, lon=lon, **station_parameters)
                link = LinkBudget(station=station, **link_parameters)
                try:
                    circles.append(link.get_shapely_contour())
                except IndexError:
                    radius[_i] = 0
                    circles.append(Point(lon, lat).buffer(radius[_i]))
                    print("Could not get shapely contour")
        combined_circles = unary_union(circles)
        if main_land.intersects(combined_circles):
            try:
                intersection = shapely.intersection(main_land, combined_circles)
            except RuntimeWarning:
                intersection = shapely.intersection(combined_circles, main_land)
        else:
            return 1, 0, 0
        coverage_area = intersection.area
        coverage = coverage_area / brazil_area
        intersect = coverage_area / best_coverage_area
        return 1 - coverage, coverage, intersect

```

```

    for Eb_N0_min in possible_combinations.keys():
        for D in possible_combinations[Eb_N0_min]:
            link_parameters["Eb_N0_min"] = Eb_N0_min
            x0_sufix_results = f"{D:02d}_antennas_Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_CR_{int(10 * CR):02d}_F_a_{int(100 * F_a):03d}_F_b_{int(100 * F_b):03d}_fixed_radius_True"
            sufix = f"Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_{D}_antennas_boolean_fixed_radius_{use_fixed_radius}"
            print("\n=======================================================\n")
            print(f"Eb_N0 = {Eb_N0_min}, epsilon = {epsilon}, D = {D}, F_a = {F_a}, F_b = {F_b}, CR = {CR}")
            print("\n=======================================================\n")
            fixed_radius = numpy.load(
                f'arrays/vcub1/radius_scipy_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_{D}_antennas.npy')
            if use_fixed_radius:
                x0 = numpy.load(
                    f'arrays/vcub1/xbest_scipy_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_{D}_antennas.npy')
            else:
                x0 = numpy.load(
                    f'arrays/vcub1/best_result_{x0_sufix_results}.npy')
            best_coverage_area = D * numpy.pi * fixed_radius ** 2
            fit_before_snip, coverage_before_snip, intersect_before_snip = fit(x0)
            for i in range(len(x0)):
                ant = x0[i]
                if ~main_land_contains(Point(ant)):
                    near = nearest_points(main_land, Point(ant))[0]
                    x0[i] = numpy.array([near.coords.xy[0][0], near.coords.xy[1][0]])
            fit_after_snip, coverage_after_snip, intersect_after_snip = fit(x0)
            X = numpy.zeros(shape=(generations, NP, D, 2))
            if use_fixed_radius:
                for i in range(NP):
                    X[0, i] = brazil_points_inside.loc[numpy.random.choice(brazil_points_inside.index.values, D, replace=False)][
                        ['lon', 'lat']].values
                combs = combinations(x0, 2)

                possible_antennas = []
                for comb in combs:
                    ant = (comb[0] + comb[1]) / 2
                    if ~main_land_contains(Point(ant)):
                        near = nearest_points(main_land, Point(ant))[0]
                        ant = numpy.array([near.coords.xy[0][0], near.coords.xy[1][0]])
                    for _k in range(len(x0)):
                        new_ant = x0.copy()
                        new_ant[_k] = ant
                        possible_antennas.append(new_ant)

                for _k in range(min(len(possible_antennas), NP - 1)):
                    X[0, _k + 1] = possible_antennas[_k]
            else:
                X[0, :] = numpy.load(
                    f'arrays/vcub1/best_result_{x0_sufix_results}.npy')
                for i in range(1, 50):
```

```
        X[0, i] = \
            brazil_points_inside.loc[
                numpy.random.choice(brazil_points_inside.index.values, D, replace=False)][
                ['lon', 'lat']].values
X[0, 0] = x0
n0 = 5
# Plot snipped antennas
circles = []
radius = numpy.zeros(D)
for _i in range(D):
    lon = x0[_i][0]
    lat = x0[_i][1]
    antenna_point = Point(lon, lat)
    if use_fixed_radius:
        radius[_i] = fixed_radius
        circles.append(Point(lon, lat).buffer(radius[_i]))
    else:
        station = Station(lat=lat, lon=lon, **station_parameters)
        link = LinkBudget(station=station, **link_parameters)
        circles.append(link.get_shapely_contour())
combined_circles = unary_union(circles)
intersection = shapely.intersection(main_land, combined_circles)
coverage = intersection.area / brazil_area
fig, axes = plt.subplots()
axes.plot(points_x, points_y, color='black')
axes.plot(x0[:, 0], x0[:, 1], '+', color='black')
axes.set_aspect('equal')
axes.set_title(f'Coverage of {coverage * 100:3.2f}\% with {D} antennas.')
for i in range(D):
    plot_polygon(circles[i], ax=axes, add_points=False, color=BLUE)
diff = main_land.difference(combined_circles)
try:
    if isinstance(diff, Polygon):
        plot_polygon(diff, ax=axes, add_points=False, color=RED)
    else:
        for pol in diff.geoms:
            plot_polygon(pol, ax=axes, add_points=False, color=RED)
except IndexError:
    pass
axes.set_xlabel('Degrees Longitude')
axes.set_ylabel('Degrees Latitude')
axes.set_xlim([long_min, long_max])
axes.set_ylim([lat_min, lat_max])
fig.savefig(
    f'final_results/vcub1/coverage_snipped_antennas_{sufix}.pdf',
    transparent=True, bbox_inches='tight')
plt.close(fig)

# DE algorithm
x_best = x0
best_fit, best_coverage, best_intersect = fit(x_best)
bests = [x_best]
bests_fits = [best_fit]
fit_X = numpy.zeros(NP)
coverage_X = numpy.zeros(NP)
intersect_X = numpy.zeros(NP)
print(f'Initial fit: {100 * best_fit:1.5f}')
```

```
        tol = 1e-7
        for g in range(generations - 1):
            for i in range(NP):
                if g == 0:
                    fit_X[i], coverage_X[i], intersect_X[i] = fit(X[g, i])
                v_i_gp1 = mutation(g)
                u_i_gp1 = recombination(g, i, v_i_gp1)
                fit_u, coverage_u, intersect_u = fit(u_i_gp1)
                if fit_u < fit_X[i]:
                    print(f'Generation {g}, iteration {i}'
                          f'\t Fit x: {100*fit_X[i]:2.5f}'
                          f'\t Fit u: {100*fit_u:2.5f}')
                    X[g + 1, i] = u_i_gp1.copy()
                    fit_X[i] = fit_u
                    coverage_X[i] = coverage_u
                    intersect_X[i] = intersect_u
                    if fit_u < best_fit:
                        best_fit = fit_u
                        best_coverage = coverage_u
                        best_intersect = intersect_u
                        x_best = u_i_gp1.copy()
                        bests.append(x_best)
                        bests_fits.append(best_fit)
                        print(f'Generation {g:02d}, iteration {i:03d}.\n'
                              f'\t Found better fit: {100 * best_fit:1.5f}')
                else:
                    X[g + 1, i] = X[g, i].copy()
            print(f"Best of generation {g:02d}: {100 * best_fit:1.5f}\n"
                  f"=============================================")
            if best_coverage >= 1 - 1e-6:
                break

        sufix_results = f"{D:02d}_antennas_Eb_N0_{int(10 * Eb_N0_min):03d}_eps_{int(10 * epsilon):03d}_CR_{int(10*CR):02d}_F_a_{int(100*F_a):03d}_F_b_{int(100*F_b):03d}_fixed_radius_{use_fixed_radius}"
        numpy.save(
            f'arrays/vcub1/best_generation_{sufix_results}.npy',
            X[g])
        numpy.save(
            f'arrays/vcub1/xbests_{sufix_results}.npy',
            numpy.array(bests))
        numpy.save(
            f'arrays/vcub1/best_result_{sufix_results}.npy',
            x_best)

        circles = []
        radius = numpy.zeros(D)
        for _i in range(D):
            lon = x_best[_i][0]
            lat = x_best[_i][1]
            antenna_point = Point(lon, lat)
            if use_fixed_radius:
                radius[_i] = fixed_radius
                circles.append(Point(lon, lat).buffer(radius[_i]))
            else:
                station = Station(lat=lat, lon=lon, **station_parameters)
                link = LinkBudget(station=station, **link_parameters)
```

```
                    circles.append(link.get_shapely_contour())
            combined_circles = unary_union(circles)
            intersection = shapely.intersection(main_land, combined_circles)
            coverage = intersection.area / brazil_area

            fig, axes = plt.subplots()
            axes.plot(points_x, points_y, color='black')
            axes.plot(x_best[:, 0], x_best[:, 1], '+', color='black')
            axes.set_aspect('equal')
            axes.set_title(f'Coverage of {coverage * 100:3.2f}\% with {D} antennas.')
            for i in range(D):
                plot_polygon(circles[i], ax=axes, add_points=False, color=BLUE)
            diff = main_land.difference(combined_circles)
            try:
                if isinstance(diff, Polygon):
                    plot_polygon(diff, ax=axes, add_points=False, color=RED)
                else:
                    for pol in diff.geoms:
                        plot_polygon(pol, ax=axes, add_points=False, color=RED)
            except IndexError:
                pass
            axes.set_xlabel('Degrees Longitude')
            axes.set_ylabel('Degrees Latitude')
            axes.set_xlim([long_min, long_max])
            axes.set_ylim([lat_min, lat_max])
            fig.savefig(
                f'final_results/vcub1/coverage_{sufix_results}_after_{g + 1:03d}_iterations.pdf',
                transparent=True, bbox_inches='tight')
            plt.close(fig)

            all_results.append({
                "$\\nicefrac{E_b}{N_0}_{\min}$": Eb_N0_min,
                "Active Antennas": D,
                "Coverage Before Snip": f"{coverage_before_snip * 100:3.2f}",
                "Coverage After Snip": f"{coverage_after_snip * 100:3.2f}",
                "Best Coverage": f"{best_coverage * 100:3.2f}",
                "Iterations": f"{g}"
            })

    df = pandas.DataFrame(all_results)
    df.to_pickle(f'final_results/vcub1/de_result_{sufix_results}.zip')
    print(df.to_latex(index=False))
```

B.11 Fixed Positions

```
from arraytools import *
import numpy
import pandas
from matplotlib import use, rc
from shapely.geometry import Point
import shapely
import matplotlib.pyplot as plt
from shapely.geometry import LineString, LinearRing, Polygon
from shapely.plotting import plot_line, plot_points, plot_polygon
```

```python
from figures import SIZE, BLACK, BLUE, GRAY, YELLOW, RED, set_limits
from shapely.ops import unary_union
from scipy.optimize import minimize
from shapely.prepared import prep
from itertools import combinations
rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
font_size = 15
plt.rcParams.update({'font.size': font_size})
pandas.set_option('expand_frame_repr', False)


def get_antenna_coverage_and_intersection(_antennas, _diameter):
    individual_coverages = []
    intersections = []
    for n in range(len(_antennas)):
        _ground_parabola = Parabola(f=f, D=_diameter[n] / lamb, eff=0.5)
        _station = Station(G_max=_ground_parabola.G_max, lon=_antennas[n][0], lat=_antennas[n][1],
                           e_theta_e_phi_function=_ground_parabola.get_fields_sym, **station_parameters)
        _link = LinkBudget(station=_station, **link_parameters)
        if _link.passes == {}:
            individual_coverages.append(shapely.Point(_station.lon, _station.lat).buffer(0))
        else:
            individual_coverages.append(_link.get_shapely_contour())
    for comb in combinations(individual_coverages, 2):
        if comb[0].overlaps(comb[1]):
            intersections.append(comb[0].intersection(comb[1]))
    return individual_coverages, intersections


def combine_antennas(_antennas, _radius):
    return unary_union([Point(_antennas[n][0], _antennas[n][1]).buffer(_radius[n]) for n in range(len(_antennas))])


def combine_antenna_xy(_x, _y, _radius):
    return unary_union([Point(point[0], point[1]).buffer(_radius) for point in tuple(zip(_x.value, _y.value))])


def get_antennas_intersection(_antennas, _radius):
    intersections = []
    individual_coverages = [Point(_antennas[n][0], _antennas[n][1]).buffer(_radius[n]) for n in range(len(_antennas))]
    for comb in combinations(individual_coverages, 2):
        if comb[0].overlaps(comb[1]):
            intersections.append(comb[0].intersection(comb[1]))
    return unary_union(intersections)


def minimize_antennas_with_real_coverage(region_boundary, x0, d_min, d_max, desired_coverage=0.99999, algorithm="SLSQP"):
    N = len(positions)
    brazil_area = region_boundary.area
```

```python
    maximum_coverage, maximum_intersection = get_antenna_coverage_and_intersection(positions, d_max*numpy.ones(N))
    maximum_coverage = unary_union(maximum_coverage)
    maximum_coverage_area = maximum_coverage.area
    max_iterations = 10000

    def func_to_minimize(x):
        coverage, intersections = get_antenna_coverage_and_intersection(positions, x)
        coverage = unary_union(coverage)
        intersections = unary_union(intersections)
        return 0.4*(maximum_coverage_area - coverage.area) + 0.6*intersections.area

    def constraint_function(x):
        coverage, intersections = get_antenna_coverage_and_intersection(positions, x)
        coverage = unary_union(coverage)
        return region_boundary.intersection(coverage).area / brazil_area - desired_coverage

    constraints = [
        {
            'type': 'ineq',
            'fun': constraint_function
        }]

    bounds = N * [[d_min, d_max]]
    res = minimize(func_to_minimize, x0, method=algorithm, constraints=constraints, bounds=bounds,
                   options={'maxiter': max_iterations,
                            'ftol': 5e-3,
                            'eps': 1e-6})
    print("Final constraint: ", constraint_function(res.x))
    return res


def minimize_antennas(region_boundary, x0, r_min, r_max, desired_coverage=0.99999):
    N = len(positions)
    brazil_area = region_boundary.area
    region_prep = prep(region_boundary)
    maximum_coverage = combine_antennas(positions, r_max*numpy.ones(N))
    maximum_coverage_area = maximum_coverage.area
    max_iterations = 10000

    def func_to_minimize(x):
        # best_coverage_area = numpy.sum(N * numpy.pi * x ** 2)
        coverage = combine_antennas(positions, x)
        intersections = get_antennas_intersection(positions, x)
        return 0.5*(maximum_coverage_area - coverage.area) + 0.5*intersections.area

    def constraint_function(x):
        coverage = combine_antennas(positions, x)
        return region_boundary.intersection(coverage).area / brazil_area - desired_coverage

    constraints = [
```

```
        {
            'type': 'ineq',
            'fun': constraint_function
        }]

    bounds = N * [[r_min, r_max]]
    res = minimize(func_to_minimize, x0, method='SLSQP', constraints=constraints, bounds=bounds,
                   options={'maxiter': max_iterations,
                            'ftol': 5e-3,
                            'eps': 1e-6})
    print("Final constraint: ", constraint_function(res.x))
    return res


if __name__ == '__main__':
    main_land = brazil_mainland()
    south_america = get_south_america()
    # brazil = numpy.load('./input/brazil.npy')
    # main_land = Polygon(brazil)
    brazil_area = main_land.area
    brazil_points = pandas.read_pickle('input/brazil_points.zip')
    brazil_points_inside = brazil_points[brazil_points.inside]
    longs = numpy.unique(brazil_points['lon'].values)
    lats = numpy.unique(brazil_points['lat'].values)
    longs_inside = numpy.unique(brazil_points_inside['lon'].values)
    lats_inside = numpy.unique(brazil_points_inside['lat'].values)
    points_x = main_land.exterior.coords.xy[0]
    points_y = main_land.exterior.coords.xy[1]
    long_min, lat_min, long_max, lat_max = main_land.bounds
    # plot_long_min, plot_lat_min, plot_long_max, plot_lat_max = south_america.bounds
    plot_long_min, plot_lat_min, plot_long_max, plot_lat_max = -76, -43, -25, 16
    sat = read_pickle('./input/satellite_vcub1.pkl')
    f = 2244e6
    lamb = c0 / f
    Eb_N0_min = 7.5
    diameter_min = 1.5
    diameter_max = 6
    station_parameters = {'f': f, 'eff': 0.5, 'temp': 312, 'bandwidth': 6e6}
    link_parameters = {'satellite': sat, 'R_spec': 10e6, 'Eb_N0_min': Eb_N0_min, 'calc_transmitted_data': True, 'G_other':-1.6}

    antenna_ranges = []
    diameters = numpy.linspace(diameter_min, diameter_max, 20)
    for diam in diameters:
        ground_parabola = Parabola(f=f, D=diam / lamb, eff=0.5)
        station = Station(G_max=ground_parabola.G_max, lat=antenna_positions['unb']['lat'], lon=antenna_positions['unb']['lon'],
                          e_theta_e_phi_function=ground_parabola.get_fields_sym, **station_parameters)
        link = LinkBudget(station=station, **link_parameters)
        if link.passes == {}:
            antenna_reach = 0
        else:
            coords = tuple(zip(link.data['lon_WGS84_deg'].values, link.data['lat_WGS84_deg'].values))
            points = [Point(coord) for coord in coords]
```

```
            link.data['radius'] = shapely.distance(points, Point(antenna_positions['unb']['lon'], antenna_positions['unb']['lat']))
            antenna_reach = link.data[link.data['Eb_N0'] >= Eb_N0_min].sort_values('Eb_N0').iloc[0].radius
            antenna_ranges.append(antenna_reach)
    antenna_ranges = numpy.array(antenna_ranges)

    N = len(positions)
    x0 = numpy.ones(N)
    r_tol = 1e-5
    r_max = numpy.max(antenna_ranges)
    initial_res = minimize_antennas(main_land, x0, r_tol, r_max)

    initial_diameters = numpy.interp(initial_res.x, antenna_ranges, diameters)
    initial_diameters[initial_diameters <= 1.05*diameter_min] = 0
    print(initial_res)
    radius = initial_res.x
    # used_antennas = ~numpy.isclose(r_tol, res.x)
    coverages, intersection = get_antenna_coverage_and_intersection(positions, initial_diameters)
    used_antennas = []
    for n in range(len(positions)):
        individual_cov = coverages[n]
        if unary_union(coverages[:n] + coverages[n+1:]).intersection(individual_cov).area <= 0.8 * individual_cov.area:
            used_antennas.append(True)
        else:
            used_antennas.append(False)

    # used_antennas = initial_res.x > r_min_on
    # used_antennas = numpy.ones_like(radius, dtype=bool)
    D = numpy.count_nonzero(used_antennas)
    filled_polygon = combine_antennas(positions[used_antennas], radius[used_antennas])
    coverage = main_land.intersection(filled_polygon).area / brazil_area
    fig, axes = plt.subplots(figsize=(6, 6))
    axes.plot(south_america.exterior.coords.xy[0], south_america.exterior.coords.xy[1], color='lightgrey')
    axes.plot(points_x, points_y, color='black')
    axes.plot(positions[used_antennas][:, 0], positions[used_antennas][:, 1], '+', color='black')
    axes.set_aspect('equal')
    axes.set_title(f'Coverage of {coverage * 100:3.2f}\% with {D} antennas.', fontsize=font_size - 1)
    axes.tick_params(left=False, right=False, labelleft=False, labelbottom=False, bottom=False)
    print("Lat, Lon, Diameter, Range")
    n = 0
    for antenna, r, diam in zip(positions, radius, initial_diameters):
        print(f"{antenna[0]:0.2f} & {antenna[1]:0.2f} & {diam:0.2f} & {r:0.2f}")
        if used_antennas[n]:
            plot_polygon(Point(antenna[0], antenna[1]).buffer(r), ax=axes, add_points=False, color=BLUE)
        n += 1
    diff = main_land.difference(filled_polygon)
    try:
        if isinstance(diff, Polygon):
```

```
            plot_polygon(diff, ax=axes, add_points=False, color=RED)
        else:
            for pol in diff.geoms:
                plot_polygon(pol, ax=axes, add_points=False, color=RED)
    except IndexError:
        pass
    axes.set_xlim([plot_long_min, plot_long_max])
    axes.set_ylim([plot_lat_min, plot_lat_max])
    fig.set_tight_layout(True)
    sufix = f"d_min_{int(10*diameter_min):02d}_d_max_{int(10*diameter_max):02d}"
    fig.savefig(f'../parts/Ground Stations Distribution/vcub1/varying_gain_results_fixed_coverage_{sufix}.pdf',
        transparent=True, bbox_inches='tight')
    full_stations_vector = numpy.hstack([positions, initial_diameters.reshape(-1, 1)])
    numpy.save(f"arrays/vcub1/full_stations_vector_varying_diameter_fixed_coverage_{sufix}.npy", full_stations_vector)
    stations_vector = numpy.hstack([positions[used_antennas], initial_diameters[used_antennas].reshape(-1, 1)])
    numpy.save(f"arrays/vcub1/stations_vector_varying_diameter_fixed_coverage_{sufix}.npy", stations_vector)
    plt.show()

    final_res = minimize_antennas_with_real_coverage(region_boundary=main_land, x0=initial_diameters, d_min=r_tol, d_max=diameter_max, algorithm="L-BFGS-B")
    print(final_res)
    final_diameters = final_res.x

    individual_patterns, individual_intersections = get_antenna_coverage_and_intersection(positions, final_diameters)
    used_antennas = []
    for n in range(len(positions)):
        individual_cov = individual_patterns[n]
        if individual_cov.area < 1 or unary_union(individual_patterns[:n] + individual_patterns[n+1:]).intersection(individual_cov).area > 0.8 * individual_cov.area:
            used_antennas.append(False)
        else:
            used_antennas.append(True)

    D = numpy.count_nonzero(used_antennas)
    filled_polygon = unary_union(individual_patterns)
    coverage = main_land.intersection(filled_polygon).area / brazil_area

    fig, axes = plt.subplots(figsize=(6, 6))
    axes.plot(south_america.exterior.coords.xy[0], south_america.exterior.coords.xy[1], color='lightgrey')
    axes.plot(points_x, points_y, color='black')
    axes.plot(positions[used_antennas][:, 0], positions[used_antennas][:, 1], '+', color='black')
    axes.set_aspect('equal')
    axes.set_title(f'Coverage of {coverage * 100:3.2f}\% with {D} antennas.', fontsize=font_size-1)
    axes.tick_params(left=False, right=False, labelleft=False, labelbottom=False, bottom=False)
    print("Lon, Lat, Diameter")
```

```
    for n in range(len(positions)):
        print(f"{positions[n][0]:0.2f} & {positions[n][1]:0.2f} & {final_diameters[n]:0.2f}")
        if used_antennas[n]:
            plot_polygon(individual_patterns[n], ax=axes, add_points=False, color=BLUE)
    diff = main_land.difference(filled_polygon)
    try:
        if isinstance(diff, Polygon):
            plot_polygon(diff, ax=axes, add_points=False, color=RED)
        else:
            for pol in diff.geoms:
                plot_polygon(pol, ax=axes, add_points=False, color=RED)
    except IndexError:
        pass
    axes.set_xlim([plot_long_min, plot_long_max])
    axes.set_ylim([plot_lat_min, plot_lat_max])
    axes.set_xlim([-83, -20])
    axes.set_ylim([-55, 23])
    fig.set_tight_layout(True)
    fig.savefig(f'../parts/Ground Stations Distribution/vcub1/varying_gain_results_real_coverage_{sufix}.pdf',
        transparent=True, bbox_inches='tight')
    full_stations_vector = numpy.hstack([positions, final_diameters.reshape(-1, 1)])
    numpy.save(f"arrays/vcub1/full_stations_vector_varying_diameter_real_coverage_{sufix}.npy", full_stations_vector)
    stations_vector = numpy.hstack([positions[used_antennas], final_diameters[used_antennas].reshape(-1, 1)])
    numpy.save(f"arrays/vcub1/stations_vector_varying_diameter_real_coverage_{sufix}.npy", stations_vector)
    plt.show()
```

B.12 MicroStrip Array

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', 500)

if __name__ == '__main__':
    # This method of project is following Balanis Chapter 14.2.C
    f = 10e9
    lamb = c0 / f
    eps_r = 2.2
    h = 0.1588e-2
    parameters = {
```

```
        "h": h,
        "eps_r": eps_r,
        "f": f
    }
    strip = MicroStrip(**parameters, hfss_path='hfss_data/E.csv')
    N = 3
    # Distance must be greater than 2*L
    distance = 0.5 * lamb
    d = distance * numpy.arange(-(N - 1) / 2, (N / 1) / 2, 1)
    a = 1 / N * numpy.ones(N)

    antenna_pos = numpy.vstack([d,
                                numpy.zeros_like(d),
                                numpy.zeros_like(d)]).T
    beta = deg2rad(45)

    step_theta_hfss = 1
    step_phi_hfss = 1
    theta_hfss = numpy.arange(-180, 180 + step_theta_hfss, step_theta_hfss)
    phi_hfss = numpy.arange(-180, 180 + step_phi_hfss, step_phi_hfss)
    phi_hfss[phi_hfss < 0] += 360
    theta_grid = deg2rad(theta_hfss)
    phi_grid = deg2rad(phi_hfss)
    n_theta_hfss = len(theta_hfss)
    n_phi_hfss = len(phi_hfss)

    Theta, Phi = numpy.meshgrid(theta_grid, phi_grid, indexing='ij')

    hfss_e_db_steered = pandas.read_csv('hfss_data/E_db_beta_45_full_theta.csv'
        , skiprows=1, header=None, index_col=0)
    hfss_e_tot_db_steered, hfss_e_phi_db_steered, hfss_e_theta_db_steered =
        parse_hfss_data(hfss_e_db_steered, phi_hfss)

    hfss_e_db_array = pandas.read_csv('hfss_data/E_db_array_full_theta.csv',
        skiprows=1, header=None, index_col=0)
    hfss_e_tot_db_array, hfss_e_phi_db_array, hfss_e_theta_db_array =
        parse_hfss_data(hfss_e_db_array, phi_hfss)

    hfss_e_db_array_steered = pandas.read_csv('hfss_data/
        E_db_array_beta_45_full_theta.csv', skiprows=1, header=None,
                                              index_col=0)
    hfss_e_tot_db_array_steered, hfss_e_phi_db_array_steered,
        hfss_e_theta_db_array_steered = parse_hfss_data(
        hfss_e_db_array_steered, phi_hfss)

    e_plane_phi = 0
    h_plane_phi = 90
    y_min = -30

    betas = 0 * numpy.ones(N)
    # Find max_value
    e_theta_steered_array, e_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=Theta.flatten(), lamb=lamb,
        phi=Phi.flatten(),
        betas=betas, fields_func=strip.e_analytical)
    e_tot_steered_array = numpy.sqrt(numpy.abs(e_theta_steered_array) ** 2 +
        numpy.abs(e_phi_steered_array) ** 2)
    array_rotated_max = numpy.max(e_tot_steered_array)
```

```python

    e_theta_nominal_array, e_phi_nominal_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(e_plane_phi)),
        betas=betas, fields_func=strip.e_analytical)
    h_theta_nominal_array, h_phi_nominal_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(h_plane_phi)),
        betas=betas, fields_func=strip.e_analytical)
    e_tot_nominal_array = numpy.sqrt(numpy.abs(e_theta_nominal_array) ** 2 + numpy.abs(e_phi_nominal_array) ** 2)
    h_tot_nominal_array = numpy.sqrt(numpy.abs(h_theta_nominal_array) ** 2 + numpy.abs(h_phi_nominal_array) ** 2)
    for _array in [e_theta_nominal_array, e_phi_nominal_array, h_theta_nominal_array, h_phi_nominal_array,
                   e_tot_nominal_array, h_tot_nominal_array]:
        _array = _array / array_rotated_max
    fig_size = 3
    y_min = -30
    fig, axes = plt.subplots(nrows=3, ncols=2, sharex=True, sharey=True, figsize=(3 * fig_size, 2 * fig_size))
    axes[0][0].set_title(f"E-Plane ($\phi = 0^\circ$)")
    axes[0][1].set_title(f"H-Plane ($\phi = 90^\circ$)")
    fig.suptitle(f"Analytical model array vs HFSS array")
    axes[0][0].set_ylabel(r"$E_\theta [^\circ]$ [dB]")
    axes[1][0].set_ylabel(r"$E_\phi$ [dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$ [dB]")
    axes[2][0].set_xlabel(r"$\theta [^\circ]$")
    axes[2][1].set_xlabel(r"$\theta [^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_array[e_plane_phi])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_array[e_plane_phi])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_array[e_plane_phi])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_nominal_array), db=20))
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_nominal_array), db=20))
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_nominal_array), db=20))

    axes[0][1].plot(theta_hfss, hfss_e_theta_db_array[h_plane_phi], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_array[h_plane_phi])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_array[h_plane_phi])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_nominal_array), db=20), label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_nominal_array), db=20))
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_nominal_array), db=20))
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 0.1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Antenna Array/hfss_vs_analytical_array.pdf',
```

```
        transparent=True, bbox_inches='tight', pad_inches=0)

    # Array rotated
    # Find max_value
    betas = -beta * numpy.ones(N)
    e_theta_steered_array, e_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=Theta.flatten(), lamb=lamb,
        phi=Phi.flatten(),
        betas=betas, fields_func=strip.e_analytical)
    e_tot_steered_array = numpy.sqrt(numpy.abs(e_theta_steered_array) ** 2 +
       numpy.abs(e_phi_steered_array) ** 2)
    array_rotated_max = numpy.max(e_tot_steered_array)

    e_theta_steered_array, e_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid,
                            deg2rad(e_plane_phi)),
        betas=betas, fields_func=strip.e_analytical)

    e_tot_steered_array = numpy.sqrt(numpy.abs(e_theta_steered_array) ** 2 +
       numpy.abs(e_phi_steered_array) ** 2)
    e_theta_steered_array = e_theta_steered_array / array_rotated_max
    e_phi_steered_array = e_phi_steered_array / array_rotated_max
    e_tot_steered_array = e_tot_steered_array / array_rotated_max

    h_theta_steered_array, h_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid,
                            deg2rad(h_plane_phi)),
        betas=betas, fields_func=strip.e_analytical)

    h_tot_steered_array = numpy.sqrt(numpy.abs(h_theta_steered_array) ** 2 +
       numpy.abs(h_phi_steered_array) ** 2)
    h_theta_steered_array = h_theta_steered_array / array_rotated_max
    h_phi_steered_array = h_phi_steered_array / array_rotated_max
    h_tot_steered_array = h_tot_steered_array / array_rotated_max

    y_min = -30
    fig, axes = plt.subplots(nrows=3, ncols=2, sharex='col', sharey=True,
       figsize=(3 * fig_size, 2 * fig_size))
    axes[0][0].set_title(f"E-Plane ($\phi = 0^\circ$)")
    axes[0][1].set_title(f"H-Plane ($\phi = 90^\circ$)")
    fig.suptitle(f"Analytical model vs HFSS - Steered $\\beta = {rad2deg(beta)
       :0.0f}^\circ$")
    axes[0][0].set_ylabel(r"$E_\theta [^\circ]$ [dB]")
    axes[1][0].set_ylabel(r"$E_\phi$ [dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$ [dB]")
    axes[2][0].set_xlabel(r"$\theta [^\circ]$")
    axes[2][1].set_xlabel(r"$\theta [^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_array_steered[e_plane_phi])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_array_steered[e_plane_phi])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_array_steered[e_plane_phi])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_steered_array), db=20))
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_steered_array), db=20))
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_steered_array), db=20))
```

```
    axes[0][1].plot(theta_hfss, hfss_e_theta_db_array_steered[h_plane_phi],
        label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_array_steered[h_plane_phi])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_array_steered[h_plane_phi])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_steered_array), db=20),
         label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_steered_array), db=20))
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_steered_array), db=20))
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 0.1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    for ax in axes[:, 0]:
        ax.set_xlim([-90 + rad2deg(beta), 90 + rad2deg(beta)])
        ax.set_xticks(numpy.arange(-90 + rad2deg(beta), 91 + rad2deg(beta),
            step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Antenna Array/hfss_vs_analytical_array_steered.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)

    # HFSS combined array
    betas = 0 * numpy.ones(N)
    e_theta_hfss_combined_array, e_phi_hfss_combined_array =
        combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(e_plane_phi)),
        betas=betas,
        fields_func=strip.e_hfss)
    h_theta_hfss_combined_array, h_phi_hfss_combined_array =
        combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(h_plane_phi)),
        betas=betas,
        fields_func=strip.e_hfss)
    e_tot_hfss_combined_array = numpy.sqrt(
        numpy.abs(e_theta_hfss_combined_array) ** 2 + numpy.abs(
            e_phi_hfss_combined_array) ** 2)
    h_tot_hfss_combined_array = numpy.sqrt(
        numpy.abs(h_theta_hfss_combined_array) ** 2 + numpy.abs(
            h_phi_hfss_combined_array) ** 2)

    fig, axes = plt.subplots(nrows=3, ncols=2, sharex=True, sharey=True,
        figsize=(3 * fig_size, 2 * fig_size))
    axes[0][0].set_title(f"E-Plane ($\phi = 0^\circ$)")
    axes[0][1].set_title(f"H-Plane ($\phi = 90^\circ$)")
    fig.suptitle(f"Analytically Combined Array vs HFSS array")
    axes[0][0].set_ylabel(r"$E_\theta [^\circ]$ [dB]")
    axes[1][0].set_ylabel(r"$E_\phi$ [dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$ [dB]")
    axes[2][0].set_xlabel(r"$\theta [^\circ]$")
```

```
    axes[2][1].set_xlabel(r"$\theta [^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_array[e_plane_phi].loc[theta_hfss])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_array[e_plane_phi].loc[theta_hfss])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_array[e_plane_phi].loc[theta_hfss])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_hfss_combined_array), db=20), '.')
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_hfss_combined_array), db=20), '.')
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_hfss_combined_array), db=20), '.')

    axes[0][1].plot(theta_hfss, hfss_e_theta_db_array[h_plane_phi].loc[theta_hfss], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_array[h_plane_phi].loc[theta_hfss])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_array[h_plane_phi].loc[theta_hfss])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_hfss_combined_array), db=20), '.', label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_hfss_combined_array), db=20), '.')
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_hfss_combined_array), db=20), '.')
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 0.1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Theoretical Foundation/hfss_analytically_array_vs_hfss_array.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)

    # HFSS combined and rotated array
    betas = -beta * numpy.ones(N)
    e_theta_steered_array, e_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=Theta.flatten(), lamb=lamb,
        phi=Phi.flatten(),
        betas=betas, fields_func=strip.e_hfss)
    e_tot_steered_array = numpy.sqrt(numpy.abs(e_theta_steered_array) ** 2 + numpy.abs(e_phi_steered_array) ** 2)
    array_rotated_max = numpy.max(e_tot_steered_array)

    e_theta_hfss_combined_array, e_phi_hfss_combined_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(e_plane_phi)),
        betas=betas,
```

```
        fields_func=strip.e_hfss)

    h_theta_hfss_combined_array, h_phi_hfss_combined_array = combining_general_array(
        antenna_pos=antenna_pos, a=a,
        theta=theta_grid, lamb=lamb,
        phi=numpy.full_like(theta_grid, deg2rad(h_plane_phi)),
        betas=betas,
        fields_func=strip.e_hfss)
    e_tot_hfss_combined_array = numpy.sqrt(
        numpy.abs(e_theta_hfss_combined_array) ** 2 + numpy.abs(e_phi_hfss_combined_array) ** 2)
    h_tot_hfss_combined_array = numpy.sqrt(
        numpy.abs(h_theta_hfss_combined_array) ** 2 + numpy.abs(h_phi_hfss_combined_array) ** 2)

    fig, axes = plt.subplots(nrows=3, ncols=2, sharex='col', sharey=True, figsize=(3 * fig_size, 2 * fig_size))
    axes[0][0].set_title(f"E-Plane ($\phi = 0^\circ$)")
    axes[0][1].set_title(f"H-Plane ($\phi = 90^\circ$)")
    fig.suptitle(f"Analytically Rotated and Combined Array vs HFSS - Steered $\\beta = {rad2deg(beta):0.0f}^\circ$")
    axes[0][0].set_ylabel(r"$E_\theta [^\circ]$ [dB]")
    axes[1][0].set_ylabel(r"$E_\phi$ [dB]")
    axes[2][0].set_ylabel(r"$E_{tot}$ [dB]")
    axes[2][0].set_xlabel(r"$\theta [^\circ]$")
    axes[2][1].set_xlabel(r"$\theta [^\circ]$")

    axes[0][0].plot(theta_hfss, hfss_e_theta_db_array_steered[e_plane_phi].loc[theta_hfss])
    axes[1][0].plot(theta_hfss, hfss_e_phi_db_array_steered[e_plane_phi].loc[theta_hfss])
    axes[2][0].plot(theta_hfss, hfss_e_tot_db_array_steered[e_plane_phi].loc[theta_hfss])
    axes[0][0].plot(theta_hfss, to_db(numpy.abs(e_theta_hfss_combined_array) / array_rotated_max, db=20), '.')
    axes[1][0].plot(theta_hfss, to_db(numpy.abs(e_phi_hfss_combined_array) / array_rotated_max, db=20), '.')
    axes[2][0].plot(theta_hfss, to_db(numpy.abs(e_tot_hfss_combined_array) / array_rotated_max, db=20), '.')

    axes[0][1].plot(theta_hfss, hfss_e_theta_db_array_steered[h_plane_phi].loc[theta_hfss], label="HFSS")
    axes[1][1].plot(theta_hfss, hfss_e_phi_db_array_steered[h_plane_phi].loc[theta_hfss])
    axes[2][1].plot(theta_hfss, hfss_e_tot_db_array_steered[h_plane_phi].loc[theta_hfss])
    axes[0][1].plot(theta_hfss, to_db(numpy.abs(h_theta_hfss_combined_array) / array_rotated_max, db=20), '.',
                    label="Analytical")
    axes[1][1].plot(theta_hfss, to_db(numpy.abs(h_phi_hfss_combined_array) / array_rotated_max, db=20), '.')
    axes[2][1].plot(theta_hfss, to_db(numpy.abs(h_tot_hfss_combined_array) / array_rotated_max, db=20), '.')
    axes[0][1].legend()

    for ax in axes:
        for _ax in ax:
            _ax.grid(True)
```

```
            _ax.set_ylim([y_min, 1])
            _ax.set_yticks(numpy.arange(y_min, 0.1, step=6))
            _ax.set_xlim([-90, 90])
            _ax.set_xticks(numpy.arange(-90, 91, step=30))
    for ax in axes[:, 0]:
        ax.set_xlim([-90 + rad2deg(beta), 90 + rad2deg(beta)])
        ax.set_xticks(numpy.arange(-90 + rad2deg(beta), 91 + rad2deg(beta),
            step=30))
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.savefig(
        f'../parts/Theoretical␣Foundation/
            hfss_analytically_rotated_array_vs_hfss_array.pdf',
        transparent=True, bbox_inches='tight', pad_inches=0)
```

B.13 Validation Model for Optimizing Array

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
from scipy.optimize import minimize

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', False)
numpy.set_printoptions(edgeitems=30, linewidth=100000)

costs = []

if __name__ == '__main__':
    f = 0.433e9
    lamb = c0 / f
    theta_min = 0
    theta_max = 180
    theta_deg = numpy.linspace(theta_min, theta_max, 200)
    full_theta_deg = numpy.linspace(theta_min, theta_max, 200)
    theta_grid = deg2rad(theta_deg)
    full_theta_grid = deg2rad(full_theta_deg)
    n_theta_hfss = len(theta_deg)
    e_plane_phi = 90
    phi_plane = numpy.full_like(theta_grid, deg2rad(e_plane_phi))

    E_theta, E_phi, directivity = get_Etheta_Ephi_directivity(full_theta_grid,
        phi_plane)

    # Plotting 1 element
    fig_size = 3
    fig, axes = plt.subplots(nrows=3, ncols=1, sharey=True, sharex=True,
        figsize=(8, 9))
    axes[0].plot(full_theta_deg, to_db(numpy.abs(E_phi).astype(float), db=20),
        label=f'One␣element')
    axes[0].set_ylabel(f"$E_\\phi$␣[dB]")
```

```python
    axes[1].set_ylabel(f"$E_\\theta$ [dB]")
    axes[2].set_ylabel(f"D [dB]")
    axes[1].plot(full_theta_deg, to_db(numpy.abs(E_theta).astype(float), db=20), label=f'One element')
    axes[2].set_xlabel(r"$\theta [^\circ]$")
    axes[1].set_xlim([0, 180])
    axes[2].plot(full_theta_deg, to_db(numpy.abs(directivity), db=20), label=f'One element')
    for ax in axes:
        ax.grid()
        ax.legend()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.set_tight_layout(True)

    N = 3
    # Distance must be greater than 2*L
    distance = 0.5 * lamb
    d = distance * numpy.arange(-(N - 1) / 2, (N / 1) / 2, 1)
    a = numpy.ones(N)

    def fields_func(theta, phi):
        _e_theta, _e_phi, _dir = get_Etheta_Ephi_directivity(theta, phi)
        return _e_theta, _e_phi

    antenna_pos = numpy.vstack([numpy.zeros_like(d),
                                d,
                                numpy.zeros_like(d)]).T

    alphas = deg2rad(numpy.array(N * [0]))
    betas = deg2rad(numpy.array(N * [-90]))
    gammas = deg2rad(numpy.array(N * [55]))

    e_theta_steered_array, e_phi_steered_array = combining_general_array(
        antenna_pos=antenna_pos, fields_func=fields_func,
        a=a, theta=theta_grid, phi=phi_plane,
        lamb=lamb, alphas=alphas,
        betas=betas, gammas=gammas)

    e_tot_steered_array = numpy.sqrt(numpy.abs(e_theta_steered_array) ** 2 + numpy.abs(e_phi_steered_array) ** 2)
    e_theta_steered_array = e_theta_steered_array.T
    e_phi_steered_array = e_phi_steered_array.T
    e_tot_steered_array = e_tot_steered_array.T

    f_d_theta = e_theta_steered_array
    f_d_phi = e_phi_steered_array
    epsilon = 1
    max_iterations = 500


    def get_e_theta_e_phi_from_x(x):
        x = x.reshape(-1, n_var)
        pos = x[:, :3]
        alphas = x[:, 3]
        betas = x[:, 4]
        gammas = x[:, 5]
        a_module = x[:, 6]
        a_phase = x[:, 7]
        a = a_module * exp(1j * a_phase)
```

```
        _e_theta, _e_phi = combining_general_array(
            antenna_pos=pos, fields_func=fields_func,
            a=a, theta=theta_grid, phi=phi_plane,
            lamb=lamb, alphas=alphas,
            betas=betas, gammas=gammas)
        return _e_theta, _e_phi


    def func_to_minimize(x):
        global costs
        _e_theta, _e_phi = get_e_theta_e_phi_from_x(x)
        f_theta = _e_theta
        f_theta = f_theta.T
        f_phi = _e_phi
        f_phi = f_phi.T
        min_value = numpy.sum(numpy.abs(f_theta - f_d_theta)) + numpy.sum(numpy.abs(f_phi - f_d_phi))
        print(f"Difference: {min_value}")
        costs.append(min_value)
        return min_value


    def constraint_function(x):
        _e_theta, _e_phi = get_e_theta_e_phi_from_x(x)
        f_theta = _e_theta
        f_theta = f_theta.T
        f_phi = _e_phi
        f_phi = f_phi.T
        constraint = epsilon - numpy.sum(numpy.abs(f_theta - f_d_theta)) - numpy.sum(numpy.abs(f_phi - f_d_phi))
        return constraint


    d0 = 0.8 * lamb
    n_var = 8
    x0 = numpy.zeros(shape=(N, n_var))
    x0[:, :3] = numpy.vstack([numpy.array([0.2, 0.2, 0.2]) * d0,
                              d0 * numpy.arange(-(N - 1) / 2, (N / 1) / 2, 1),
                              numpy.zeros(N)]).T
    x0[:, 3] = deg2rad(numpy.array(N * [0]))
    x0[:, 4] = deg2rad(numpy.array(N * [-90]))
    x0[:, 5] = deg2rad(numpy.array([90, 30, 110]))
    x0[:, 6] = numpy.ones(N)
    x0[:, 7] = numpy.zeros(N)

    x0 = x0.flatten()

    bounds = N * [[-N * lamb, N * lamb], [-N * lamb, N * lamb], [0, N * lamb], [-pi / 2, pi / 2], [-pi / 2, pi / 2],
                  [-pi / 2, pi / 2], [0, 1], [0, 2 * pi]]

    constraints = (
        {
            'type': 'ineq',
            'fun': constraint_function
        })
    algorithm = "SLSQP"
    res = minimize(func_to_minimize, x0, method=algorithm, bounds=bounds, constraints=constraints,
```

```
                   options={'maxiter': max_iterations, 'disp': True})

    x = res.x
    x = x.reshape(-1, n_var)
    pos = x[:, :3]
    _alphas = x[:, 3]
    _betas = x[:, 4]
    _gammas = x[:, 5]
    _a_module = x[:, 6]
    _a_phase = x[:, 7]
    _a = _a_module * exp(1j * _a_phase)
    e_theta_opt, e_phi_opt = combining_general_array(
        antenna_pos=pos, fields_func=fields_func,
        a=_a, theta=theta_grid, phi=phi_plane,
        lamb=lamb, alphas=_alphas,
        betas=_betas, gammas=_gammas
    )
    e_tot_opt = numpy.sqrt(numpy.abs(e_theta_opt) ** 2 + numpy.abs(e_phi_opt) ** 2)
    e_theta_opt = e_theta_opt.T
    e_phi_opt = e_phi_opt.T
    e_tot_opt = e_tot_opt.T

    print("Desired array positions:")
    print(antenna_pos / lamb)
    print("Final array positions:")
    print(pos / lamb)
    print("Desired array alphas:")
    print(rad2deg(alphas))
    print("Final array alphas:")
    print(rad2deg(_alphas))
    print("Desired array betas:")
    print(rad2deg(betas))
    print("Final array betas:")
    print(rad2deg(_betas))
    print("Desired array gammas:")
    print(rad2deg(gammas))
    print("Final array gammas:")
    print(rad2deg(_gammas))
    print("Desired a:")
    print(a)
    print("Final a:")
    print(_a)

    # Plotting results
    fig, axes = plt.subplots(nrows=3, ncols=1, sharey=True, sharex=True, figsize=(8, 6))
    axes[0].plot(theta_deg, to_db(numpy.abs(e_phi_steered_array).astype(float), db=20),
                 label=f'Array with {N} elements')
    axes[0].plot(theta_deg, to_db(numpy.abs(e_phi_opt).astype(float), db=20), '--', label=f'SQLQ result')
    axes[0].set_ylabel(f"$E_\\phi$ [dB]")
    axes[1].set_ylabel(f"$E_\\theta$ [dB]")
    axes[1].plot(theta_deg, to_db(numpy.abs(e_theta_steered_array).astype(float), db=20),
                 label=f'Array with {N} elements')
    axes[1].plot(theta_deg, to_db(numpy.abs(e_theta_opt).astype(float), db=20), '--',
```

```
                     label=f'SQLQ result')
    axes[-1].set_xlabel(r"$\theta [^\circ]$")
    axes[-1].set_xlim([theta_min, theta_max])
    axes[2].set_ylabel(f"$|E|$ [dB]")
    axes[2].plot(theta_deg, to_db(numpy.abs(e_tot_steered_array).astype(float), db=20), label=f'Array with {N} elements')
    axes[2].plot(theta_deg, to_db(numpy.abs(e_tot_opt).astype(float), db=20), '--', label=f'SQLQ result')
    for ax in axes:
        ax.grid()
        ax.legend()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.set_tight_layout(True)

    # Plotting cost
    fig1, axes1 = plt.subplots(nrows=1, ncols=1, sharey=True, sharex=True, figsize=(8, 6))
    axes1.set_yscale("log")
    axes1.plot(numpy.array(costs), '.', label="Cost")
    axes1.legend()
    axes1.grid()
    axes1.set_xlim([0, len(costs)])
    axes1.set_title(f'Final cost = {numpy.min(numpy.array(costs)):0.2E}')
    axes1.set_xlabel("Number of Evaluations")
    axes1.set_ylabel("Cost")
    fig.set_tight_layout(True)

    df = pandas.DataFrame({
        '$x [\lambda]$': x[:, 0] / lamb,
        '$y [\lambda]$': x[:, 1] / lamb,
        '$z [\lambda]$': x[:, 2] / lamb,
        '$\\alpha [^\circ]$': rad2deg(_alphas),
        '$\\beta [^\circ]$': rad2deg(_betas),
        '$\\gamma [^\circ]$': rad2deg(_gammas)
    })

    save_results = True
    if save_results:
        df.to_csv(f'./../../parts/Antenna Array/array_configuration_{algorithm}.csv')
        df.to_latex(f'./../../parts/Antenna Array/array_configuration_{algorithm}.tex', float_format="%.2f")
        fig.savefig(f'./../../parts/Antenna Array/field_comparison_{algorithm}.pdf', transparent=True,
                    bbox_inches='tight', pad_inches=0)
        fig1.savefig(f'./../../parts/Antenna Array/algorithm_convergence_{algorithm}.pdf', transparent=True,
                     bbox_inches='tight', pad_inches=0)
```

B.14 Optimizing Array Pattern Design

```
from arraytools import *
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
```

```python
from scipy.optimize import minimize

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', False)
numpy.set_printoptions(edgeitems=30, linewidth=100000)


costs = []

if __name__ == '__main__':
    f = 0.433e9
    lamb = c0 / f
    theta_min = 0
    theta_max = 90
    fov_theta = 70
    maximum_e_theta_drop = 0.5
    theta_deg = numpy.linspace(theta_min, fov_theta, 200)
    full_theta_deg = numpy.linspace(theta_min, theta_max, 200)
    theta_grid = deg2rad(theta_deg)
    full_theta_grid = deg2rad(full_theta_deg)
    phi_deg = numpy.linspace(0, 360, 200)
    phi_grid = deg2rad(phi_deg)
    n_theta_hfss = len(theta_deg)
    n_phi_hfss = len(phi_deg)
    e_plane_phi = 90
    phi_plane = numpy.full_like(theta_grid, deg2rad(e_plane_phi))

    E_theta, E_phi, directivity = get_Etheta_Ephi_directivity(full_theta_grid,
        phi_plane)

    # Plotting 1 element
    fig_size = 3
    fig, axes = plt.subplots(nrows=3, ncols=1, sharey=True, sharex=True,
        figsize=(8, 9))
    axes[0].plot(full_theta_deg, to_db(numpy.abs(E_phi).astype(float), db=20),
        label=f'One element')
    axes[0].set_ylabel(f"$E_\\phi$ [dB]")
    axes[1].set_ylabel(f"$E_\\theta$ [dB]")
    axes[2].set_ylabel(f"D [dB]")
    axes[1].plot(full_theta_deg, to_db(numpy.abs(E_theta).astype(float), db=20)
        , label=f'One element')
    axes[2].set_xlabel(r"$\theta [^\circ]$")
    axes[1].set_xlim([theta_min, theta_max])
    axes[2].plot(full_theta_deg, to_db(numpy.abs(directivity), db=20), label=f'
        One element')
    for ax in axes:
        ax.grid()
        ax.legend()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.set_tight_layout(True)

    # Plotting 1 element
    fig_size = 3
    fig, axes = plt.subplots(nrows=3, ncols=1, sharey=True, sharex=True,
        figsize=(8, 9))
```

```
    axes[0].plot(full_theta_deg, numpy.abs(E_phi).astype(float), label=f'One element')
    axes[0].set_ylabel(f"$E_\\phi$")
    axes[1].set_ylabel(f"$E_\\theta$")
    axes[2].set_ylabel(f"D")
    axes[1].plot(full_theta_deg, numpy.abs(E_theta).astype(float), label=f'One element')
    axes[2].set_xlabel(r"$\theta [^\circ]$")
    axes[1].set_xlim([theta_min, theta_max])
    axes[2].plot(full_theta_deg, numpy.abs(directivity), label=f'One element')
    for ax in axes:
        ax.grid()
        ax.legend()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.set_tight_layout(True)


def fields_func(theta, phi):
    _e_theta, _e_phi, _dir = get_Etheta_Ephi_directivity(theta, phi)
    return _e_theta, _e_phi


N = 5
a = numpy.ones(N)
epsilon = 0
max_iterations = 300


def get_e_theta_e_phi_from_x(x):
    x = x.reshape(-1, 6)
    pos = x[:, :3]
    alphas = x[:, 3]
    betas = x[:, 4]
    gammas = x[:, 5]
    _e_theta, _e_phi = combining_general_array(
        antenna_pos=pos, fields_func=fields_func,
        a=a, theta=theta_grid, phi=phi_plane,
        lamb=lamb, alphas=alphas,
        betas=betas, gammas=gammas)
    return _e_theta, _e_phi


def func_to_minimize(x):
    global costs
    _e_theta, _e_phi = get_e_theta_e_phi_from_x(x)
    f_theta = _e_theta
    f_theta = f_theta.T
    f_phi = _e_phi
    f_phi = f_phi.T
    e_tot = numpy.sqrt(numpy.abs(f_theta)**2 + numpy.abs(f_theta)**2)
    delta = e_tot.max() / e_tot.min() - epsilon
    print(f"Delta: {delta}")
    costs.append(delta)
    return delta


distance = 0.5 * lamb
d0 = distance * numpy.arange(0, N, 1)
x0 = numpy.zeros(shape=(N, 6))
```

```
    x0[:, :3] = numpy.vstack([numpy.array(N * [0]) * d0,
                              d0,
                              numpy.zeros_like(d0)]).T
    x0[:, 3] = deg2rad(numpy.array(N * [0]))
    x0[:, 4] = deg2rad(numpy.array(N * [0]))
    x0[:, 5] = deg2rad(numpy.array(N * [0]))

    df0 = pandas.DataFrame({
        '$x_0␣[\lambda]$': x0[:, 0] / lamb,
        '$y_0␣[\lambda]$': x0[:, 1] / lamb,
        '$z_0␣[\lambda]$': x0[:, 2] / lamb,
        '$\\alpha_0␣[^\circ]$': rad2deg(x0[:, 3]),
        '$\\beta_0␣[^\circ]$': rad2deg(x0[:, 4]),
        '$\\gamma_0␣[^\circ]$': rad2deg(x0[:, 5])
    })

    x0 = x0.flatten()

    bounds = N * [[-N * lamb, N * lamb], [-N * lamb, N * lamb], [0, N * lamb],
        [-pi / 2, pi / 2], [-pi / 2, pi / 2],
                  [-pi / 2, pi / 2]]

    algorithm = 'L-BFGS-B'
    res = minimize(func_to_minimize, x0, method=algorithm, bounds=bounds,
                   options={'maxiter': max_iterations, 'disp': True})

    x = res.x
    x = x.reshape(-1, 6)
    pos = x[:, :3]
    _alphas = x[:, 3]
    _betas = x[:, 4]
    _gammas = x[:, 5]
    e_theta_result, e_phi_result = combining_general_array(
        antenna_pos=pos, fields_func=fields_func,
        a=a, theta=full_theta_grid, phi=phi_plane,
        lamb=lamb, alphas=_alphas,
        betas=_betas, gammas=_gammas)

    e_tot_steered_array_sqlq = numpy.sqrt(numpy.abs(e_theta_result) ** 2 +
        numpy.abs(e_phi_result) ** 2)
    e_theta_result = e_theta_result.T
    e_phi_result = e_phi_result.T
    e_tot_steered_array_sqlq = e_tot_steered_array_sqlq.T

    print("Final␣array␣positions:")
    print(pos)
    print("Final␣array␣alphas:")
    print(rad2deg(_alphas))
    print("Final␣array␣betas:")
    print(rad2deg(_betas))
    print("Final␣array␣gammas:")
    print(rad2deg(_gammas))

    # Plotting results
    fig_size = 3
    fig, axes = plt.subplots(nrows=2, ncols=1, sharey=True, sharex=True,
        figsize=(8, 6))
    axes[0].plot(full_theta_deg, to_db(numpy.abs(e_phi_result).astype(float),
        db=20), label=f'SQLQ␣result')
```

```
    axes[0].set_ylabel(f"$E_\\phi$ [dB]")
    axes[1].set_ylabel(f"$E_\\theta$ [dB]")
    axes[1].plot(full_theta_deg, to_db(numpy.abs(e_theta_result).astype(float), db=20), label=f'SQLQ result')
    axes[1].set_xlabel(r"$\theta [^\circ]$")
    axes[1].set_xlim([theta_min, theta_max])
    for ax in axes:
        ax.grid()
        ax.legend()
    plt.subplots_adjust(wspace=0.1, hspace=0.1)
    fig.set_tight_layout(True)

    # Plotting cost
    fig1, axes1 = plt.subplots(nrows=1, ncols=1, sharey=True, sharex=True, figsize=(8, 6))
    axes1.set_yscale("log")
    axes1.plot(numpy.array(costs), '.', label="Cost")
    axes1.legend()
    axes1.grid()
    axes1.set_xlim([0, len(costs)])
    axes1.set_title(f'Final cost = {numpy.min(numpy.array(costs)):0.2E}')
    axes1.set_xlabel("Number of Evaluations")
    axes1.set_ylabel("Cost")
    fig.set_tight_layout(True)

    df = pandas.DataFrame({
        '$x [\lambda]$': x[:, 0] / lamb,
        '$y [\lambda]$': x[:, 1] / lamb,
        '$z [\lambda]$': x[:, 2] / lamb,
        '$\\alpha [^\circ]$': rad2deg(_alphas),
        '$\\beta [^\circ]$': rad2deg(_betas),
        '$\\gamma [^\circ]$': rad2deg(_gammas)
    })
    print(df0)
    print(df)
    save_results = False
    if save_results:
        sufix = f"for_{fov_theta}_degrees_{N}_antennas_{algorithm}"
        fig.savefig(f'./../../parts/Antenna Array/field_result_{sufix}.pdf',
                    transparent=True, bbox_inches='tight', pad_inches=0)
        fig1.savefig(f'./../../parts/Antenna Array/algorithm_convergence_{sufix}.pdf',
                     transparent=True, bbox_inches='tight', pad_inches=0)
        df.to_csv(f'./../../parts/Antenna Array/array_configuration_{sufix}.csv')
        df.to_latex(f'./../../parts/Antenna Array/array_configuration_{sufix}.tex',
                    float_format="%.2f")
        df0.to_latex(
            f'./../../parts/Antenna Array/initial_array_configuration_{sufix}.tex',
            float_format="%.2f")
```

B.15 Optimizing Array from Group Passes

```
from arraytools import *
```

```python
import numpy
import pandas
from matplotlib import pyplot as plt
from matplotlib import use, rc
from datetime import datetime
import multiprocessing
import requests
import json
import configparser
from scipy.optimize import minimize
from time import sleep
from aftk.antenna.eigenantenna import get_eigenantennas_smc, get_max_directivity

rc('text', usetex=True)
rc('font', family='serif')
use('Qt5Agg')
plt.rcParams.update({'font.size': 15})
pandas.set_option('expand_frame_repr', False)
pandas.set_option('display.max_rows', False)
numpy.set_printoptions(edgeitems=30, linewidth=100000)

costs = []

if __name__ == '__main__':
    num_cores = multiprocessing.cpu_count()
    f = 2244e6
    lamb = c0 / f
    sat_number = "56215"
    start_time = '2024-03-04 00:00:00.000'
    stop_time = '2024-05-03 23:59:59.999'
    prop_end_time = stop_time

    update_tle = False
    if update_tle:
        df = get_tle_from_spacetrack(sat_number, start_time, stop_time)
        line1 = df.iloc[0].TLE_LINE1
        line2 = df.iloc[0].TLE_LINE2
    else:
        line1 = '1 56215U 23054AP  24064.22928279  .00040375  00000-0  10529-2 0  9998'
        line2 = '2 56215  97.3763 323.0596 0008862 130.9637 229.2376 15.38687649 50033'

    sat = Satellite(eirp=2, start_time=start_time, end_time=prop_end_time, N=500000, calc_gain_pattern_sym=False,
                    line1=line1, line2=line2, f=f)
    main_land = brazil_mainland()
    long_min, lat_min, long_max, lat_max = main_land.bounds
    station_lon, station_lat = main_land.centroid.coords[0]
    ground_antenna = Parabola(f=f, D=1.5 / lamb, eff=0.5, temp=312, bandwidth=6e6)
    station = Station(f=f, lat=station_lat, lon=station_lon, e_theta_e_phi_function=ground_antenna.get_fields_sym)
    link = LinkBudget(satellite=sat, station=station, R_spec=10e6, Eb_N0_min=7.5, calc_transmitted_data=False,
                      G_other=-1.6)
    # Getting only the biggest pass
```

```
    link.passes.loc[:, 'max elev'] = link.passes['max elev'].astype(float).apply(numpy.rad2deg)
    link.passes.loc[:, 'mean azi'] = link.passes['mean azi'].astype(float).apply(numpy.rad2deg)
    link.passes.sort_values(by='duration', inplace=True, ascending=False)
    # link.passes = link.passes[link.passes['max elev'] > 5]

    n_theta = 90
    n_phi = 180
    theta_max_deg = 180
    theta_deg = numpy.linspace(0, theta_max_deg, n_theta)
    phi_deg = numpy.linspace(0, 360, n_phi)
    theta_grid = deg2rad(theta_deg)
    phi_grid = deg2rad(phi_deg)
    Theta, Phi = numpy.meshgrid(theta_grid, phi_grid, indexing='ij')
    Theta = Theta.flatten()
    Phi = Phi.flatten()

    l_max = 5
    q = get_eigenantennas_smc(l_max, 0, 0)[:, 0].reshape(-1, 1)
    # q = "./yagi.smc.est"
    # Best design:
    # for N, elev_min, elev_max in [(4, 20, 30), (5, 30, 40), (5, 40, 50), (6, 60, 70), (9, 70, 80), (8, 80, 90), (9, 80, 90)]:
    for N in [5]:
        for elev_min, elev_max in [(60, 70)]:
            filter_direction = True
            direction = "descending"
            right_passes = True
            left_passes = False
            max_iterations = 500
            from_last_run = True
            minimize_dir = False
            use_constraints = False
            minimize = "e_tot"
            algorithm = "COBYLA"
            algorithm = "L-BFGS-B"
            algorithm = "SLSQP"
            # optimize_for_x0 = "angles_and_a_factor"
            optimize_for_x0 = "position_and_angles"
            direction_for_x0 = "descending_right"
            # optimize = "all"
            # optimize = "angles_and_a_factor"
            optimize = "position_and_angles"
            print(f"Optimizing {optimize} for {N} antennas, from elevation {elev_min} to {elev_max}.")
            print(
                f"from_last_run = {from_last_run} | right_passes = {right_passes} | left_passes = {left_passes} | use_constraints = {use_constraints}")
            # Plotting passes
            fig_mask, axes_mask = plt.subplots(figsize=(8, 6))
            axes_mask.plot(*main_land.exterior.coords.xy, color='black')
            axes_mask.plot(station_lon, station_lat, "x", color='black')
            if filter_direction:
                filtered_passes = link.passes[link.passes['direction'] == direction].copy()
            else:
                filtered_passes = link.passes.copy()
```

```
        filtered_passes = filtered_passes[filtered_passes['max␣elev'] > elev_min].copy()
        filtered_passes = filtered_passes[filtered_passes['max␣elev'] < elev_max].copy()
        if right_passes and not left_passes:
            filtered_passes = filtered_passes[filtered_passes['position'] == 'right']
            direction += "_right"
        elif left_passes and not right_passes:
            filtered_passes = filtered_passes[filtered_passes['position'] == 'left']
            direction += "_left"
        else:
            direction += "_right_left"
        for index, row in filtered_passes.iterrows():
            _lat = link.data[row.init:row.end]['lat_WGS84_deg']
            _lon = link.data[row.init:row.end]['lon_WGS84_deg']
            axes_mask.plot(_lon, _lat, '.')
        axes_mask.set_aspect('equal')
        axes_mask.set_xlabel('Degrees␣Longitude')
        axes_mask.set_ylabel('Degrees␣Latitude')
        axes_mask.set_title(f'Pass␣mask')
        fig_mask.set_size_inches(8, 6)
        axes_mask.set_xlim([long_min, long_max])
        axes_mask.set_ylim([lat_min, lat_max])
        fig_mask.tight_layout()
        plt.show()
        print("Filtered␣passes:")
        print(filtered_passes)
        # Selecting pass
        el_rad = numpy.concatenate(
            [link.data[row.init:row.end]['el_rad'].values for index, row in filtered_passes.iterrows()])
        az_rad = numpy.concatenate(
            [link.data[row.init:row.end]['az_rad'].values for index, row in filtered_passes.iterrows()])
        # Polar angle = 90 - Elevation
        pass_theta_grid = pi / 2 - el_rad
        # Azimuth angle = 180 - Azimuth
        pass_phi_grid = pi - az_rad
        pass_phi_grid[pass_phi_grid < 0] += 2 * pi
        pass_theta_deg = rad2deg(pass_theta_grid)
        pass_phi_deg = rad2deg(pass_phi_grid)
        kx, ky, kz = from_spherical_to_cartesian(numpy.ones_like(pass_theta_grid), pass_theta_grid, pass_phi_grid)
        k = numpy.vstack([kx, ky, kz])
        k0 = k[:, 0]
        kn = k[:, -1]
        satellite_direction = kn - k0
        k = numpy.vstack([kx, ky, kz])
        theta_min = min(pass_theta_deg)
        theta_max = max(pass_theta_deg)
        phi_min = min(pass_phi_deg)
        phi_max = max(pass_phi_deg)

        # Plotting az, el, theta_grid and phi_grid
        figure_pass, axes_pass = plt.subplots(2, 2, sharex=True, figsize=(10, 10))
        i = 0
```

```
            for index, row in filtered_passes.iterrows():
                data = link.data[row.init:row.end]
                _el_rad = data.el_rad.values
                _az_rad = data.az_rad.values
                _theta_grid = pi / 2 - _el_rad
                _phi_grid = pi - _az_rad
                _theta_deg = rad2deg(_theta_grid)
                _phi_deg = rad2deg(_phi_grid)
                axes_pass[0][0].plot(data.index, numpy.rad2deg(_az_rad), '.', color=f"C{i}")
                axes_pass[0][0].set_title('Azimuth')
                axes_pass[0][1].plot(data.index, numpy.rad2deg(_el_rad), '.', color=f"C{i}")
                axes_pass[0][1].set_title('Elevation')

                axes_pass[1][0].plot(data.index, _phi_deg, '.', color=f"C{i}")
                axes_pass[1][0].set_title(r'$\phi$')
                axes_pass[1][1].plot(data.index, _theta_deg, '.', color=f"C{i}")
                axes_pass[1][1].set_title(r'$\theta$')
                i += 1
            for ax in axes_pass:
                for _ax in ax:
                    _ax.grid()
            # xloc = md.MinuteLocator(interval=2)
            xloc = md.DayLocator(interval=5)
            # majorFmt = md.DateFormatter('%H:%M')
            majorFmt = md.DateFormatter('%d/%m')
            axes_pass[0][-1].xaxis.set_major_locator(xloc)
            axes_pass[1][-1].xaxis.set_major_locator(xloc)
            axes_pass[0][-1].xaxis.set_major_formatter(majorFmt)
            axes_pass[1][-1].xaxis.set_major_formatter(majorFmt)
            figure_pass.set_tight_layout(True)

            folder_path = './../../parts/Antenna Array/'
            x0_sufix = f'{N}_antennas_{algorithm}_{optimize_for_x0}_multipass_from_{elev_min}_to_{elev_max}_{direction_for_x0}_l_max_{l_max}_minimizing_{minimize}'
            x0_path = f'{folder_path}array_configuration_{x0_sufix}.csv'
            sufix = f'{N}_antennas_{algorithm}_{optimize}_multipass_from_{elev_min}_to_{elev_max}_{direction}_l_max_{l_max}_minimizing_{minimize}'

            if from_last_run:
                a_threshold = 0
                first_df0 = pandas.read_csv(x0_path, index_col=0)
                first_df0 = first_df0[first_df0['$|a|$'].values > a_threshold]
                N = len(first_df0)
                _x0 = first_df0['$x [\\lambda]$'].values * lamb
                _y0 = first_df0['$y [\\lambda]$'].values * lamb
                _z0 = first_df0['$z [\\lambda]$'].values * lamb
                alphas_0 = deg2rad(first_df0['$\\alpha [^\\circ]$'].values)
                betas_0 = deg2rad(first_df0['$\\beta [^\\circ]$'].values)
                gammas_0 = deg2rad(first_df0['$\\gamma [^\\circ]$'].values)
                initial_pos = numpy.vstack([_x0,
                                            _y0,
                                            _z0]).T
                if optimize == "angles_and_a_factor" or optimize == "a_factor":
                    a = first_df0['$|a|$'].values
```

```python
                a_phase = deg2rad(first_df0['$\\phase{a}[^\\circ]$'].values)
            else:
                a = numpy.ones(N)
                a_phase = numpy.zeros(N)
        else:
            distance = 0.5 * lamb
            pos_x0 = numpy.zeros(N)
            pos_y0 = distance * numpy.arange(-(N - 1) / 2, (N / 1) / 2, 1)
            pos_z0 = numpy.zeros(N)
            if N % 2 == 0:
                pointing_el = numpy.linspace(numpy.min(el_rad), numpy.max(el_rad), int((N + 2) / 2))[1:]
                pointing_el = numpy.concatenate([pointing_el, pointing_el[::-1]])
                pointing_theta = (pi / 2 - pointing_el) * numpy.ones(N)
            else:
                pointing_el = numpy.linspace(numpy.min(el_rad), numpy.max(el_rad), int((N + 3) / 2))[1:]
                pointing_el = numpy.concatenate([pointing_el[:-1], pointing_el[::-1]])
                pointing_theta = (pi / 2 - pointing_el) * numpy.ones(N)
            if right_passes:
                pointing_azi = numpy.linspace(numpy.min(az_rad), numpy.max(az_rad), N)
                gammas_0 = pointing_azi - pi / 2
                betas_0 = numpy.zeros(N)
                alphas_0 = pointing_theta  # pitch
            else:
                pointing_phi = numpy.linspace(numpy.min(pass_phi_grid), numpy.max(pass_phi_grid), N)
                gammas_0 = pi / 2 - pointing_phi  # yaw
                betas_0 = numpy.zeros(N)
                alphas_0 = pointing_theta  # pitch
            a = numpy.ones(N)
            a_phase = numpy.zeros(N)
            initial_pos = numpy.vstack([pos_x0,
                                        pos_y0,
                                        pos_z0]).T

        _e_theta_0, _e_phi_0 = combining_physical_steered_array(
            initial_pos, a, Theta, Phi, lamb,
            alphas=alphas_0, betas=betas_0, gammas=gammas_0,
            antenna=q, l_max=l_max)
        _e_tot_0 = numpy.sqrt(numpy.abs(_e_theta_0) ** 2 + numpy.abs(_e_phi_0) ** 2)
        fig0, axes0 = plot_polar_contour_mag_db(Theta, Phi, to_db(_e_tot_0, db=20),
                                                title=f'$|E|$ for {N} antennas', ylabel=f"$|E|$ [dB]")
        axes0.plot(pass_phi_grid, pass_theta_deg, '.', color='white')
        plt.show()
        res, e_theta_result, e_phi_result, df0, df, costs = minimize_pattern_diff(
            theta_grid=pass_theta_grid, phi_grid=pass_phi_grid, lamb=lamb, N=N, optimize=optimize,
            a=a, a_phase=a_phase, algorithm=algorithm,
            initial_pos=initial_pos, use_constraints=use_constraints,
```

```
            antenna=q, l_max=l_max, minimize=minimize,
            alphas_0=alphas_0, betas_0=betas_0, gammas_0=gammas_0, max_iterations=max_iterations)

        e_tot_result = numpy.sqrt(numpy.abs(e_theta_result) ** 2 + numpy.abs(e_phi_result) ** 2)
        e_theta_result = e_theta_result.T
        e_phi_result = e_phi_result.T
        e_tot_result = e_tot_result.T
        xf = df['$x [\\lambda]$'].values * lamb
        yf = df['$y [\\lambda]$'].values * lamb
        zf = df['$z [\\lambda]$'].values * lamb
        final_alphas = deg2rad(df['$\\alpha [^\\circ]$'].values)
        final_betas = deg2rad(df['$\\beta [^\\circ]$'].values)
        final_gammas = deg2rad(df['$\\gamma [^\\circ]$'].values)
        final_a = df['$|a|$'].values * numpy.exp(1j * df['$\\phase{a}[^\\circ]$'].values)
        final_pos = numpy.vstack([xf,
                                  yf,
                                  zf]).T
        distances = numpy.array(
            [numpy.min(
                numpy.sum(numpy.sqrt((final_pos[k, :] - numpy.delete(final_pos, k, axis=0)) ** 2), axis=1)) / lamb
             for k in
             range(len(final_pos))])
        min_dist = numpy.min(distances)
        df["Min Distance $[\\lambda]$"] = distances
        if from_last_run:
            print(first_df0)
        print(df0)
        print(df)
        print(f"Min dist: {min_dist}")
        e_theta_full, e_phi_full = combining_physical_steered_array(
            final_pos, final_a, Theta, Phi, lamb,
            antenna=q, l_max=l_max,
            alphas=final_alphas, betas=final_betas,
            gammas=final_gammas)
        e_tot_full = numpy.sqrt(numpy.abs(e_theta_full) ** 2 + numpy.abs(e_phi_full) ** 2)
        # Plotting results
        fig, axes = plot_fields_versus_phi(e_theta_result, e_phi_result, e_tot_result, pass_phi_deg)
        fig_theta, axes_theta = plot_fields_versus_phi(e_theta_result, e_phi_result, e_tot_result, pass_theta_deg)
        # Plotting cost
        fig1, axes1 = plt.subplots(nrows=1, ncols=1, sharey=True, sharex=True, figsize=(8, 6))
        axes1.set_yscale("log")
        axes1.plot(numpy.array(costs), '.', label="Cost")
        axes1.legend()
        axes1.grid()
        axes1.set_xlim([0, len(costs)])
        axes1.set_title(f'Final cost = {numpy.min(numpy.array(costs)):0.2E}')
        axes1.set_xlabel("Number of Evaluations")
        axes1.set_ylabel("Cost")
        fig1.set_tight_layout(True)

```

```
        save_results = True

        fig2d, axes2d = plot_polar_contour_mag_db(Theta, Phi, to_db(e_tot_full, db=20),
                                                  title=f'$|E|$ for {N} antennas', ylabel=f"$|E|$ [dB]")
        axes2d.plot(pass_phi_grid, pass_theta_deg, '.', color='white')

        if save_results:
            fig.savefig(f'{folder_path}field_result_{sufix}.pdf', transparent=True)
            fig1.savefig(f'{folder_path}algorithm_convergence_{sufix}.pdf', transparent=True)
            df.to_csv(f'{folder_path}array_configuration_{sufix}.csv')
            df.to_latex(f'{folder_path}array_configuration_{sufix}.tex', float_format="%.2f")
            df0.to_csv(f'{folder_path}initial_array_configuration_{sufix}.csv')
            df0.to_latex(f'{folder_path}initial_array_configuration_{sufix}.tex', float_format="%.2f")
            fig2d.savefig(f'{folder_path}field_2d_{sufix}.pdf', transparent=True)

        fig_pos, axes_pos = plt.subplots(subplot_kw={'projection': '3d'})
        axes_pos.scatter(df["$x [\lambda]$"].values, df["$y [\lambda]$"].values, df["$z [\lambda]$"].values)
```

References

[1] T. T. Taylor. *One Parameter Family of Line Sources Producing sin $\pi u/\pi u$ Patterns*. 1953. URL: "http://www.ece.rutgers.edu/~orfanidi/ewa/taylor-1953.pdf".

[2] J. Kaiser and R. Schafer. "On the use of the I0-sinh window for spectrum analysis". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.1 (1980), pp. 105–107. DOI: 10.1109/TASSP.1980.1163349.

[3] Popescu Theodor. "Introduction to signal processing, by S.J. Orfanidis, Prentice Hall Signal Processing Series, Prentice Hall, Upper Saddle River, New Jersey, 1996 - Book review". In: *Control Engineering Practice* 4 (Dec. 1996), 1771–1772. DOI: 10.1016/S0967-0661(96)90009-X.

[4] Rainer Storn and Kenneth Price. "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11 (Jan. 1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[5] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*. Second. Prentice-hall Englewood Cliffs, 1999.

[6] X. Huang, U. Behr, and W. Wiesbeck. "Automatic base station placement and dimensioning for mobile network planning". In: *Vehicular Technology Conference Fall 2000. IEEE VTS Fall VTC2000. 52nd Vehicular Technology Conference (Cat. No.00CH37152)*. Vol. 4. 2000, 1544–1549 vol.4. DOI: 10.1109/VETECF.2000.886090.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[8] Constantine A Balanis. *Antenna theory: analysis and design*. Wiley-Interscience, 2005.

[9] Felix Miranda Dan Mandl Mary Ann Ingram et al. "Hybrid Ground Phased Array Antennas for Low Earth Orbiting Satellites". In: *ESTO AIST* (2005).

[10] Mary Weitnauer et al. "LEO Download Capacity Analysis for a Network of Adaptive Array Ground Stations". In: (Feb. 2005).

[11] E. Mezura-Montes, J. Velazquez-Reyes, and C.A. Coello Coello. "Modified Differential Evolution for Constrained Optimization". In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 25–32. DOI: 10.1109/CEC.2006.1688286.

[12] Sebastien Rondineau et al. "Ground Stations of Arrays to Increase the LEO Download Capacity". In: *2006 European Microwave Conference*. 2006, pp. 874–877. DOI: 10.1109/EUMC.2006.281059.

[13] Steven Diamond and Stephen Boyd. "CVXPY: A Python-embedded modeling language for convex optimization". In: *Journal of Machine Learning Research* 17.83 (2016), pp. 1–5.

[14] Benjamin Fuchs and Sébastien Rondineau. “Array Pattern Synthesis With Excitation Control via Norm Minimization”. In: *IEEE Transactions on Antennas and Propagation* 64.10 (2016), pp. 4228–4234. DOI: 10.1109/TAP.2016.2594300.

[15] Sophocles J. Orfanidis. *Electromagnetic Waves and Antennas*. 2016. URL: https://eceweb1.rutgers.edu/~orfanidi/ewa//.

[16] Jiangbin Lyu et al. “Placement Optimization of UAV-Mounted Mobile Base Stations”. In: *IEEE Communications Letters* 21.3 (2017), pp. 604–607. DOI: 10.1109/LCOMM.2016.2633248.

[17] Akshay Agrawal et al. “A rewriting system for convex optimization problems”. In: *Journal of Control and Decision* 5.1 (2018), pp. 42–60.

[18] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[19] Rafael Rodrigues Luz Benevides. *Modelling and Identification of Time-domain Electromagnetic Fields of an Antenna*. Master’s thesis. Brasília, DF, 2022.

[20] Vítor Lima Aguirra. *Desenvolvimento teórico de família de antenas dipolo e Yagi-Uda e ferramentas computacionais para projeto de arranjos*. Monography. Brasília, DF, 2023.

[21] Yushun Xia, Yujun Luo, and Yuxuan Feng. “Site planning for 5G communication base stations based on the idea of binary mask”. In: *Third International Symposium on Computer Engineering and Intelligent Communications (ISCEIC 2022)*. Ed. by Xianye Ben. Vol. 12462. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Feb. 2023, 1246202, p. 1246202. DOI: 10.1117/12.2660822.

[22] IBGE. *Bases cartográficas continuas - Brasil*. URL: https://www.ibge.gov.br/geociencias/cartas-e-mapas/bases-cartograficas-continuas/15759-brasil.html?=&t=acesso-ao-produto.

[23] *Laboratório Compartilhado de Ensino e Pesquisa em Telecomunicações*. URL: https://lab-telecom.unb.br/.

[24] SciPy. *SciPy documentation*. URL: https://docs.scipy.org/doc/scipy/reference/optimize.minimize-slsqp.html#optimize-minimize-slsqp.