UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Master's degree in

BIOMEDICAL ENGINEERING

Optimization of Metasurface for Mobile Phone Applications in Reverberant Enviroment

Supervisor:

Prof.Franco Moglie

Author:

Nicola Di Viesti

Co Supervisor:

Prof.Valter Mariani Primiani

ANNO ACCADEMICO 2021/2022

A papà e mamma,
i pilastri della mia vita

Indice

Capitolo 1

INTRODUCTION

1.1 Metamaterial

Metamaterials are one of the most intriguing and fascinating study fields in the nanotechnology business right now. The electromagnetic characteristics that may be acquired with these materials are non-existent in nature, and as a result, they have sparked international interest from a variety of perspectives. Solid state physics explains many of the macroscopic properties of materials through a direct link with their microscopic properties. The electromagnetic properties, for example, are heavily influenced by two factors: the chemical properties of the individual atoms that comprise the material and how they are organized in space. Metamaterials are laboratory-created materials made out of basic cells that replace typical materials' molecules and atoms. Geometry and spatial order in them manage to affect the interaction with electromagnetic waves ranging from microwaves to visible and infrared, giving them distinct properties. [9]

1.1.1 Metamaterials and Maxwell's equations

Figura 1.1: *James Clerk Maxwell*

The electromagnetic characteristics of materials are characterized by two fundamental variables using Maxwell's equations: electric permittivity and magnetic permeability. These are material specific quantities that are, in general, tensor quantities. Permittivity explains how an electric field and a substance interact physically, i.e. how much the material polarizes when it interacts with an electric field. The same holds true for permeability, where the interaction field is magnetic:

$$\mathbf{P} = \epsilon_0(\epsilon_r - 1)\mathbf{E} \tag{1.1}$$

$$\mathbf{M} = (\mu_r - 1)\mathbf{H} \tag{1.2}$$

where $\mathbf{P}$ and $\mathbf{M}$ are the dielectric and magnetic polarizability, respectively, $\mathbf{E}$ and $\mathbf{H}$ are the electric and magnetic fields,

$$\epsilon_0 \tag{1.3}$$

$$\mu_0 \tag{1.4}$$

$$\epsilon_r \tag{1.5}$$

$$\mu_r \tag{1.6}$$

are the permittivity and permeability of the vacuum and the relative ones of the medium, respectively. Simplifying with respect to the vacuum constants, the characteristic quantities of the materials are defined

$$\epsilon = \epsilon_r\epsilon_0 \tag{1.7}$$

$$\mu = \mu_r\mu_0 \tag{1.8}$$

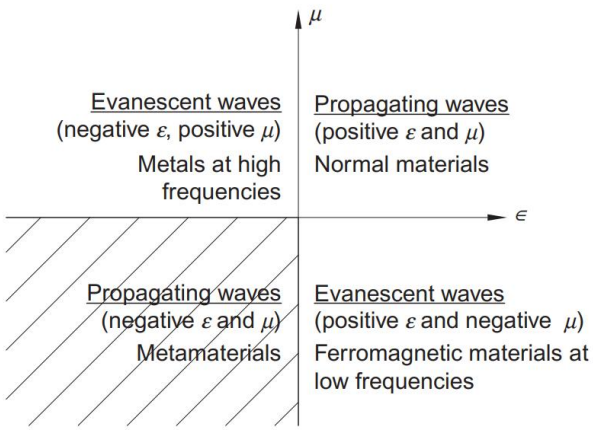

Figura 1.2: *Division of all materials existing in nature in base of their permittivity and permeability*

From a macroscopic standpoint, it is feasible to segment all natural materials based on the values epsilon and mu, as illustrated in Figure 1. Materials having ϵ and μ, both positive, are found in area 1. This is the most typical instance, and it includes several dielectric materials. Negative permittivity materials are found in the second quadrant. This occurs in metals, doped semiconductors, and ferroelectric materials, at least in specific wavelength ranges below the plasma frequency. Region 4 contains ferrite-based materials with negative permeability but magnetic activity that decays beyond the microwave range. The third quadrant, which contains materials with both negative emissivity and permeability, is totally empty, unlike the others: there are no such compounds in nature. Perhaps because of their unusual non-existence, the materials of the third quadrant have sparked early theoretical interest in predicting and evaluating their potential electromagnetic characteristics [9].

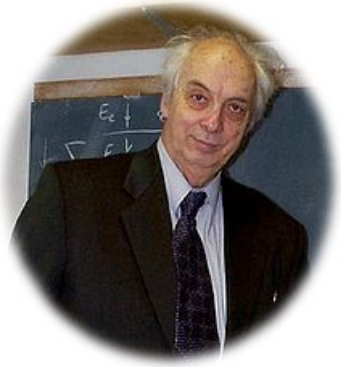

Figura 1.3: *Victor Veselago*

Victor Veselago, a Russian theoretical physicist, actually examined this phenomenon's potential features in 1968. Think at, for instance, how a monochromatic plane wave moves across such a medium. Typically, the wave's electric and magnetic components may be expressed as:

$$\mathbf{E}(\omega,\mathbf{k}) = E_0 e^{i\mathbf{kr}-i\omega t} \tag{1.9}$$

$$\mathbf{H}(\omega,\mathbf{k}) = H_0 e^{i\mathbf{kr}-i\omega t} \tag{1.10}$$

where ω is the frequency and $\mathbf{k}$ is the wave vector.
The Maxwell equations in local form are:

$$\nabla \cdot \mathbf{B} = 0 \tag{1.11}$$

$$\nabla \cdot \mathbf{D} = \rho \tag{1.12}$$

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \tag{1.13}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1.14}$$

where $\mathbf{D}=\epsilon\mathbf{E}=\epsilon_0\epsilon_r\mathbf{E}$ and $\mathbf{B}=\mu\mathbf{H}=\mu_0\ \mu_r\mathbf{H}$ are are the fields of electric and magnetic induction. If we consider the case in which there are neither free charges (ρ) nor currents ($\mathbf{J}$), the equations are simplified:

$$\nabla \times (E_0 e^{i\mathbf{kr}-i\omega t}) = -\frac{\partial}{\partial t}(\mu H_0 e^{i\mathbf{kr}-i\omega t}) = i\mathbf{k}\times\mathbf{E} = i\omega\mu\mathbf{H} \tag{1.15}$$

$$\nabla \times (H_0 e^{i\mathbf{kr}-i\omega t}) = -\frac{\partial}{\partial t}(\epsilon E_0 e^{i\mathbf{kr}-i\omega t}) = i\mathbf{k}\times\mathbf{H} = -i\omega\epsilon\mathbf{E} \tag{1.16}$$

Thus, the following system of equations is obtained:

$$\begin{cases} \mathbf{k}\times\mathbf{E} = \mu\omega\mathbf{H} \\ \mathbf{k}\times\mathbf{H} = -\varepsilon\omega\mathbf{E} \end{cases} \tag{1.17}$$

From these equations it follows that the vectors k, E and H form a triad of right-handed vectors when a plane wave propagates in a dielectric medium with μ and positive. Conversely, if μ and ϵ are both negative the triplet is left-handed and the Poynting vector, defined as $S = E$ X H, is antiparallel to the wave vector k. Furthermore, for the conservation of causality, the refractive index of these materials, defined as

$$n = \pm\sqrt{|\varepsilon_r|\,|\mu_r|} \tag{1.18}$$

takes on a negative sign. Because of these last two properties, materials with simultaneously negative μ and ϵ are called left-handed materials or materials with a negative refractive index [17]. As described by Veselago in his work, these materials would have properties that are completely out of the ordinary as we know them. When an electromagnetic wave propagates from a medium with refractive index n_1 to one with refractive index n_2, its direction of propagation θ_1 with respect to the normal is deflected along the θ_2 direction defined by Snell's law:

$$n_1 \operatorname{sen}\theta_1 = n_2 \operatorname{sen}\theta_2 \tag{1.19}$$

What happens if, on the other hand, light propagates passing from a medium with a positive refractive index to one with a negative refractive index? In this case, counterintuitively and absolutely non-existent in nature, the entrance and exit angles must have opposite sines to continue to satisfy Snell's law, i.e. the refracted light will propagate at negative angles, on the same side as the incident light, as schematically indicated in 1.4

The first experimental realization of these materials' properties, after Veselago had theoretically described them in 1968, happened in 1996 thanks to the work of English physicist John Pendry, who developed a system that could circumvent natural limitations and thus produce the first metamaterial. The concept is very straightforward in theory. In fact, by starting with common materials and arranging them into single elements (known as meta atoms) that are distributed periodically or randomly and have dimensions and spacings much smaller than the wavelength of electromagnetic radiation with which they interact, it is possible to recreate in the

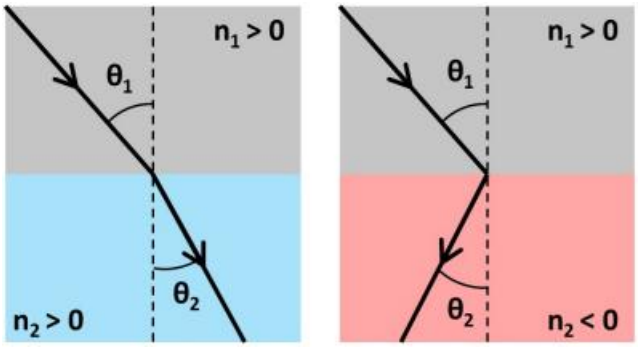

Figura 1.4: Refraction of light in materials with positive (left) and negative (right) refractive index.

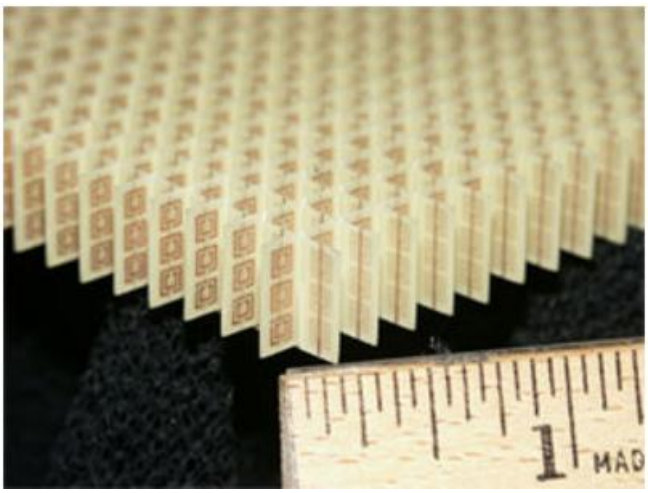

Figura 1.5: First example of metamaterial proposed by Pendry.

laboratory materials with optical properties that can be engineered at will. Since each meta-atom's microscopic characteristics are thus rendered "invisible" to electromagnetic radiation, the behavior of all the meta-atoms as a whole has an impact on the material's response. In other words, from the perspective of the macroscopic response, the inhomogeneous set of meta-atoms can be described as a homogeneous medium with effective electric permittivity and magnetic permeability $\epsilon_{r,eff}$ and $\mu_{r,eff}$.

Pendry developed a set of split ring resonators distributed as arrays (antennas in the form of rings not completely closed) that showed negative magnetic permeability after first demonstrating the properties of a periodic array of copper wires with specific dimensions that displayed an electromagnetic response with negative electric permittivity. Finally, the two structures were combined to create the first metamaterial, a structure that simultaneously had μ_r 0 and ϵ_r 0 in the microwave spectral area [3].

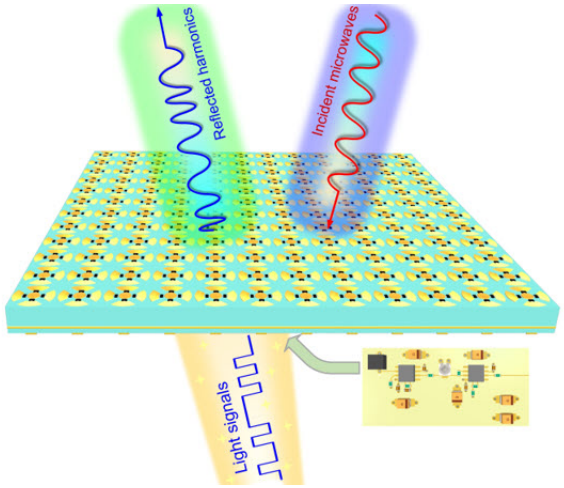

Figura 1.6: Metasurface.

1.2 Metasurface

Metasurfaces are an innovative technology that has attracted a lot of attention in recent years due to their outstanding properties and numerous applications. These are artificial surfaces made up of a series of microscopic structures called metaspheres which are arranged in a precise and controlled way to produce specific effects. Thanks to their highly flexible and controllable properties, metasurfaces can be used to fabricate never-before-thought-of devices, paving the way for a huge range of new applications. Metasurfaces have been studied in diverse fields, from communication technology to medicine, from energy production to national security. Designing and fabricating metasurfaces is a significant technological challenge. However, in recent years, new methods for designing and fabricating metasurfaces have been developed, resulting in highly flexible and controllable surfaces with optical properties. These advances pave the way for new applications and potential revolutions in many fields. One of the most important benefits of metasurfaces is their ability to manipulate light in ways that are not possible with natural surfaces. Metasurfaces can be engineered to have highly flexible and controllable optical properties, such as the ability to selectively reflect, absorb, or emit light, or control the direction and polarization of light. These properties make metasurfaces very useful in a wide range of applications, from communication technology to medicine. Metasurfaces can also be used to create devices that control the direction and polarization of light, such as polarizers. These devices are essential for many imaging and communication applications and can be used to improve image quality and data transfer rates. Furthermore, metasurfaces can be used to create materials with extraordinary properties, such as the ability to become invisible to radar or to selectively absorb light and heat. These properties can be used in many fields, from military defense to the production of environmentally sustainable buildings.

1.2.1 Reconfigurable Intelligent Surface (RIS)

Reconfigurable intelligent surfaces, also known as intelligent reflecting surfaces or huge intelligent surfaces, are made up of a variety of reflecting features that allow the incident signals to be changed. The capacity of RISs to proactively alter the wireless communication environment has made them a focus of research in wireless communications to address a variety of issues that arise in various wireless networks. [8]

RIS's benefits

The following is a list of RIS's benefits:

- **Simple to use:** RISs are electromagnetic (EM)-based, virtually passive devices. Due to their inexpensive cost, RISs can be installed on a variety of buildings, including but not limited to car windows, internal walls, aerial platforms, roadside billboards, highway polls, and pedestrians' clothing, as shown in Fig. 1.7;

- **Spectral efficiency improvement:** By making up for power loss over long distances, RISs are able to change the wireless propagation environment.By passively reflecting the radio signals that are impinging, base stations (BSs) and mobile users can create virtual line-of-sight (LoS) relationships. When barriers, like as tall buildings, hinder the LoS link between BSs and users, the throughput improvement becomes important. A software-defined wireless environment may be built as a result of the intelligent deployment and design of RISs, which has the potential to improve the received signal-to-interference-plus-noise ratio (SINR);

- **Environmentally friendly:** RISs can shape the incoming signal by adjusting the phase shift of each reflecting element instead of using a power amplifier, in contrast to standard relaying systems like amplify-and-forward (AF) and decode-and-forward (DF). Deploying RISs is therefore more eco-friendly and energy-efficient than using traditional AF and DF systems.

- **Compatibility:** RISs support full-duplex (FD) and fullband transmission because they only reflect EM waves, which makes them compatible. Furthermore, RIS-enhanced wireless networks are hardware and standard-compliant with current wireless networks.

Applications of RISs in various wireless communication networks are shown in Fig. 1.7. RIS-enhanced cellular networks are shown in Fig. 1.7(a), where RISs are used to get over barriers that stand in the way of BSs and users. As a result, mobile edge computing (MEC) networks' latency performance and quality of service (QoS) in heterogeneous networks are both enhanced. By reducing interference in device-to-device (D2D) communication networks, RISs can facilitate huge connectivity on

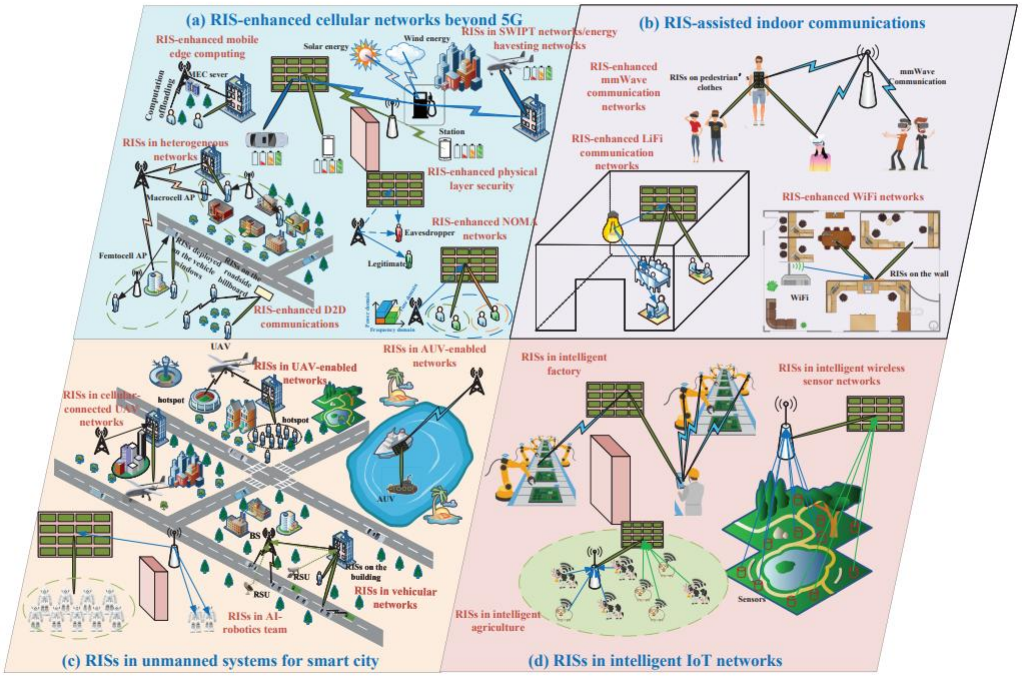

Figura 1.7: RISs in wireless communication networks.

the other hand. They can also cancel unwanted signals by cleverly structuring passive beamforming in the context of physical layer security (PLS). Additionally, RISs can be used to increase the strength of the signal that mobile phone customers receive, as well as to reduce interference from neighboring cells and power loss over long distances can be compensated in simultaneous wireless information and power transfer (SWIPT) networks.

Figure 1.7(b) shows an illustration of RIS-assisted indoor communications, where RISs may be installed on walls to improve the QoS in some rate-hungry interior scenarios, such as virtual reality (VR) applications. Additionally, a concatenated virtual RIS-aided LoS link between the access points (APs) and the users can be created with the help of RISs in order to ensure that some block-sensitive scenarios, such as visible light communications and wireless fidelity (WiFi) networks, have no blind spots in the coverage area. This means that both the propagation links between the APs and the RISs as well as between the RISs and the users can be in LoS.

Unmanned systems that have been upgraded by RIS are shown in Fig. 1.7(c). By fully utilizing the aforementioned RIS advantages, RISs can be used to improve the performance of unmanned aerial vehicle (UAV) enabled wireless networks, cellular-connected UAV networks, autonomous vehicular networks, autonomous underwater vehicle (AUV) networks, and intelligent robotic networks. To create concatenated virtual LoS connections between the UAVs and the users in RIS augmented UAV-

aided wireless networks, for instance, one can modify the phase shifts of the RISs rather than directing the movement of the UAVs. In order to decrease movement manipulations and energy consumption of UAVs, the concatenated virtual LoS linkages cannot be built even with the help of RISs for the UAVs to keep the hovering position.
IoT networks that are RIS-enhanced are shown in Fig. 1.7(d), where RISs are used to support intelligent wireless sensor networks, intelligent agriculture, and intelligent factories.

Different Categories of RISs

Since RISs have a specific structure, they may be implemented utilizing metamaterial or patch-array based technology. Metasurfaces are RISs that are based on metamaterial. When deployed at various places, RISs may be made to function as waveguide surfaces at the BS or as reflecting/refracting surfaces between the base station and the user. Considering the tuning mechanisms, RISs can be thermally, mechanically, or electrically changed. RISs can be characterized as passive-lossy, passive-lossless, or active according on how much energy they use. The active or passive nature of RISs determines their ultimate performance capabilities.

- **Waveguide RIS:** R. Smith et al. [2] offered a theoretical investigation of waveguide-fed metasurfaces in their article on waveguide RIS. Modeled as uncoupled magnetic dipoles are the constituents of the metasurface. Each dipole element's amplitude and polarizability are inversely correlated with the product of the reference wave. The beamforming function of the metasurface antenna is adjusted by polarizability. Every component of the metasurface functions as a tiny antenna. The compact waveguide metasurface can transmit at broader angles and takes up less room than traditional antenna arrays.

- **Refracting RIS:** Viktar S. and colleagues [16] put up a theoretical concept for properly reflecting and refracting metasurfaces. To properly optimize the tangential field components at the two sides of the metasurface, the authors developed an equivalent impedance matrix model. Also covered are self-oscillating teleportation metasurfaces, non-local metasurfaces, and metasurfaces made entirely of lossless components. The use of omega-type bianisotropy in creating realizations of fully reflective surfaces with lossless components is described.

- **Reflecting RIS:** A digital coding reflecting metasurface was created by Dai et al. [6] Varactor diodes with a configurable biasing voltage are present in the metasurface's constituent components. Each element may perform discrete phase shifts and accomplish beamforming for the reflected wave by pre-designing a number of digital biasing voltage levels. The operational guidelines for RISs that serve as reflectors are the main topic of the remaining paragraphs in this section.

In a wireless communication environment, a delivered radio signal frequently collides with a number of objects, producing reflected, diffracted, and dispersed duplicates of the original signal. These duplicates, also known as multipath components, have different magnitudes, phases, and delays that, when put together in both beneficial and detrimental ways, greatly amplify the received signal's distortion. This phenomenon, known as fading in wireless communications, poses a substantial obstacle to the advancement of both present and future wireless communication systems. RISs are used largely to create a controllable radio environment in which the very chaotic wireless channel is converted into a predictable space by painstakingly re-engineering the propagation of the EM waves in a software-controlled way.

MODEL OF THE CONVENTIONAL TWO-RAY SYSTEM

The received signal in this paradigm is made up of two rays: the line-of-sight (LOS) ray and the ray that is reflected off the ground. We assume that the ground-plane is sufficiently big in respect to the transmission wavelength and solely produces specular reflections. The Fermat's principle, which asserts that the path a ray takes between two places is the path that is travelled in the shortest amount of time, is followed in the propagation of radio waves when they are modelled as rays. The point of reflection *G* and the transmit and receive antennas are separated by a distance indicated by *r1* and *r2*, respectively, while the distance between the transmit and receive antennas is indicated by the letter *l*. Considering *x(t)* the complex baseband transmitted signal and τ the relative time delay between the ray reflected from the ground and the LOS path, which is given by:

$$\tau = \frac{(r_1 + r_2 - 1)}{c} \tag{1.20}$$

with c being the speed of light, we have:

$$\mathbf{x(t)} \approx x(t - \tau) \tag{1.21}$$

The received (noise-free) baseband signal may then be summarized as follows:

$$r(t) = \frac{\lambda}{4\pi}\left(\frac{e^{-\frac{j2\pi l}{\lambda}}}{l} + \frac{R \times e^{-\frac{j2\pi(r_1+r_2)}{\lambda}}}{r_1 + r_2}\right) x(t) \tag{1.22}$$

The LOS and ground-reflected signals, which have phase delays of :

$$2\pi l/\lambda 2\pi(r_1 + r_2)/\lambda_1 \tag{1.23}$$

The received signal is created by adding these, which are proportional to propagation distances. Assuming that the transmit power of x(t) is Pt, the following definition of the received power Pr may be made in terms of Pt:

$$P_r = P_t\left(\frac{\lambda}{4\pi}\right)^2 \left|\frac{1}{l} + \frac{R \times e^{-j\Delta\phi}}{r_1 + r_2}\right|^2 \tag{1.24}$$

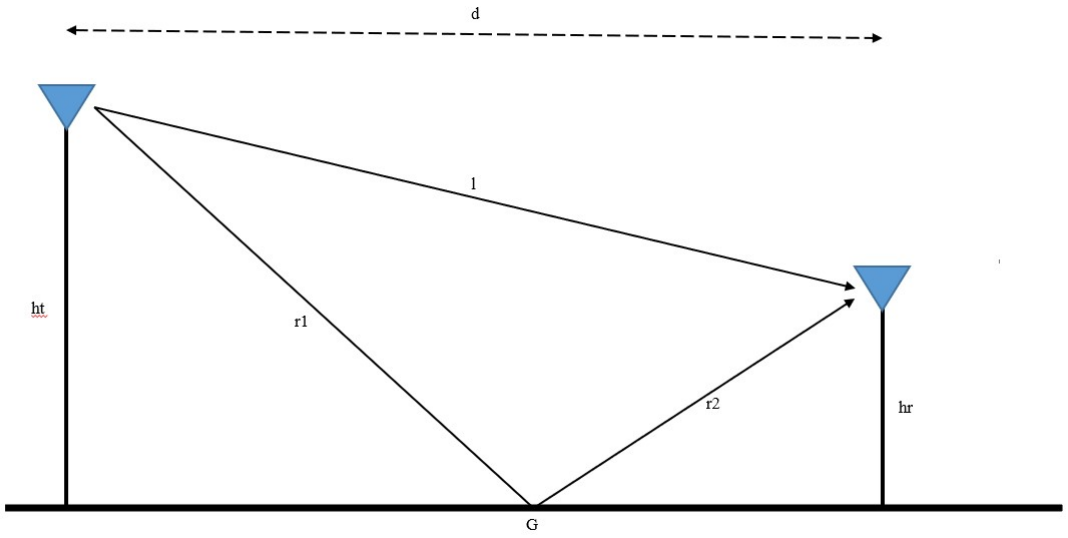

Figura 1.8: Two ray system model.

Where $\Delta\phi$:

$$\Delta\phi = \frac{2\pi(r_1 + r_2 - l)}{\lambda} \tag{1.25}$$

is the distinction in phases between the two pathways. Assuming that the distance d is large enough, considering d much higher than $ht + hr$, then we obtain $d \approx 1 \approx r1 + r2$ and $R \approx -1$ for a specular reflection from the ground. The previous equation can be reduced into:

$$P_r \propto P_t \left(\frac{1}{d^2}\right)^2 \tag{1.26}$$

This indicates that the received power degrades by a factor of d/4. The received power resulting only from LOS propagation decays with the second power of the distance in the absence of ground reflection since the second component of the preceding equation would not be present:

$$P_r = P_t \left(\frac{\lambda}{4\pi d}\right)^2 \tag{1.27}$$

Is it feasible to plainly discern the destructive impact that the uncontrolled reflection off the ground has on the received signal's strength because of the two paths' mismatched phases? A single unanticipated reflection from the ground can drastically lower the received signal strength, even in the most optimistic transmission scenario with no user movement and no random environmental affects. [4]

MODEL OF A TWO-RAY SYSTEM WITH ONE RECONFIGURABLE METASURFACE

Let's examine the same system model with the addition that a repositionable meta-surface has been laid out on the surface to aid in communication between the transmitter and receiver. To be more precise, we focus on the simplest case study where the meta-surface acts as a reflecting surface and can modify the reflected ray's direction. We assume that the ground is covered by a reconfigurable meta-surface. To guarantee that the LOS and reflected rays sum up coherently and provide the brightest signal feasible, the reconfigurable meta-surface may be conceptualized as a perfect phase shifter that can adjust the reflected wave's phase. In order to optimally maximize the phase of the reflected ray, the following findings would be achieved if we assumed that the reconfigurable meta surface is capable of coherently synchronizing the phases of the direct and reflected ray at any angles of incidence and reflection:

$$P_r = P_t \left(\frac{\lambda}{4\pi}\right)^2 \left|\frac{1}{l} + \frac{1}{r_1 + r_2}\right|^2 \approx 4P_t \left(\frac{\lambda}{4\pi d}\right)^2 \tag{1.28}$$

We show that the scaling rule that governs how much power is received as a function of distance can be changed by using reconfigurable meta-surfaces: the received power now only decays with the second power of distance, which is the same as the LOS ray, instead of the fourth. [4]

1.3 FDTD

The Finite-Difference Time-Domain (FDTD), commonly known as computational electrodynamics, is a numerical analytic approach used to simulate how electromagnetic fields interact with actual physical objects and equipment in a particular environment. Numerous electromagnetic modeling issues have found applications for the finite-difference time-domain (FDTD) approach. It was first employed to model the interactions with a single signal, often a sinusoid, because it is a time-domain approach. With just one computer run, different frequency information may now be obtained thanks to the creation of the running Fourier Transform. However, since the materials are frequency dependent and the original FDTD paradigm only recognized constant values of conductivity and dielectric constant, the utility of this approach for biological and other sorts of applications has been limited. The creation of the frequency dependent FDTD method has been one of the most important developments in the use of the FDTD method. This has made a wide range of simulations of EM interactions with different kinds of materials possible [13]

1.3.1 History of FDTD

In the case of complex and asymmetrical geometries, Maxwell's equations did not have accurate analytical solutions until 1960, despite their formulation in 1873. Sin-

ce 1960, as computer technology has advanced, academics have been exploring the use of numerical approaches to get beyond the drawbacks of traditional ones, particularly for the precise characterisation of complicated geometry. The finite element method (FEM), which breaks a complex system into smaller, simpler components known as finite elements, is a commonly used numerical technique for solving partial differential equations in two or three spatial variables. By dividing the entire domain into smaller chunks, it is possible to accurately describe complicated geometries, inhomogeneous materials, and the overall solution. An exact, full-wave numerical method called the Method of Moments (MoM) is used to solve open boundary electromagnetic issues. The MoM is an integral equation methodology that solves Maxwell's equations in their integral form rather than their differential form, which is what the finite element techniques do. In order to solve Maxwell's equations, the Finite Difference Time Domain (FDTD) approach uses the finite difference technique. Space is organized into discrete units called cells in FDTD. Each cell is allocated points on its surface, and each of those points must meet the Maxwell equations. In this manner, electromagnetic waves are virtually replicated as they would in the actual world, propagating into a numerical space. Kana Yee had initially introduced the FDTD approach in 1966, and Allen Taflove had come up with the moniker and abbreviation "FDTD" in 1980. It uses finite differences as an approximation to both the spatial and temporal derivatives that appear in Maxwell's equations to solve electromagnetic field problems in a variety of applications and fields of work, from the design of antennas to bio-photonics. This technique is regarded as the simplest (both in terms of implementation and concept) of the full wave numerical techniques. The sole drawback is that it is known to be computationally costly; in fact, solutions may necessitate a significant amount of memory and calculation time. Nonetheless, it can handle challenging issues and investigate electromagnetic phenomena at radio and microwave frequencies. When comparing FDTD's computational efficiency for big problems to other approaches, its benefits may be summed up as its capacity to operate in a variety of contexts, devices, and frequencies. [11] [5] [7]

1.3.2 FDTD Method: YEE'S ALGORITHM

A time-dependent problem with solutions With a few exceptions, Maxwell's equations are unknown in their general form. The application of the boundary requirements is the major cause of the issue. In this work, LT,7e will demonstrate how to numerically arrive at the solution when the boundary condition is suitable for a perfect conductor. The most general instance can theoretically use this numerical assault. Although numerical solutions to a scattering issue for which the ratio of the typical linear dimension of the barrier to the m-avelength is high still appear to be impossible due to the restricted memory capacity of current digital computers.

THE EQUIVALENT SET OF THE FINITES DIFFERENCE EQUATION AND MAXWELL'S EQUATION

Maxwell's equations in an isotropic medium are:

$$\frac{\partial B}{\partial t} + \nabla \times E = 0, \tag{1.29}$$

$$\frac{\partial D}{\partial t} - \nabla \times H = J, \tag{1.30}$$

$$B = \mu H, \tag{1.31}$$

$$D = \epsilon E, \tag{1.32}$$

Whre J, μ and ϵ are assumed to be given functions of space and time. In a rectangular coordinate system, (1.29) and (1.30) are equivalent to the following system of scalar equations:

$$-\frac{\partial B_x}{\partial t} = \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z}, \tag{1.33}$$

$$-\frac{\partial B_y}{\partial t} = \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x}, \tag{1.34}$$

$$\frac{\partial B_z}{\partial t} = \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}, \tag{1.35}$$

$$\frac{\partial D_x}{\partial t} = \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - J_x, \tag{1.36}$$

$$\frac{\partial D_v}{\partial t} = \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - J_y, \tag{1.37}$$

$$\frac{\partial D_x}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - J_z, \tag{1.38}$$

A set of finite difference equations for (1.33)-(1.38) that will be found convenient for perfectly conducting boundary condition is as follows. For (1.33) we have:

$$\begin{aligned} &\frac{B_{x^{n+1/2}}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right) - B_{x^{n-1/2}}\left(i, j+\frac{1}{2}, k+\frac{1}{2}\right)}{\Delta t} \\ &= \frac{{E_y}^n\left(i, j+\frac{1}{2}, k+1\right) - {E_y}^n\left(i, j+\frac{1}{2}, k\right)}{\Delta z} \\ &- \frac{{E_z}^n\left(i, j+1, k+\frac{1}{2}\right) - {E_z}^n\left(i, j, k+\frac{1}{2}\right)}{\Delta y} \end{aligned} \tag{1.39}$$

Similar constructions may be made for the finite difference equations corresponding to (1.34) and (1.35), respectively. For (1.36), there is:

$$\begin{aligned}
&\frac{{D_x}^n\left(i+\frac{1}{2},j,k\right)-D_x^{n-1}\left(i+\frac{1}{2},j,k\right)}{\Delta t}\\
&=\frac{{H_z}^{n-1/2}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)-{H_z}^{n-1/2}\left(i+\frac{1}{2},j-\frac{1}{2},k\right)}{\Delta y}\\
&-\frac{{H_y}^{n-1/2}\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-{H_y}^{n-1/2}\left(i+\frac{1}{2},j,k-\frac{1}{2}\right)}{\Delta z}\\
&+{J_x}^{n-1/2}\left(i+\frac{1}{2},j,k\right).
\end{aligned} \tag{1.40}$$

Similar constructions may be made for the equations corresponding to (1.37) and (1.38), respectively.

BOUNDARY CONDITIONS

When computationally simulating wave propagation processes in an infinite domain, it is standard practice to truncate the computational domain to a limited domain. The newly constructed exterior border is somewhat artificial in order to minimize undesired numerical consequences. Due to reflections, special handling is necessary at these borders. Once the electric and magnetic fields on the exterior surface of the region of interest are known, it is possible to utilize appropriate boundary conditions on the domain's outside perimeter when using FDTD to extend the solutions obtained using the Yee method to an infinite domain. The theoretical underpinnings of RBCs, or Radiation Boundary Conditions, and ABCs, or Absorbing Boundary Conditions, two types of boundary conditions, are different. The ABCs solutions are the ones that are most frequently used with the FDTD technique, despite the fact that both create errors and false reflections. In 1981, Gerrit Mur was the first to discuss the need for suitable boundary conditions. For methods that employ a spatial grid to solve Maxwell's equations, Mur's solution is the first workable response to the boundary condition issue. Because of its simplicity and computing economy, it is a useful option for situations that do not need for a high accuracy solution. All of the ABCs results from Mur and other authors whose studies were published before 1994 produce effective outer-boundary reflection coefficients between -35 and -45 dB, notwithstanding the need for simulations. For the best dynamic range, the effective reflection coefficient must be reduced by 40 dB [18]. Additionally, when an oblique wave is incident, they are constrained because, specifically, oblique waves cannot be absorbed in the first order Mur's solution; as a result, they are reflected within the boundary. For the Maxwell's equations, Berenger developed the perfectly matched layer (PML), which serves as a workaround for these limitations. It entails breaking down the electric or magnetic field components in the absorbing boundary area into a subcomponent that can be properly absorbed by the material of the perfectly

matched layer. As a result, a nonphysical absorbing medium with a wave impedance independent of the angle of incidence and frequency of outgoing dispersed waves is produced in close proximity to the outer FDTD mesh border. Because of this, when a wave crosses the limit of the region of interest, it is exponentially attenuated rather than reflected. The maximum dynamic range may be expanded with this technique to more than 80 dB [14]. The PML ABC is the only ABC with the unique combination of broadband efficacy, resilience, and computing efficiency. It is crucial to keep in mind that the absorbing boundary condition is a mathematical technique used at the 26 artificial numerical limits of a computational domain to reduce or completely remove false reflections that appear in wave propagation simulations. This condition has no real-world analogue. [12] The FDTD algorithm is based on an orthogonal, regular Cartesian lattice, which has variable and unstructured meshing. Due to the orthogonality and uniform spacing of the grid, the first-order derivatives of Maxwell's equations may be approximated using central difference operators. This results in a second-order precise solution in space and time as well as a discrete approximation for the fields based on a uniform orthogonal lattice. On the other hand, structures with fine geometrical details may struggle to cling to the uniform lattice's edges. When the field interaction is highly dependent on the form of the boundary and the grid does not adjust to the geometry of the boundary, as is the case with a curved or flat boundary, a large calculation error may result. Because of this, rather of applying the boundary constraints directly to the boundary, they must be applied to an auxiliary boundary, which is a staircase approximation of the actual boundary. It is occasionally essential to use non-uniform grids and lower the actual cell size in order to adequately represent the local fields. In 1991, Sheen introduced a quasi-uniform grid FDTD approach. Although the approach is restricted to particular geometries that fit into this specialized grid, the nonuniform FDTD algorithm provides a strong and flexible tool for intricate and highly complex circuits with rectangular geometry [15]. Materials that are lossy, dispersive, nonlinear, and gain: Controlling or processing short electromagnetic pulses requires an understanding of the nature of pulse interactions with materials over vast bandwidths. Gain, nonlinearity, and material dispersion are crucial elements in short-pulse physics [18]. Linear dispersion is the frequency-dependent fluctuation of a material's permittivity and/or permeability at low electromagnetic wave intensities. Nonlinearity: When an electromagnetic wave interaction is strong locally, especially at high intensities, a material's dielectric permittivity and/or permeability will change. Nonlinear dispersion: The intensity of a material's nonlinear characteristics depends on the interaction electromagnetic wave's sinusoidal frequency content. Instead of an exponential loss, as in conventional materials, gain: offers an exponential increase in the interaction wave with propagation distance in the material. Gain can be frequencydispersive and nonlinear, however it typically depends on the frequency and power of the interacting waves [18][15].

1.3.3 FDTD: Overlay of Plane Waves

This section explains how plane waves with evenly distributed random incidence are generated. The computation of incident fields and the separation plane between the total field region and the dispersed field region are introduced. A validation of the approach mimicking an empty volume is presented at the conclusion.

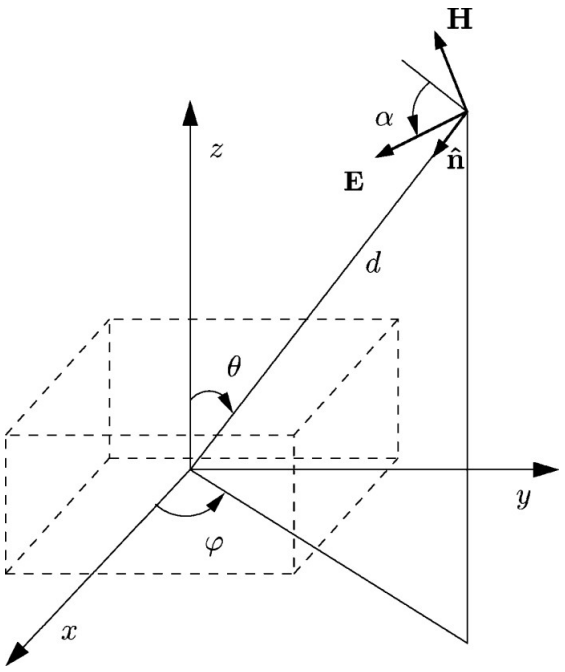

Figura 1.9: *An incident random plane wave's geometry on the computing volume.*

Generation of the Random Plane Wave

The incident fields added to or subtracted from the FDTD fields on the separation plane are obtained as a superposition of random plane waves. The parameters of each plane wave are generated in the step of FDTD initialization, and they are stored and recalled for each FDTD temporal iteration. Referring to Fig. 1.9, the generated parameters are the angles ϕ and θ, the distance d, and the polarization α. The angles ϕ and θ can be linked to a point on the surface of a sphere. The uniform distribution on the spherical surface is used to extract the points. In fact, the incidence $\hat{n}$ can come from any direction with the same probability. Fig. 1.10 shows the sphere and a randomly generated point (P). If the values θ and ϕ are extracted independently, the resulting distribution is not uniform, and the extracted points thicken around the poles. [10] In order to obtain a uniform distribution of the points on the spherical surface, we generate the angle θ and the angle ϕ as described in the following steps.

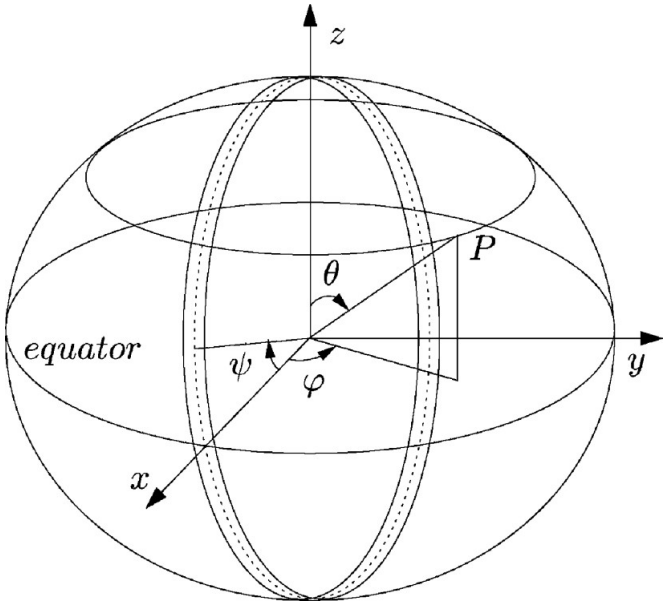

Figura 1.10: *The surface of a sphere is used to symbolize the incidence of a plane wave that is created randomly. One may compare it to the Earth with its "equator" and "parallels."*

1. In the range [0,π], the angle θ is created at random with a uniform probability. In the range [0,2π], an auxiliary angle ϕ is created at random with uniform probability;

2. If 2πsin(θ)>=ϕ', the angles θ and ϕ=ϕ' /sin(θ) are stored. The plane wave will be used in the FDTD iteration steps;

3. Otherwise, the angles and are disregarded, and step 1 of the procedure is repeated.

In actuality, the angles and are first separately generated. When comparing the sphere to the Earth, the angle corresponds to a "parallel" (see Fig. 1.10). Since its circumference is bigger, the "equator" fits with a value of 90. When the other "parallels"' length disappears, their circumference falls in the direction of the poles (θ=0 and θ=180). The value of is taken and is appropriately drawn if is smaller than the length of the "parallel," otherwise they are not taken into consideration. Assuming the length of the equator is 2. Analyzing the points on a small gap around a circle allows us to verify the sphere's surface is distributed uniformly. The circumference on the *xz*-plane is shown in Fig. 1.10, but numerous additional circumferences parallel to other Cartesian planes have also been examined. We can quickly determine if it is uniform or not by analyzing the probability distribution of these spots on the perimeter while adjusting the angle. After using the suggested processes and extracting the angles and separately, Fig. 1.11 displays the cumulative distribution function (CDF). It is clear that if we want a consistent distribution of the extracted points P on the spherical surface, the angles cannot be chosen

separately. We attempted a variety of additional circumferences that we measured by intercepting planes and spheres, and we discovered that the CDF was consistently uniform. The additional parameters are extracted to have a random plane wave when the angles and are chosen (Fig. 1.9). The distance d, which is connected to the plane wave's phase and is evenly distributed throughout a wavelength, is created at random outside the computational container. The distance d, which is connected to the plane wave's phase and is evenly distributed throughout a wavelength, is created at random outside the computational container. In the range [0,2], the angle, which represents the polarization of the plane wave, is extracted with a uniform distribution. Only the wave magnitude, which is E_0=1 V/m for every plane wave, is constant. [10]

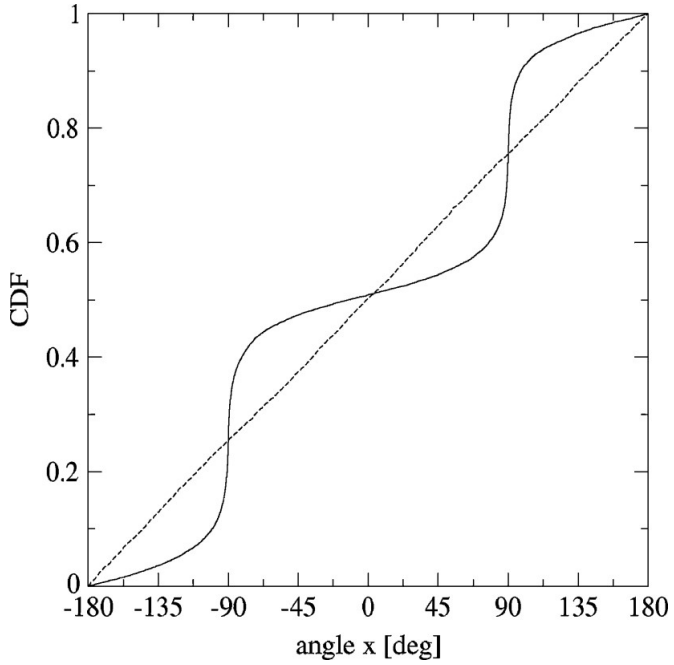

Figura 1.11: *The xz-plane's cumulative distribution of the points around the perimeter. The independent creation of the angle and the angle is referred to as a continuous line. Around the poles (ϕ=90), the extracted points are concentrated. The uniform CDF generated using the mentioned approach is shown by the dashed line.*

Calculations in the Field and Separation Planes

To distinguish between the incident fields and the dispersed fields, perpendicular planes to the three Cartesian axes are provided. Their collective term is "separation plane." The fields are situated inside the whole field region when there are no objects in the working volume. In Fig. 1.12, a horizontal slice of the electric field produced by 100 randomly occurring plane waves is depicted. It is feasible to observe that the incident fields vanish in the exterior region.

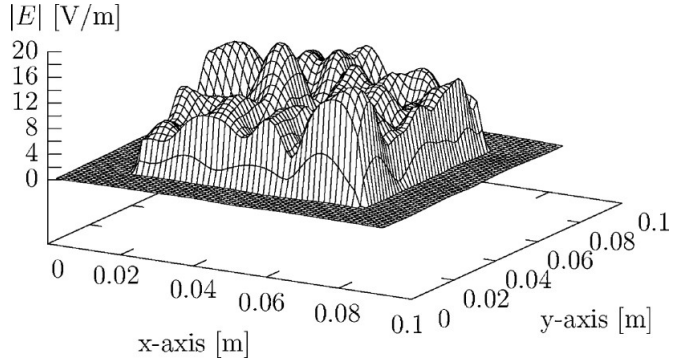

Figura 1.12: *An illustration of the size of the electric field produced by 100 incident random plane waves at 1 GHz on a portion of the working volume.*

The electric and magnetic fields are computed on one of the separation planes by adding or deducting each element from the incident fields. In instance, the incident magnetic field $H(n + 1/2)$inc is added to the magnetic field $H(n + 1/2)$ for each temporal iteration that calculates the electric field $E(n + 1)$. In the opposite direction, the incident electric field Eninc is deducted from the electric field En when calculating the magnetic field. Our version of Yee's method places the origin of the axes in the first cell, the initial elements of the Ex|y|z arrays on the Cartesian axes, and the first elements of the Hx|y|z arrays on the Cartesian planes. As a result, there is a variation along each of the three directions between the start and finish of the FDTD grid. The electric and magnetic fields, which are added or removed at the beginning of the grid, are in a cell prior to the cell in which the field is computed. The fields that are added or deleted are in the same cell at the grid's end. For instance, if we use this notation to write the typical FDTD formulations

$$\begin{aligned}\dot{E}_x\big|_{i,j,k}^{n+1} = E_x|_{i,j,k}^{n} + \frac{\Delta t}{\varepsilon_0 \Delta y}\left[H_z|_{i,j,k}^{n+1/2} - H_z|_{i,j-1,k}^{n+1/2}\right] \\ - \frac{\Delta t}{\varepsilon_0 \Delta z}\left[H_y|_{i,j,k}^{n+1/2} - H_y|_{i,j,k-1}^{n+1/2}\right]\end{aligned} \tag{1.41}$$

The following formulae may be found within the lower separation plane if we represent H(l)|nx,inc as the magnetic field along x axis caused by the l-th plane

wave in the interface surface perpendicular at the z plane:

$$\begin{aligned}
E_x|_{i,j,k}^{n+1} &= \dot{E}_x|_{i,j,k}^{n+1} + \frac{\Delta t}{\varepsilon_0 \Delta z} \sum_{l=1}^{N} H(l)_{y,\text{ inc}} \Big|_{i,j,k-1}^{n+1/2} \\
&\quad - F \frac{\Delta t}{\varepsilon_0 \Delta y} \sum_{l=1}^{N} H(l)_{z,\text{ inc}} \Big|_{i,j-1,k}^{n+1/2} \\
E_y|_{i,j,k}^{n+1} &= \dot{E}_y|_{i,j,k}^{n+1} - \frac{\Delta t}{\varepsilon_0 \Delta z} \sum_{l=1}^{N} H(l)_{x,\text{ inc}} \Big|_{i,j,k-1}^{n+1/2} \\
&\quad + F \frac{\Delta t}{\varepsilon_0 \Delta x} \sum_{l=1}^{N} H(l)_{z,\text{ inc}} \Big|_{i-1,j,k}^{n+1/2} \\
H_x|_{i,j,k}^{n+1/2} &= \dot{H}_x|_{i,j,k}^{n+1/2} - \frac{\Delta t}{\mu_0 \Delta z} \sum_{l=1}^{N} E(l)_{y,\text{ inc}} \Big|_{i,j,k+1}^{n} \\
H_y|_{i,j,k}^{n+1/2} &= \dot{H}_y|_{i,j,k}^{n+1/2} + \frac{\Delta t}{\mu_0 \Delta z} \sum_{l=1}^{N} E(l)_{x,\text{ inc}} \Big|_{i,j,k+1}^{n}
\end{aligned} \tag{1.42}$$

and inside the higher separation plane, the following formulae are obtained:

$$\begin{aligned}
E_x|_{i,j,k}^{n+1} &= \dot{E}_x|_{i,j,k}^{n+1} - \frac{\Delta t}{\varepsilon_0 \Delta z} \sum_{l=1}^{N} H(l)_{y,\text{ inc}} \Big|_{i,j,k}^{n+1/2} \\
&\quad - F \frac{\Delta t}{\varepsilon_0 \Delta y} \sum_{l=1}^{N} H(l)_{z,\text{ inc}} \Big|_{i,j-1,k}^{n+1/2} \\
E_y|_{i,j,k}^{n+1} &= \dot{E}_y|_{i,j,k}^{n+1} + \frac{\Delta t}{\varepsilon_0 \Delta z} \sum_{l=1}^{N} H(l)_{x,\text{ inc}} \Big|_{i,j,k}^{n+1/2} \\
&\quad + F \frac{\Delta t}{\varepsilon_0 \Delta x} \sum_{l=1}^{N} H(l)_{z,\text{ inc}} \Big|_{i-1,j,k}^{n+1/2} \\
H_x|_{i,j,k}^{n+1/2} &= \dot{H}_x|_{i,j,k}^{n+1/2} + \frac{\Delta t}{\mu_0 \Delta z} \sum_{l=1}^{N} E(l)_{y,\text{ inc}} \Big|_{i,j,k}^{n} \\
H_y|_{i,j,k}^{n+1/2} &= \dot{H}_y|_{i,j,k}^{n+1/2} - \frac{\Delta t}{\mu_0 \Delta z} \sum_{l=1}^{N} E(l)|_{x,\text{ inc}} \Big|_{i,j,k}^{n}
\end{aligned} \tag{1.43}$$

When F=0 occurs when the cell is on the separation surface and N is the number of independent plane waves occurring on the entire field region. Additionally, the

value F=1 includes the additional fields outside the total field region when the cell is on some corners and some vertices. The separation planes are in the vacuum for all simulations (=0 F/m and =0 S/m), and equations have been presented for this situation. The common FDTD composition makes it simple to expand them to any media.

As long as the incident pulse differs from zero, these formulae are applied. In order to save calculation time, the source is disregarded after the incident pulse vanishes and the usual FDTD formulae are applied.

Making the Incident Fields Calculable

The incident fields can be computed in two different methods. Both the analytical approach and an additional one-dimensional (1-D) FDTD grid may be used to determine the field. When the interior volume is empty, the two approaches create a dispersion error that may be extracted from the field in the exterior dispersed field zone. In this instance, simply the contact between the total field region and the dispersed field region generates the fields. In both situations, the dispersion error is quantitatively assessed by contrasting the field acquired in the interior volume with the maximum field in the exterior region following M simulations. The supplementary 1-D FDTD simulation decreases the dispersion error when just one plane wave is stimulated. If a 1-D FDTD grid is utilized for each plane wave, this error reduction can be achieved by superimposing plane waves. We attempted to apply interpolations to simply a 1-D FDTD grid for all plane waves, however this resulted in a larger inaccuracy than utilizing the analytical computation of the incidence fields. In the end, we decided to use the analytical computation of the incident fields to shorten computation time and lower error. The size of the electric field produced by 100 incident random plane waves at 1 GHz on any flat plane parallel to the xy plane is seen in fig. 1.12. The following are the simulation data: A 60x60x60 FDTD grid with cells that are 0.025 m in size and 10 exterior cells in the dispersed field zone at the start and end of each axis have been used to simulate a volume. Every simulation has a time step of t=45 ps and 2247 iterations. Using several arbitrarily produced plane wave sets to examine each of the studied cases for the whole external dispersed field region, it is consistently found that the electric field's magnitude is less than 2% of the field region's overall values. The size of dispersed fields, on the other hand, was less than 5% of the total fields when employing a single 1-D FDTD grid for all plane waves. The incident field's temporal behavior in the conventional FDTD approach is a pulse modulated by a sinusoid. With just one FDTD compute run, this source enables frequency range investigation. The fast Fourier transform (FFT) of the fields in the investigational points and the incident field must be calculated during post-processing. In the suggested technique, the incidence field may be created either by a single plane wave or by superimposing N plane waves. While the field in the second instance is analytically determined in a point in the working volume, the field in the first case is derived by exciting the wave. We made an effort to use both options. For some frequencies and for some specific sets, the superpo-

sition of N plane waves might cause the incident field to vanish, which reduces the precision of the findings. On the M=200 simulations, this occurs around 10 times across all tests. As a result, the first method must be used. [10]

1.4 Aim of the study

The development of an intelligent wireless environment through which electromagnetic fields can be redirected and managed could also be beneficial with regard to the safety of biological tissues that are used to live and move within this environment. The main biological effect of the penetration of electromagnetic waves into the human body is heating. However, the levels we are normally exposed to are too low to cause significant warming. There are currently no known health effects caused by long-term exposure. However, intelligent use of reconfigurable smart surfaces could manage electromagnetic fields so that they can be redirected more precisely to everyday devices, thus avoiding excessive exposure of biological tissues to e.m. radio frequency. fields. The purpose of this study is divided into two parts:

- Develop a manually reconfigurable Intelligent Surface FDTD model, evaluating its operating principles, its response to an incident plane wave by evaluating the distribution of the electric field on it.
- The construction of a FDTD environment in which the presence of a metasurface, a plane wave incident on the metasurface and the reflected wave are simulated.

Two types of tests were performed, one of maximization and one of minimization, for the calculation of the magnitude of the reflected field in the desired direction. These two tests were performed both for a single plane wave and for a beam of 10 plane waves.

Capitolo 2

METHODS AND CONFIGURATION

The C programming language was used to implement the FDTD approach for all simulation analysis. GSL functions were used.

2.1 GSL and OPTIMIZATION

2.1.1 GSL Function

A numerical library for C and C++ programmers is called the GNU Scientific Library (GSL). Under the General Public License (GNU), it is free software. Numerous mathematical tools, like least-squares fitting, special functions, and random number generators, are included in the collection. There are more than 1000 functions in all, including special functions, numerical integration, solving differential equations, linear algebra, and optimization, along with a comprehensive test suite. A number of functions for computing unique functions, including gamma, Bessel, Airy, and Struve functions, are also included in the GSL library. In physics, mathematics, and engineering, these functions are frequently utilized to address certain issues. There are several functions for generating random numbers included in the GSL library. It has a top-notch random number generator that makes use of methods like Mersenne Twister and Ranlux. These operations can be used to produce random numbers with a variety of distributions, such as normal, exponential, uniform, etc. With its extensive collection of mathematical functions for use in scientific and engineering calculation, the GSL library is a highly potent library. Physics, mechanical engineering, electrical engineering, computer engineering, aerospace engineering, and many other scientific and technical domains make extensive use of it. Numerous numerical computing-related subjects are covered in the library. There are routines available for the locations listed in table 2.1. [1] The library has been used in many scientific and engineering applications, including simulation models, data analysis, image processing, signal processing, and automatic control. Due to its wide range

Tabella 2.1: Routines Areas

$ComplexNumbers$	$RootsofPolynomials$	$SpecialFunctions$
$VectorsandMatrices$	$Permutations$	$Combinations$
$Sorting$	$BLASSupport$	$LinearAlgebra$
$CBLASLibrary$	$FastFourierTransforms$	$Eigensystems$
$RandomNumbers$	$Quadrature$	$RandomDistributions$
$Quasi-RandomSequences$	$Histograms$	$Statistics$
$MonteCarloIntegration$	$N-Tuples$	$DifferentialEquations$
$SimulatedAnnealing$	$NumericalDifferentiation$	$Interpolation$
$SeriesAcceleration$	$ChebyshevApproximations$	$Root-Finding$
$DiscreteHankelTransforms$	$Least-SquaresFitting$	$Minimization$
$IEEEFloating-Point$	$PhysicalConstants$	$BasisSplines$
$Wavelets$	$SparseBLASSupport$	$SparseLinearAlgebra$

of math functions, its robustness, and its easy integration with other libraries, the GSL library is a popular choice for developers working in scientific and engineering fields.

2.1.2 Optimization

In this study the function of the GSL that has been used is that of optimization. Optimization is one of the main areas covered by the GSL library. It includes a set of algorithms for solving minimization and maximization problems of functions of several variables. The optimization algorithms available in the GSL library include:

- Gradient Methods: Use the gradient of the function to move along the direction of maximum decrement of the function.
- Newton methods: use the Hessian matrix of the function to calculate the direction of maximum decrease of the function.
- Quasi-Newton methods: use an approximation of the Hessian matrix to compute the direction of maximum decrease of the function.
- Powell methods: a global optimization algorithm based on direction finding.
- Search algorithms based on Simplex: such as the Nelder-Mead method.

In general, optimization consists of finding the values of one or more variables that maximize or minimize an objective function. There are several categories of optimization algorithms, including minimum-finding algorithms, gradient algorithms,

and evolution algorithms. Minimum-finding algorithms use a process of trial and error to find the minimum of a function. These algorithms are often used when the shape of the objective function is not known or is too complex to be analyzed mathematically. Gradient algorithms use the derivative of the objective function to find the minimum. These algorithms are often more efficient than minimum-finding methods, but require the objective function to be continuous and differentiable. Evolution algorithms, such as the genetic algorithm and the intelligent swarm algorithm, are based on the idea of simulating natural selection to find the minimum of a function. These algorithms are often used for large optimization problems or for problems where the shape of the objective function is not known. The GSL library provides a variety of algorithms for optimizing mathematical functions, including some of the methods mentioned above. The use of these algorithms depends on the specific needs of the optimization problem and on the characteristics of the objective function. All of these algorithms can be used to solve minimization and maximization problems and can be used with functions of multiple variables. Furthermore, the GSL library also offers functions for solving non-linear optimization problems with constraints. In summary, GSL is a powerful and versatile library that offers a wide range of mathematical functions for scientific and engineering computation. It is open-source, well-documented, easy to use, and community-developed. It is used in many fields of science and engineering and can be easily integrated with other libraries or systems.

2.1.3 Multidimensional Minimization

This section covers techniques for locating the minimum of any multidimensional function. Low-level components for several iterative minimizers and convergence tests are included in the library. The user can combine these to produce the required result, giving complete access to the algorithms' intermediary phases. Because each method class shares the same foundation, you may choose between minimizers at runtime without recompiling your application. Minimizers may be utilized in multi-threaded systems since each instance keeps track of its state. By flipping the sign of a function, minimization procedures may be utilized to maximize it. The problem of multidimensional minimization requires finding a point x such that the scalar function,

$$f(x_1, ..., x_n) \tag{2.1}$$

The multidimensional minimization issue entails finding a point x such that the scalar function f takes on a value less than any neighboring point. The gradient

$$g = \nabla f \tag{2.2}$$

vanishes to zero for smooth functions. There are no parenthetical approaches for reducing n-dimensional functions in general. The algorithms begin with an initial guess and use a search algorithm that aims to move downward. [1] Algorithms that use the function gradient minimize a one-dimensional line in this direction until

the lowest point with an appropriate tolerance is discovered. Then, using local information from the function and its derivatives, the search direction is modified, and the process is repeated until the true n-dimensional minimum is found. Different tactics are used by algorithms that do not require the gradient of the function. The Nelder-Mead Simplex technique, for example, keeps $n + 1$ test parameter vectors as vertices of an n-dimensional simplex. It attempts to improve the poorest vertex of the simplex by geometric modifications at each iteration. Iterations continue until the simplex's total size is small enough. Both types of algorithms rely on a common foundation. The user supplies the algorithms with a high-level driver, while the library provides the particular functions required for each step. The iteration process is divided into three stages. The steps are as follows:

- initialize minimizer state, s, for algorithm T;
- update s using the iteration T;
- test s for convergence, and repeat iteration if necessary

Either the line-minimization of the current direction is improved, or the search direction is updated, throughout each iteration step. It should be noted that the minimization algorithms can only find one local minimum at a time. When there are numerous local minima in the search region, the first minimum detected will be returned; however, which of the minima this will be is impossible to anticipate. If you try to discover a local minimum in an area where there are many, no error will be reported in most circumstances. It's also crucial to remember that the minimization methods only locate local minima; it is impossible to tell if a particular minimum corresponds to the function's global minimum.

2.1.4 Initializing the Multidimensional Minimizer

The function initializes a multidimensional minimizer. It depends only on the problem size and the algorithm and can be reused for different problems. A pointer to a freshly allocated instance of an n-dimensional minimizer of type T is returned by this function. If there is insufficient memory to generate the minimizer, the method produces a null pointer and an error code is sent to the error handler. Starting from the initial point x, the function initializes the minimizer s to minimize the function fdf. Step-size specifies the size of the initial trial step. The tol specifies the precision of the line minimization. The exact meaning of this parameter is determined by the method utilized. Line minimization is often regarded successful if the gradient of the function g is orthogonal to the current search direction p and has a relative accuracy of tol, where:

$$p \cdot g < tol|p||g| \tag{2.3}$$

Because line reduction only has to be done roughly, a tol value of 0.1 is adequate for most tasks. It is important to note that setting tol to z ero forces the usage of "precise" line-searches, which are exceedingly costly.

2.1.5 Providing a function to minimize

You must supply a parametric function with n variables for the minimizers to work with. You may additionally need to give a code that calculates the function's gradient and a third method that calculates both the function value and the gradient. The following data types are used to define the functions in order to accommodate generic parameters:
$type"gsl"multimin"function"fdf$
This data type represents a generic function with n variables and parameters, as well as the related gradient vector of the derivatives,

$$f(x, params) \tag{2.4}$$

for argument x and parameters params. If the function cannot be computed, an error value should be retourned.

2.1.6 Iteration

Each algorithm's iteration is driven by the following function. The function does one iteration to update the minimizer's state. Because the same function is used by all minimizers, alternative techniques may be swapped at runtime without modifying the code.
$intgslmultiminfdfminimizeriterate(gslmultiminfdfminimizer * s)$
$intgslmultiminfminimizeriterate(gslmultiminfminimizer * s)$
These routines execute one iteration of the minimizers. If an unanticipated problem occurs during the loop, an error code will be returned. The error number indicates that the minimizer has been unable to improve on its present estimate, either because of numerical complexity or because a true local minimum has been obtained. [1]

2.1.7 Stopping Criteria

A minimizing process ought to come to an end when one of the following is true:

- A minimum has been found to within the user-specified precision;
- A user-specified maximum number of iterations has been reached;
- An error has occurred.

The user has discretion over how these circumstances are handled. The functions listed below help the user to validate the current result's accuracy.
$intgslmultimintestgradient(constgslvector * g, doubleepsabs)$
This function compares the gradient g norm to the absolute tolerance epsabs. At a minimum, the gradient of a multidimensional function is zero. The test returns GSL SUCCESS if the following condition is achieved:

$$|g| < epsabs \tag{2.5}$$

Otherwise, it returns GSL CONTINUE. For tiny fluctuations in x, an acceptable selection of epsabs can be made based on the required precision in the function.

2.2 CODE DESCRIPTION

In this section of the thesis, I present a series of scripts written in C language used to process the data collected during my research. The C language was chosen for its effectiveness in manipulating large amounts of data and for its speed of execution. The scripts have been developed to analyze the collected data in order to identify significant patterns and provide detailed results on the relationships between the studied variables. The first script shown is related to the data, it contains the definitions of the parameters used in the fdtd. [1]

```
1     /* Frequenza di lavoro */
2 #define RCFDTD_FREQ 3.165E9
3 #define RCFDTD_FREQ_MIN 0.8E9
4 #define RCFDTD_FREQ_MAX 8.4E9
5
6 /* Seleziona il tipo di celle e il tipo di mezzo.
7  * se omogeneo e a griglia costante degli array si riducono a
     numeri con notevoli vantaggi sul
8  * ‘cache missing‘. */
9 #define  RCFDTD_DELTA_COSTANTE 0  /* 0: almeno un passo delle
     celle e‘ variabile. 1: il passo delle celle e‘ costante */
10 #define  RCFDTD_CELLA_CUBO 0      /* 0: le celle sono
     parallelepipedi. 1: tutte le celle sono un cubo e sono
     uguali */
11 #define  RCFDTD_MEZZO_OMOGENEO 0  /* 0: il mezzo e‘
     disomogeneo. 1: il mezzo e‘ omogeneo */
12
13 /* definizione per la PML */
14 #define RCFDTD_PMLORD 3   /*ordine della PML */
15 /*#define RCFDTD_NPML  16                   celle attenuatrici
      */
16 /*#define RCFDTD_PMLDB -160                 attenuazione MAX
     */
17 #define RCFDTD_NPML  8    /* celle attenuatrici */
18 #define RCFDTD_PMLDB -60  /* attenuazione MAX */
19
20 /* definizione del passo spaziale e temporale */
21 #define RCFDTD_DELTA_BASE_X 0.001
22 #define RCFDTD_DELTA_BASE_Y 0.001
23 #define RCFDTD_DELTA_BASE_Z 0.001
24 #define RCFDTD_DELTA_T 1.5E-12
25
26 /* Impedenze e costante dielettrica relativa delle linee di
     trasmissione collegate alle antenne */
27 /* RG/58U epslon_r=2.26, mu_r=1, a=0.406 mm, b=1.48 mm, Z0
     =51.6 ohm */
```

```
28 #define RCFDTD_LINEA1_Z0 51.6
29 #define RCFDTD_LINEA1_EPSR 2.26
30 #define RCFDTD_LINEA2_Z0 51.6
31 #define RCFDTD_LINEA2_EPSR 2.26
32
33 /* DIMensioni massime dei vettori ove imporre la sorgente */
34 #define RCFDTD_NSX 10
35 #define RCFDTD_NSY 10
36 #define RCFDTD_NSZ 10
37
38 /* Distanza della separazione dal bordo */
39 #define RCFDTD_DISTSEP 20
40 /* Distanza dell'oggetto dalla separazione */
41 #define RCFDTD_DISTOGG 35
42
43 /* In input per l'eccitazione della struttura */
44 /* #define INPUT_SIGNAL_READ yes */
45
46 #define RCFDTD_TIPO 1   /* Eccitazione con un impulso (1) */
47 /*#define RCFDTD_TIPO 2*/       /* Eccitazione con solo una
      sinusoide (2) con due sinusoidi (3) */
48 #define RCFDTD_NOP 1    /* numero di onde piane incidenti */
49
50 /* Dimensioni della mappa biologica utilizzata */
51 #define UOMO_I 1
52 #define UOMO_J 1
53 #define UOMO_K 1
54
55 /* Celle da aggiungere intorno al dominio, prima delle ABC */
56 #define BCINX 20
57 #define BCDIMX 20
58 #define BCINY 20
59 #define BCDIMY 20
60 #define BCINZ 20
61 #define BCDIMZ 20
```

Here the parameters are defined, modifiable according to the needs, such as the frequencies, all the parameters relating to the cells, the impedances and the relative dielectric constant of the transmission lines connected to the antennas and the maximum dimensions of the vectors where to impose the source.

The following code defines some options for the simulation output, such as printing the fields in the time interval, printing the fields in all points of the grid, printing the geometry in some ASCII files, calculating the effective volume and the printout of the quantities in the frequency domain. Additionally, there are some definitions of numerical parameters for the simulation, such as grid size, number of points probed, number of stirrer angles, and array size for Fourier transform.

```
    #define PREFISSO_SIM "PW-BS-gruppi_"
#define LABEL_SIM 5
#define POSIZIONI_STIRRER 1 /* Number of plane wave sets -
   Usare una potenza del due o meglio il numero `bg_size`
   della simulazione */
#define POSIZIONI_STIRRER1 8
#define POSIZIONI_STIRRER2 8
#define POSIZIONI_STIRRER3 8

#define COPERCHIO 0 /* solo nel caso di BSL, =1 => schermo
   presente, =0 => schermo assente */

/* Parameters for output */

/* Print the fields in the time domain */
/* #define FDTD_PRINT_TIME yes */

/* Print the field values in all the cells */
/* #define FDTD_PRINT_TIME_FIELDS yes */

/* Print the geometry in some ascii files */
/* #define FDTD_PRINT_GEO yes */
/* Count the filled metallic cell and compute the effective
   volume for the computation */
/* #define FDTD_COMPUTE_VOL yes */

/* Print the fields in the frequency domain in linear scale */
#define FFT_PRINT_FREQ yes
/*  - Print the fields in the frequency domain - do not print
   all the data but print one field every `FFT_PRINT_FREQ_STEP
   ` fields */
#define FFT_PRINT_FREQ_STEP 2
/* Print the fields in the frequency domain in linear scale */
/*#define FFT_PRINT_FREQ_LOG yes*/
/*  - Print the fields in the frequency domain - do not print
   all the data but print `N_LOG_SPACED` values */
/*#define N_LOG_SPACED 100*/

/* Print in the time domain the transmitted and reflected
   signals (only, any grid point is printed) */
#define FDTD_PRINT_PARS yes

/* Print the correlation matrix for every frequency - case UP-
   SM - http://arxiv.org/abs/1404.6335 */
/* #define STAT_PRINT_CORR_MATRIX yes */

/* Print the maximum field in the last period for harmonic
   source */
#define FDTD_PRINT_MAX_FIELDS yes

/* For the case when the probed points are read from a file,
   it is the maximum dimension for the array
```

```
42  * (for each MPI process) */
43 #define MAX_DIM_POINTS 1
44 /* For the case when the probed points are read from a file,
       it is the number of files where the point
45  * coordinates are stored */
46 #define NFILE_POINTS 1
47 /* For the case when the MPI simulations differ from other
       parameters than stirrer angles it is the number
48  * of investigates different conductivities of the air */
49 #define POS_SIGMA 1
50
51 /* Il range dei valori partono da un minimo di 3 per ogni
       componente (il calcolo statistico ‘antenne‘
52  * vuole il punto centrale fino ad un massimo che dipende
       dalla RAM disponibile */
53 #define PRINT_NX 10   /* Celle da stampare lungo l'asse x */
54 #define PRINT_NY 1    /* Celle da stampare lungo l'asse y */
55 #define PRINT_NZ 10   /* Celle da stampare lungo l'asse z */
56 #define PRINT_DELTA_X 20  /* Celle lungo x da saltare nella
       stampa */
57 #define PRINT_DELTA_Y 20  /* Celle lungo y da saltare nella
       stampa */
58 #define PRINT_DELTA_Z 20  /* Celle lungo z da saltare nella
       stampa */
59 #define PRINT_XIN 0.010   /* Coordinata x del primo punto da
       stampare */
60 #define PRINT_YIN 0.102   /* Coordinata y del primo punto da
       stampare */
61 #define PRINT_ZIN 0.010   /* Coordinata z del primo punto da
       stampare */
62 #define PRINT_DIM_BUFFER_MB 8 /* Dimensione del buffer da
       mandare in stampa tutto insieme per renderla piu‘ veloce */
63 #define FFT_N 524288    /* Dimensione dell'array da
       trasformare */
64 /*#define FFT_N 1048576 */  /* Dimensione dell'array da
       trasformare */
```

Now the script containing the calculation routines of the FDTD is shown:

```
1     #include "struttura_dati.h"
2 #include "dati_main.h"
3 int
4 calcola_fdtd (long int time, double dt, struct_campoEM *
    campo_g1, struct_fdtd * fdtd_g1, struct_griglia *
    griglia_g1,
5               struct_eccitazione * ecc, struct_piani_sep *
    campi_sep, struct_dim_sep * sep, struct_point_meter *
    origine, struct_onda_piana * op,
6               int rank)
7 {
8   /* int i, j, k; */
9   /*double tempo; */
10  /* CHIAMATE A TUTTE LE FUNZIONI NECESSARIE */
11
12 #ifdef DEBUG_FDTD
13   printf ("Calcola H (time=%ld):\n", time);
14 #endif
15   calcola_h (campo_g1, fdtd_g1, griglia_g1);
16      if (time < ecc->time_pulse_stop)
17        {
18          calcola_h_sep_pianoz (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 1, time, dt, op, ecc);
19          calcola_h_sep_pianoy (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 2, time, dt, op, ecc);
20          calcola_h_sep_pianox (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 3, time, dt, op, ecc);
21          calcola_h_sep_pianox (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 4, time, dt, op, ecc);
22          calcola_h_sep_pianoy (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 5, time, dt, op, ecc);
23          calcola_h_sep_pianoz (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 6, time, dt, op, ecc);
24        }
25 #ifdef DEBUG_FDTD
26   printf ("Calcola E (time=%ld):\n", time);
27 #endif
28   calcola_e (campo_g1, fdtd_g1, griglia_g1);
29       if (time < ecc->time_pulse_stop)
30         {
31           calcola_e_sep_pianoz (campo_g1, fdtd_g1, griglia_g1,
     campi_sep, sep, (unsigned int) 1, time, dt, op, ecc);
32           calcola_e_sep_pianoy (campo_g1, fdtd_g1, griglia_g1,
     campi_sep, sep, (unsigned int) 2, time, dt, op, ecc);
33           calcola_e_sep_pianox (campo_g1, fdtd_g1, griglia_g1,
     campi_sep, sep, (unsigned int) 3, time, dt, op, ecc);
34           calcola_e_sep_pianox (campo_g1, fdtd_g1, griglia_g1,
     campi_sep, sep, (unsigned int) 4, time, dt, op, ecc);
35           calcola_e_sep_pianoy (campo_g1, fdtd_g1, griglia_g1,
```

```
    campi_sep, sep, (unsigned int) 5, time, dt, op, ecc);
36          calcola_e_sep_pianoz (campo_g1, fdtd_g1, griglia_g1,
    campi_sep, sep, (unsigned int) 6, time, dt, op, ecc);
37        }
38
39  /* CHIAMATA ALLE CONDIZIONI AL CONTORNO DI MUR 2 */
40  sing_inz_2 (campo_g1, fdtd_g1, griglia_g1);
41  sing_dimz_2 (campo_g1, fdtd_g1, griglia_g1);
42  sing_iny_2 (campo_g1, fdtd_g1, griglia_g1);
43  sing_dimy_2 (campo_g1, fdtd_g1, griglia_g1);
44  sing_inx_2 (campo_g1, fdtd_g1, griglia_g1);
45  sing_dimx_2 (campo_g1, fdtd_g1, griglia_g1);
46  spigoli (campo_g1, griglia_g1);
47  ricopia_inz_2 (campo_g1, fdtd_g1, griglia_g1);
48  ricopia_dimz_2 (campo_g1, fdtd_g1, griglia_g1);
49  ricopia_iny_2 (campo_g1, fdtd_g1, griglia_g1);
50  ricopia_dimy_2 (campo_g1, fdtd_g1, griglia_g1);
51  ricopia_inx_2 (campo_g1, fdtd_g1, griglia_g1);
52  ricopia_dimx_2 (campo_g1, fdtd_g1, griglia_g1);
53
54  return (0);
55 }
```

This code performs an electromagnetism simulation based on a numerical modeling technique called Finite-Difference Time-Domain (FDTD). In particular, this function calculates the evolution of an electromagnetic field over time on the basis of a series of partial differential equations which describe the interaction of electromagnetic waves with the medium in which they propagate. The function takes various parameters as input, including the current time, the time increment, the electromagnetic field, the calculation grid, the electromagnetic field excitation and the boundary conditions. The function performs a series of calculations to determine the value of the electromagnetic field at a later point in time. In particular, it computes the magnetic field H and the electric field E, and then applies the boundary conditions of type Mur 2.

The file containing the FDTD initialization routines is shown in the next part:

```
1     #include "struttura_dati.h"
2
3  void posizione_sorgente_1D (struct_eccitazione_1D * ecc1D);
4  void metti_ms_basetta (struct_campoEM * campo, struct_fdtd *
       fdtd, struct_griglia * griglia, double dt, struct_ms * ms,
5                         const gsl_vector * v, int * point_cap,
       int fl_diemet, int fl_solopos, int fl_solocond, int rank);
6  int
7  InizializzaDati_CaricaConf (double dt, struct_griglia *
       griglia_g1, struct_campoEM * campo_g1, struct_campo_max *
       campo_max_g1,
8                              struct_fdtd * fdtd_g1,
       struct_eccitazione * ecc, struct_onda_piana * op,
9                              struct_piani_sep * campi_sep,
       struct_dim_sep * sep, struct_point_meter * origine,
       struct_ms * ms,
10                             double sigma, const gsl_vector *v,
       double *p, int * point_cap, int rank)
11 {
12
13   int sim_completa = (int) (p[0] + 0.5);  // Simulazione
       completa prima o dopo l'ottimizzazione con la stampa dell'
       S21 su tutta la banda
14   /* Compute the domain as function on elementary cells */
15   /* fino alla .... */
16   ms->dir = 1;
17   ms->nx = (int) (p[4] + 0.5);
18   ms->ny = 0;
19   ms->nz = (int) (p[5] + 0.5);
20   ms->Dx = (int) (p[6] + 0.5);
21   ms->Dy = 0;
22   ms->Dz = (int) (p[7] + 0.5);
23   ms->wx = 1;
24   ms->wy = 0;
25   ms->wz = 1;
26   ms->Gx = (int) (p[10] + 0.5);
27   ms->Gy = 0;
28   ms->Gz = (int) (p[11] + 0.5);
29   ms->d = 1;
30   int right_shift = 0;
31   int Lx_basetta = ms->nx * (ms->Dx + ms->wx) + ms->wx;
32   int Ly_basetta = ms->d;
33   int Lz_basetta = ms->nz * (ms->Dz + ms->wz) + ms->wz;
34   ms->Lx = Lx_basetta;
35   ms->Ly = Ly_basetta;
36   ms->Lz = Lz_basetta;
37   int Lx_dominio = Lx_basetta;
38   int Ly_dominio = Ly_basetta;
39   int Lz_dominio = Lz_basetta;
```

```
40
41   double appo, d_max, distanza;
42   long int i, j, k, ind;
43   struct_point_meter punto;
44
45 /* Tipo di griglia utilizzata */
46   griglia_g1->delta_costante = RCFDTD_DELTA_COSTANTE;
47   griglia_g1->cella_cubo = RCFDTD_CELLA_CUBO;
48   griglia_g1->mezzo_omogeneo = RCFDTD_MEZZO_OMOGENEO;
49
50   /* Aggiungo al volume le condizioni al contorno e la
      separazione */
51   griglia_g1->dimx = (Lx_dominio + (RCFDTD_DISTSEP +
      RCFDTD_DISTOGG) * 2) + 1;
52   griglia_g1->dimy = (Ly_dominio + (RCFDTD_DISTSEP +
      RCFDTD_DISTOGG) * 2) + 1;
53   griglia_g1->dimz = (Lz_dominio + (RCFDTD_DISTSEP +
      RCFDTD_DISTOGG) * 2) + 1;
54
55   /* These values must be assigned just after dimx, dimy, dimz
       */
56   piu_meno (griglia_g1);
57
58   /* Alloco lo spazio in memoria per il campo */
59   alloca_campi (campo_g1, griglia_g1);
60   alloca_campi_max (campo_max_g1, griglia_g1);
61   /* alloca_campi_max (campo_max_h1, griglia_g1); */
62   /* Alloco lo spazio in memoria per la condizione di Mur2 */
63   alloca_mur2 (fdtd_g1, griglia_g1);
64   /* Alloco lo spazio in memoria per i coefficienti */
65   alloca_coef (fdtd_g1, griglia_g1);
66   /* Allocate the memory for the grid and set the basic epsr
      and sig */
67   alloca_griglia (griglia_g1, (double) 1, sigma);
68   /* Alloca gli array ove fare la separazione dei campi */
69   alloca_piani_sep (campi_sep, griglia_g1);
70
71   /* Coordinate della separazione */
72   sep->inx = RCFDTD_DISTSEP;
73   sep->iny = RCFDTD_DISTSEP;
74   sep->inz = RCFDTD_DISTSEP;
75   sep->dimx = griglia_g1->dimx - RCFDTD_DISTSEP;
76   sep->dimy = griglia_g1->dimy - RCFDTD_DISTSEP;
77   sep->dimz = griglia_g1->dimz - RCFDTD_DISTSEP;
78
79   // origine degli assi coordinati riferita all'origine delle
      celle (in metri)
80   origine->x = (double) (sep->inx + sep->dimx) * (double) 0.5
      *(double) RCFDTD_DELTA_BASE_X;
81   origine->y = (double) (sep->iny + sep->dimy) * (double) 0.5
      *(double) RCFDTD_DELTA_BASE_Y;
82   origine->z = (double) (sep->inz + sep->dimz) * (double) 0.5
```

```
    *(double) RCFDTD_DELTA_BASE_Z;
83  //printf("Origine per l'onda incidente in coordinate celle:
    x=%g, y=%g z=%g\n",origine->x*(double)FATTORE/(double)
    RCFDTD_DELTA_BASE_X,
84  //        origine->y*(double)FATTORE/(double)
    RCFDTD_DELTA_BASE_Y,origine->z*(double)FATTORE/(double)
    RCFDTD_DELTA_BASE_Z);
85
86  /*-----------------------------------------------------*/
87  /* Da richiamare 4 volte:
88   * Prima per le dimensioni
89   * Seconda per il dielettrico
90   * Terza per il metallo
91   * Quarta per i condensatori
92   * Qui richiamo per le dimensioni */
93  ms->x0 = (Lx_dominio - Lx_basetta) / 2 + right_shift;
94  ms->y0 = (Ly_dominio - Ly_basetta) / 2;
95  ms->z0 = (Lz_dominio - Lz_basetta) / 2;
96  check_punto_in (ms->x0, ms->y0, ms->z0, griglia_g1, rank);
97  /* .... Non chiamare, messe nel programma principale .... */
98  //metti_ms_basetta (campo_g1, fdtd_g1, griglia_g1, dt, ms, v
    , point_cap, 0, 1, 0, rank);
99  printf("Coordinate primo punto basetta: ms->x0: %d ms->y0: %
    d ms->z0: %d\n", ms->x0, ms->y0, ms->z0);
100
101 appo = (double) 0;
102 for (i = 0; i < griglia_g1->dimx; i++)
103   {
104     griglia_g1->ellex[i] = appo - origine->x;
105     griglia_g1->dx[i] = (double) RCFDTD_DELTA_BASE_X;
106     appo += griglia_g1->dx[i];
107   }
108 appo = (double) 0;
109 for (j = 0; j < griglia_g1->dimy; j++)
110   {
111     griglia_g1->elley[j] = appo - origine->y;
112     griglia_g1->dy[j] = (double) RCFDTD_DELTA_BASE_Y;
113     appo += griglia_g1->dy[j];
114   }
115 appo = (double) 0;
116 for (k = 0; k < griglia_g1->dimz; k++)
117   {
118     griglia_g1->ellez[k] = appo - origine->z;
119     griglia_g1->dz[k] = (double) RCFDTD_DELTA_BASE_Z;
120     appo += griglia_g1->dz[k];
121   }
122
123 ecc->tipo = RCFDTD_TIPO;  /* impulso o sinusoidi */
124
125 /* Tipo di eccitazione */
126 switch (ecc->tipo)
127   {
```

```
128     case 1:
129       strcpy (ecc->nome, "Eccitazione ad impulso modulato");
130       ecc->fmin = p[2];
131       ecc->fmax = p[3];
132       ecc->freq = (ecc->fmin + ecc->fmax) / 2.0;
133       ecc->omega = 2.0 * (double) RCFDTD_PI *ecc->freq;
134       ecc->tg =
135     12.0 / (((double) RCFDTD_PI * (ecc->fmax - ecc->fmin)) * ((
        double) RCFDTD_PI * (ecc->fmax - ecc->fmin)));
136       ecc->t0 = 3.0 * sqrt (ecc->tg);
137       ecc->passi_per = 0.5 + 1.0 / (ecc->freq * dt);
138       /*ecc.Nperiod=atoi(riga);  periodi da osservare */
139       /*ecc->Nperiod = 7500;     periodi da osservare */
140       /*ecc->Nperiod = 16500;    periodi da osservare */
141       ecc->Nperiod = 100;      /* periodi da osservare */
142       //ecc->Nperiod = 15;  /* periodi da osservare */
143       /* Stesso ‘tfinale‘ sia nel caso dell'analisi completa,
        sia per la valutazione durante il ciclo iterativo */
144       ecc->tfinale = ecc->Nperiod * sqrt (ecc->tg) * (ecc->
        fmax - ecc->fmin) /
145                        (dt * ((double) RCFDTD_FREQ_MAX - (double
        ) RCFDTD_FREQ_MIN)) ;
146       ecc->time_pulse_stop = ecc->tfinale;
147       ecc->omegadt = 2.0 * (double) RCFDTD_PI *ecc->freq * dt;
148       ecc->omega = 2.0 * (double) RCFDTD_PI *ecc->freq;
149       break;
150     case 2:
151       strcpy (ecc->nome, "Eccitazione Sinusoidale");
152       ecc->freq = (double) RCFDTD_FREQ;
153       ecc->omega = 2.0 * (double) RCFDTD_PI *ecc->freq;
154       ecc->fmin = ecc->freq;
155       ecc->fmax = ecc->freq;
156       ecc->Nperiod = 20;  /* periodi da osservare */
157       //ecc->Nperiod = 20;      /* periodi da osservare */
158       ecc->tfinale = ecc->Nperiod * (0.5 + 1.0 / (ecc->freq *
        dt));
159       ecc->time1 = (ecc->Nperiod - 1) * (0.5 + 1.0 / (ecc->
        freq * dt));
160       ecc->time2 = ecc->time1 + 1.0;
161       ecc->passi_per = 0.5 + 1.0 / (ecc->freq * dt);
162       ecc->omegadt = 2.0 * (double) RCFDTD_PI *ecc->freq * dt;
163       ecc->E0min = 1.0;
164       break;
165     default:
166       return (EXIT_FAILURE);
167     }
168
169     /* seleziona il tipo di sorgente usa freq, chiamare dopo
        */
170   crea_onda_piana (op, ecc, griglia_g1, rank);
171
172   /* Calcolo del numero delle iterazioni dopo le quali la
```

```
     sorgente gaussiana si e' esaurita */
173    /* cicla su tutte le onde piane da applicare come sorgente
     */
174    d_max = 0.0;
175    for (ind = 0; ind < RCFDTD_NOP; ind++)
176      {
177        ecc->time_pulse_stop = ecc->tfinale;
178        for (i = 0; i < 2; i++)
179          for (j = 0; j < 2; j++)
180            for (k = 0; k < 2; k++)
181              {
182                punto.x = (sep->inx + (sep->dimx - sep->inx) * i
     ) * griglia_g1->dx[0] - origine->x;
183                punto.y = (sep->iny + (sep->dimy - sep->iny) * j
     ) * griglia_g1->dy[0] - origine->y;
184                punto.z = (sep->inz + (sep->dimz - sep->inz) * k
     ) * griglia_g1->dz[0] - origine->z;
185                distanza = ddist_punto_piano (punto, op->piano[
     ind]);
186                d_max = fmax (d_max, distanza);
187              }
188      }
189    if (ecc->tipo == 1)
190      {
191        ecc->time_pulse_stop = (double) 2 *(d_max / (double)
     RCFDTD_C0 + ecc->t0) / dt;
192        printf ("Numero di iterazioni durante le quali la
     sorgente e' accesa: %g\n", ecc->time_pulse_stop);
193      }
194    else
195      ecc->time_pulse_stop = ecc->tfinale;
196
197    /* Deve essere chiamata adesso */
198
199    /* Da richiamare 4 volte:
200     * Prima per le dimensioni
201     * Seconda per il dielettrico
202     * Terza per il metallo
203     * Quarta per i condensatori
204     * Qui richiamo per il dielettrico */
205     metti_ms_basetta (campo_g1, fdtd_g1, griglia_g1, dt, ms, v,
      point_cap, 0, 0, 0, rank);
206
207    return (0);
208  }
209
210  metti_ms_basetta (struct_campoEM * campo, struct_fdtd * fdtd,
     struct_griglia * griglia, double dt, struct_ms * ms,
211        const gsl_vector * v, int * point_cap, int fl_diemet,
     int fl_solopos, int fl_solocond, int rank)
212  {
213  /*
```

```
214  * ms->x0, ms->y0, ms->z0: coordinate in celle dove mettere l'
       inizio della metasuperficie (in coordinate della griglia
       basetta)
215  * fl_solopos = 1 inserisce solo le coordinate e l'
       orientazione del dispositivo -- fl_solopos = 0 inserisce
       anche metallo o dielettrico e condensatore
216  * fl_diemet = 0, fl_solocond = 0 inserisce solo il
       dielettrico -- fl_diemet = 1 inserisce solo il metallo
217  * fl_solocond = 1, fl_solopos = 0, fl_diemet = 0, mette il
       condensatore
218  * Coordinate in celle  (lato cella 1 mm)
219  * Metalli: PEC (conduttori senza perdite)
220  * PER IL MOMENTO SOLA BASETTA SUL PIANO X-Z
221  */
222
223   int i, j, k;
224   int spost = 0;
225   int xi, yi, zi, cin, xin, xfi, yin, yfi, zin, zfi;
226   double epsr_die = 4.4, sigma_die = 0.0025;
227   // double cap = 0.1E-12; Passate come parametro dalla
       funzione di ottimizzazione
228   spost = (int) RCFDTD_DISTSEP + (int) RCFDTD_DISTOGG;
229 /*
230 */
231   switch (ms->dir)
232     {
233     case 1:
234       if (!fl_solopos)
235   {
236     yi = ms->y0 + ms->d;
237     if (fl_diemet)
238       {
239 /* Piano di massa o basetta (piano xz) */
240 /*
241         crea_quad_pec (campo, fdtd, griglia, ms->x0 + spost,
      ms->x0 + ms->Lx + spost, ms->z0 + spost, ms->z0 + ms->Lz +
      spost, ms->y0 + spost, ms->dir);
242 */
243 /* Patch di matallo ideale (piano xz) */
244         for (i = 0; i < ms->nx; i++)
245     {
246       xin = ms->x0 + ms->wx + i * (ms->Dx + ms->wx);
247       xfi = xin + ms->Dx;
248       for (k = 0; k < ms->nz; k++)
249         {
250           zin = ms->z0 + ms->wz + k * (ms->Dz + ms->wz);
251           zfi = zin + ms->Dz;
252           crea_quad_pec (campo, fdtd, griglia, xin + spost,
      xfi + spost, zin + spost, zfi + spost, yi + spost, ms->dir)
      ;
253           //crea_cross_pec (campo, fdtd, griglia, xin + spost,
       xfi + spost, zin + spost, zfi + spost, yi + spost, ms->Gx,
```

```
 ms->Gz, ms->dir);
254         }
255      }
256        }
257      else if (fl_solocond)
258        /* Prima mettiamo tutte le capacita' orizzontali (lungo
    x), poi quelle verticali (lungo z)
259         ******* FARE RIFERIMENTO QUI PER L'ORDINE DEI DIODI
    *******/
260        {
261          for (k = 0; k < ms->nz; k++)
262      {
263        zi = ms->z0 + ms->wz + ms->Dz / 2 + k * (ms->Dz + ms->wz
    );
264        for (i = 0; i < ms->nx - 1; i++)
265          {
266            xi = ms->x0 + ms->wx + ms->Dx + i * (ms->Dx + ms->wx
    );
267            /* Se la polarizzazione dei diodi varia, e quindi la
     capacit  varia, scrivere la legge qui */
268            metti_condensatore (xi + spost, yi + spost, zi +
    spost, dt, gsl_vector_get (v, point_cap[i + k * (ms->nx -
    1)]), fdtd, griglia, 0);
269          }
270      }
271          cin = ms->nz * (ms->nx - 1);
272          for (i = 0; i < ms->nx; i++)
273      {
274        xi = ms->x0 + ms->wx + ms->Dx / 2 + i * (ms->Dx + ms->wx
    );
275        for (k = 0; k < ms->nz - 1; k++)
276          {
277            zi = ms->z0 + ms->wz + ms->Dz + k * (ms->Dz + ms->wz
    );
278            /* Se la polarizzazione dei diodi varia, e quindi la
     capacit  varia, scrivere la legge qui */
279            metti_condensatore (xi + spost, yi + spost, zi +
    spost, dt, gsl_vector_get (v, point_cap[cin + k + i * (ms->
    nz - 1)]), fdtd, griglia, 2);
280          }
281      }
282        }
283      else
284        {
285          for (i = ms->x0 + spost; i < ms->x0 + ms->Lx + spost;
    i++)
286      for (j = ms->y0 + spost; j < ms->y0 + ms->Ly + spost; j++)
287        for (k = ms->z0 + spost; k < ms->z0 + ms->Lz + spost; k
    ++)
288          {
289            griglia->epsr[i][j][k] = epsr_die;
290            griglia->sig[i][j][k] = sigma_die;
```

```
291           }
292         }
293     }
294         break;
295         /* Basetta normale all'asse-x e all'asse-z, DA FARE */
296       case 0:
297       case 2:
298       default:
299         if (!rank)
300     printf ("Chiamata errata a `metti_ms_basetta`: dir = %d  non
        e` implementato\n", ms->dir);
301         exit (EXIT_FAILURE);
302         break;
303       }
304 }
```

This code simulates metasurfaces made by patches interconnected by varactor diodes. Optimization of capacitors is implemented

```
1     #include "struttura_dati.h"
2 #include "dati_main.h"
3
4 #ifdef LIBMPI
5 int main_mpi (double dt, int rank, int size, MPI_Comm comm,
    double *buffer_stampe,
6          size_t dim_buffer, size_t npunti, struct_point_cell *
    probed_points,
7          size_t num_point_probed, const gsl_vector * v, double
    *p, int nsim);
8 int fft_mpi (double dt, double df, size_t if_ini, size_t
    numf_tutti, int nsim, int rank, int size, MPI_Comm comm,
9         double *buffer_stampe, size_t dim_buffer, size_t nmezzi
    , size_t npunti, int sim_completa);
10 #else
11 int main_mpi (double dt, int rank, int size, double *
    buffer_stampe,
12          size_t dim_buffer, size_t npunti, struct_point_cell *
    probed_points,
13          size_t num_point_probed, const gsl_vector * v, double
    *p, int nsim);
14 int fft_mpi (double dt, double df, size_t if_ini, size_t
    numf_tutti, int nsim, int rank, int size,
15         double *buffer_stampe, size_t dim_buffer, size_t nmezzi
    , size_t npunti, int sim_completa);
16 #endif
17 /* calcola il valore medio di un array (v) di n elementi */
18 double
19 media (double *v, unsigned long int n)
20 {
21   double m = (double) 0;  // valore medio
22   unsigned long int i;
23 //#pragma omp parallel for default(none) shared(n,v) private(i
    ) reduction(+:m)
24   for (i = 0; i < n; i++)
25     m += v[i];
26   m = m / (double) n;
27   return (m);
28 }
29
30 /* calcola la varianza di un array (v) di n elementi, una
    volta che la media e' gia' stata calcolata */
31 double
32 varianza_noti (double *v, double m, unsigned long int n)
33 {
34   double va, appo = 0;
35   unsigned long int i;
36 //#pragma omp parallel for default(none) shared(n,v) private(i
```

```
    ) reduction(+:appo)
37  for (i = 0; i < n; i++)
38    appo += v[i] * v[i];
39  va = (appo - m * m * n) / ((double) n - (double) 1);
40  return (va);
41 }
42
43 double
44 func_S21 (const gsl_vector * v, void * params)
45 {
46  int i, ind_f, nf, sim_completa, nsim, tipo = 0;
47  double f_min_ott, f_max_ott;  // Banda scelta per l'
     ottimizzazione
48  double f_min = (double) RCFDTD_FREQ_MIN, f_max = (double)
     RCFDTD_FREQ_MIN;  // Banda (piu' larga) per calcolare il
     parametro S21
49  double S21, *s21_lin, penalizzazione = 1;
50  double *p = (double *) params;
51  double S21_medio = 0, S21_varianza = 0;
52  double appo;
53  /* ***************  DEFINIRE IL PUNTO DA OTTIMIZZARE
     *************** */
54  int punto = 32;
55  /*
     ****************************************************************
      */
56
57  tipo = (int) RCFDTD_TIPO;
58  sim_completa = (int) (p[0] + 0.5);  // Simulazione completa
     prima o dopo l'ottimizzazione con la stampa dell'S21 su
     tutta la banda
59  nf = (int) (p[1] + 0.5);  // Numero di frequenze da
     utilizzare per il calcolo dell'S21 durante l'ottimizzazione
60  f_min_ott = p[2];
61  f_max_ott = p[3];
62  nsim = (int) (p[8] + 0.5);  // Numero della simulazione
     durante le iterazioni quando stampiamo anche i risultati
63  nsim++;
64  p[8]=nsim;
65
66  int rank, size;   /* id del processo e numero di processi
     MPI */
67  double dt, df;
68  struct_point_cell *probed_points;
69
70  /* Per il buffer di stampa o dei campi calcolati da
     scambiare con i vari sottoprogrammi */
71  double *buffer_stampe;
72  double *S21_mod;
73  double complex E[3];
74  size_t dim_buffer = 0, npunti = 0, npunticampo = 0,
     count_stampe = 0, num_point_probed = 0;
```

```
75   size_t ind_0, ind_1, ind0_tutti, numf_tutti, delta, nmezzi;
76   size_t num_f, fpp;
77   char stringa[255];
78
79 #ifdef LIBMPI
80   MPI_Comm comm = MPI_COMM_WORLD;
81   MPI_Init (&argc, &argv);  /* starts MPI *//* ***********
      CONTROLLARE argc e argv se usiamo mpi ************** */
82   MPI_Comm_rank (comm, &rank);  /* get current process id */
83   MPI_Comm_size (comm, &size);  /* get number of processes */
84 #else
85   rank = 0;
86   size = 1;
87 #endif
88   /* Allocate the structure for the coordinate of the probed
      points */
89   probed_points = (struct_point_cell *) malloc (sizeof (
      struct_point_cell) * MAX_DIM_POINTS);
90   for (i = 0; i < MAX_DIM_POINTS; i++)
91     {
92       probed_points[i].i = 0;
93       probed_points[i].j = 0;
94       probed_points[i].k = 0;
95     }
96 /*
97   sprintf (input_file, "./input/points_in_%.3d.txt",
      num_fields);
98   if ((fin = fopen (input_file, "r")) == NULL)
99     {
100      printf ("rank = %d : the input file: `%s` does not exist
      . Simulation aborted\n", rank, input_file);
101      exit (1);
102    }
103  ind_campi = 0;
104  while ((!feof (fin) && (ind_campi < MAX_DIM_POINTS)))
105    {
106      fscanf (fin, "%lg %lg %lg", &x, &y, &z);
107      probed_points[ind_campi].i = (int) (x / (double)
      RCFDTD_DELTA_BASE_X);
108      probed_points[ind_campi].j = (int) (y / (double)
      RCFDTD_DELTA_BASE_Y);
109      probed_points[ind_campi].k = (int) (z / (double)
      RCFDTD_DELTA_BASE_Z);
110      ind_campi++;
111    }
112  num_point_probed = (size_t) ind_campi - (size_t) 1;
113  fclose (fin);
114  printf ("rank = %d : probed point input file: `%s`. Number
      of points: %zd\n", rank, input_file, num_point_probed);
115 */
116
117  /* Se ce la fa non stampa i dati nel tempo ma manda il file
```

```
    di buffer all'FFT
118   * e stampa direttamente i dati in frequenza o meglio ancora
       calcola il valore di S21 da ottimizzare */
119  /* Griglia di stampa:
120   * nx*ny*nz*3: la griglia (per: Ex, Ey, Ez); 2: Rifl Ant Tx
    e Trasm Ant Rx
121   * oppure:
122   * nx*ny*nz*6: la griglia (per: Ex, Ey, Ez, Hx, Hy, Hz); 2:
    Rifl Ant Tx e Trasm Ant Rx */
123
124  if (!sim_completa)
125    npunticampo = 1;    // Punti sulla griglia da tracciare
    durante le iterazioni temporali
126  else
127    npunticampo = (size_t) PRINT_NX * (size_t) PRINT_NY * (
    size_t) PRINT_NZ;    // Punti sulla griglia da tracciare
    durante le iterazioni temporali
128 /*    npunticampo = (size_t) MAX_DIM_POINTS; */   // Punti
    sulla griglia da tracciare durante le iterazioni temporali
129  /* Other two points for the voltage in Tx and Rx
    transmission lines or V and I of the same line */
130  //npunti = npunticampo * (size_t) 3 + (size_t) 2; // Per
    ogni istante di tempo salva questi punti per fare l'FFT
131  /* Without lines */
132  npunti = npunticampo * (size_t) 3;  // Per ogni istante di
    tempo salva questi punti per fare l'FFT
133  dim_buffer = (size_t) 245000; /* deve essere maggiore dei
    passi temporali FDTD */
134  nmezzi = dim_buffer / 2;
135  buffer_stampe = (double *) malloc (sizeof (double) *
    dim_buffer * npunti);
136  for (count_stampe = 0; count_stampe < dim_buffer * npunti;
    count_stampe++)
137    buffer_stampe[count_stampe] = (double) 0;
138
139  if (!(rank))
140    {
141      /* Se non c'e', crea le directory dove scrive i
    risultati e i log del programma */
142      sprintf (stringa, "./risultati");
143      mkdir (stringa, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
144      sprintf (stringa, "./animazioni");
145      mkdir (stringa, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
146    }
147 #ifdef LIBMPI
148  MPI_Barrier (MPI_COMM_WORLD); /* altrimenti potrebbe
    scrivere files in directory che ancora non esistono */
149 #endif
150
151  /* Calcolo e controllo degli indici per la trasformata e
    della banda di fequenze da analizzare */
152  dt = (double) RCFDTD_DELTA_T;
```

```
153   df = (double) 1 / (dt * FFT_N); /* Passo in frequenza */
154 /* La simulazione la fa sempre sulla banda piu' larga, impulso
       nel tempo piu' corto */
155   if (sim_completa)
156     {
157       /* ind_0 = (size_t) ((double) RCFDTD_FREQ_MIN / df);
      *//* Indice dell'array corrispondente a freq_min */
158       /* ind_1 = (size_t) ((double) RCFDTD_FREQ_MAX / df);
      *//* Indice dell'array corrispondente a freq_max */
159     }
160 //  else
161 //    {
162 //      ind_0 = (size_t) ( f_min_ott / df);     /* Indice dell
      'array corrispondente a freq_min */
163 //      ind_1 = (size_t) ( f_max_ott / df);     /* Indice dell
      'array corrispondente a freq_max */
164 //    }
165 //  num_f = ind_1 - ind_0;      /* Numero di valori da
      prendere all'interno dell'intervallo scelto */
166   ind_0 = (size_t) ((double) RCFDTD_FREQ_MIN / df); /* Indice
      dell'array corrispondente a freq_min */
167   ind_1 = (size_t) ((double) RCFDTD_FREQ_MAX / df); /* Indice
      dell'array corrispondente a freq_max */
168   num_f = ind_1 - ind_0;  /* Numero di valori da prendere all'
      interno dell'intervallo scelto */
169   /* Ogni processore mpi si divide le frequenze da analizzzare
       per la statistica
170    * Si estende un po' l'intervallo delle frequenze da
      analizzare in modo che sia multiplo di 'size' */
171   if (num_f % size)
172     fpp = num_f / size + 1; /* frequenze che sono trattate da
      ciascun processo mpi (se il resto non e' nullo) */
173   else
174     fpp = num_f / size;   /* frequenze che sono trattate da
      ciascun processo mpi (se il resto e' nullo) */
175   numf_tutti = fpp * size;  /* totale delle frequenze che
      faranno l'analisi statistica */
176   delta = (numf_tutti - num_f) / 2; /* Meta' delle frequenze
      esterne alla banda da trattare */
177   ind0_tutti = ind_0 - delta; /* Primo indice dell'array da
      analizzare con l'estensione in frequenza */
178   /* Controllo indici. ind0_tutti<0 e' implicito perche'
      definito come size_t */
179   if ((ind0_tutti + numf_tutti) > (dim_buffer / 2))
180     {
181       printf ("Richiesto un numero di frequenze (%ld) troppo
      grande (max: %ld) \n",
182           (long int) (ind0_tutti + numf_tutti), (long int) (
      dim_buffer / 2));
183       free (buffer_stampe);
184       free (probed_points);
185 #ifdef LIBMPI
```

```
186       MPI_Finalize ();
187 #endif
188       exit (EXIT_FAILURE);
189     }
190
191   /* Controllo indici. Il numero di punti sui quali fare l'FFT
        deve essere maggiore dei passi temporali FDTD */
192   if (dim_buffer > (size_t) FFT_N)
193     {
194       if (!(rank))
195   {
196     printf ("Richiesto un numero di punti per l'FFT (%zu)
      troppo piccolo (min: %zu) \n", (size_t) FFT_N,
197       dim_buffer);
198   }
199 #ifdef LIBMPI
200       MPI_Finalize ();
201 #endif
202       exit (EXIT_FAILURE);
203     }
204
205   buffer_stampe = (double *) malloc (sizeof (double) *
      dim_buffer * npunti);
206   for (count_stampe = 0; count_stampe < dim_buffer * npunti;
      count_stampe++)
207     buffer_stampe[count_stampe] = (double) 0;
208
209   if (!rank)
210     {
211       /* Se non c'e', crea le directory dove scrive i
      risultati e i log del programma */
212       sprintf (stringa, "./risultati");
213       mkdir (stringa, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
214       /* Se non c'e', crea le directory dove scrive le
      eventuali animazioni */
215       sprintf (stringa, "./animazioni");
216       mkdir (stringa, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
217     }
218 #ifdef LIBMPI
219   MPI_Barrier (MPI_COMM_WORLD); /* altrimenti potrebbe
      scrivere files in directory che ancora non esistono */
220 #endif
221
222   /* Primo passo: FDTD */
223 #ifdef LIBMPI
224   main_mpi (dt, rank, size, comm, &buffer_stampe[0],
      dim_buffer, npunti, probed_points, num_point_probed, v, p,
      nsim);
225 #else
226   printf ("Chiamata alla FDTD per nsim = %d\n", nsim);
227   main_mpi (dt, rank, size, &buffer_stampe[0], dim_buffer,
      npunti, probed_points, num_point_probed, v, p, nsim);
```

```
228 #endif
229   if (!rank && sim_completa)
230     printf ("Modulo FDTD terminato\n");
231   /* Secondo passo: FFT */
232 if (tipo == 1)
233 {
234 #ifdef LIBMPI
235   fft_mpi (dt, df, ind0_tutti, numf_tutti, nsim, rank, size,
      comm, buffer_stampe, dim_buffer, nmezzi, npunti,
      sim_completa);
236 #else
237   fft_mpi (dt, df, ind0_tutti, numf_tutti, nsim, rank, size,
      buffer_stampe, dim_buffer, nmezzi, npunti, sim_completa);
238 #endif
239   if (!rank && sim_completa)
240     printf ("Modulo FFT terminato\n");
241 }
242 /* Parte per il calcolo di S21 medio */
243 /* La simulazione la fa sempre sulla banda piu' larga */
244   if (sim_completa)
245     {
246       ind_0 = (size_t) ((double) RCFDTD_FREQ_MIN / df); /*
      Indice dell'array corrispondente a freq_min */
247       ind_1 = (size_t) ((double) RCFDTD_FREQ_MAX / df); /*
      Indice dell'array corrispondente a freq_max */
248     }
249   else
250     {
251       ind_0 = (size_t) (f_min_ott / df);  /* Indice dell'array
      corrispondente a freq_min */
252       ind_1 = (size_t) (f_max_ott / df);  /* Indice dell'array
      corrispondente a freq_max */
253     }
254   num_f = ind_1 - ind_0;  /* Numero di valori da prendere all'
      interno dell'intervallo scelto */
255   S21_mod = (double *) malloc (sizeof (double) * num_f);
256 #pragma omp parallel for default(none) \
257           shared(buffer_stampe, num_f, ind_0, npunti, nmezzi,
      appo, S21_mod, punto) \
258           private(i, ind_f, E) \
259           num_threads(RCFDTD_THREADS)
260   for (ind_f = 0; ind_f < num_f; ind_f++)
261     {
262       appo=0;
263       //for (i = npunti - 1; i < npunti; i++) // Valori di S21
      nell'ultimo punto (o colonna)
264       for (i = punto * 3; i < (punto + 1) * 3; i++) // I 3
      valori del punto sono Ex, Ey e Ez parte reale ed
      immaginaria
265   {
266           /* primo punto E[0] e' Ex,  secondo E[1] e' Ey,
      terzo E[2] e' Ez */
```

```
267             E[i - punto * 3] = buffer_stampe[ind_f * npunti + i]
      + I * buffer_stampe[(ind_f + nmezzi) * npunti + i];
268   }
269         appo = cabs (csqrt (cpow(E[0], 2.0 + I * 0.0) + cpow(E
      [1], 2.0 + I * 0.0) + cpow(E[2], 2.0 + I * 0.0)));
270         S21_mod[ind_f] = sqrt(appo);
271       }
272   S21_medio = media (S21_mod, (unsigned long int) num_f);
273   S21_varianza = varianza_noti (S21_mod, S21_medio, (unsigned
      long int) num_f);
274   penalizzazione /= (S21_varianza / 10.0 + 1.0);  /* Per
      evitare di avere oscillazioni troppo elevate in banda */
275   S21 = 20.0 * log10 (S21_medio) * penalizzazione;
276   //if (!rank && sim_completa)
277   if (!rank)
278     printf ("Calcolo dell'S21 terminato (S21 pesato e mediato
      sulla banda = %g \n", S21);
279
280   /* Libera array */
281   free (buffer_stampe);
282   free (probed_points);
283   free (S21_mod);
284 #ifdef LIBMPI
285   MPI_Finalize ();
286 #endif
287   return (-S21);
288 }
```

This code defines several functions and two main functions: main mpi() and fft mpi(). The code contains preprocessor directives to select which version of the functions to use depending on whether the code is compiled with or without MPI support. The media() function calculates the mean value of an array of n elements. The varianza noti() function calculates the variance of an array of n elements, given that the mean value m is already calculated. The func S21() function is the function that is optimized using a nonlinear optimization algorithm. The function takes a vector of variables (v) as input and returns a scalar value that is the objective function to be minimized. The objective function calculates the S21 parameter for a given set of variables and compares it to a desired value. The variables that are optimized include the frequency range and the power levels used in the simulation. The S21 parameter is the ratio of the transmitted power to the incident power in a two-port network. The main mpi() function is the main function that is used when the code is compiled with MPI support. The function takes several arguments that are used to distribute the calculation across multiple processes. The function calls the func S21() function to calculate the objective function for each set of variables that are assigned to each process. The fft mpi() function is used to perform a fast Fourier transform (FFT) on a set of data. The function takes several arguments that are used to distribute the calculation across multiple processes when the code

is compiled with MPI support. The function performs the FFT on a subset of the data assigned to each process and then combines the results to produce the final result.
Finally we find the main for optimization where the GSL function is called:

```
1     #include "struttura_dati.h"
2 int
3 main (int argc, char *argv[])
4 {
5   int solo_analisi = 0;   // 1: solo analisi, 0: ottimizza
6   int nx = 10, nz = 10;   // Numero di patch
7   int Dx = 10, Dz = 10;   // Dimensione di una patch
8   int Gx = 4, Gz = 4;   // Dente della croce di una patch
9
10  double fmin = RCFDTD_FREQ_MIN;
11  double fmax = RCFDTD_FREQ_MAX;
12  double S21;
13  double par[1000];   /* parametri che la funzione gsl passa
     alla funzione (fdtd+fft) per il calcolo di S21 */
14  FILE *pfo = NULL, *gd = NULL;
15  char string[255];
16  size_t n_cap = nx * (nz - 1) + nz * (nx - 1);
17  size_t n_group = 10;
18
19  const gsl_multimin_fminimizer_type *T =
     gsl_multimin_fminimizer_nmsimplex2; /* T e' il tipo di
     minimizzazione che la gsl deve fare */
20  gsl_multimin_fminimizer *s = NULL;  // Puntatore al
     minimizzatore
21  gsl_vector *ss, *x;   // Puntatori a 'x' (vettore delle
     capacita' dei diodi) e 'ss' (vettore delle variazioni delle
      capacita' dei diodi)
22  gsl_multimin_function minex_func; // Funzione da ottimizzare
23
24  size_t iter = 0, i;
25  int status, ig;
26  double size = 0;
27  double ris;
28
29 /* Leggi i valori iniziali dei gruppi di diodi */
30  sprintf (string, "./input_pw/diode_group.txt");
31  if ((gd = fopen (string, "r")) == NULL)
32    exit (EXIT_FAILURE);
33
34  /* Initial analisis - call the fdtd with inizial values and
     print the scattering parameters (S21) and fields */
35  // da fare la chiamata alla fdtd e fft con i valori iniziali
      delle capacita' e stampa   ............
36  /* Scan of the whole band */
37
```

```
38   par[0] = 1;
39   par[1] = 10;       /* Per il momento, non usato */
40   par[2] = fmin;
41   par[3] = fmax;
42   par[4] = nx;
43   par[5] = nz;
44   par[6] = Dx;
45   par[7] = Dz;
46   par[8] = 0;      /* Numero per stampa dati della simulazione
      */
47   par[9] = n_cap;    /* Numero di diodi inseriti nella
      simulazione */
48   par[10] = Gx;
49   par[11] = Gz;
50   par[12] = n_group;
51   /* reserved for future use */
52   for (i = 13; i < 100; i++)
53      par[i] = 0;
54   /* Pointer of diode group */
55   for (i = 100; i < 100 + n_cap; i++)
56     {
57       fscanf (gd, "%d", &ig);
58       par[i] = ig;
59     }
60
61   /* reserved for more diodes */
62   for (i = 100 + n_cap; i < 1000; i++)
63      par[i] = 0;
64   fclose (gd);
65
66   printf ("Initial capacitance set. \n");
67
68   x = gsl_vector_alloc (n_group); //Allocazione di un array
      contenenti le capacita' che saranno aggiornate in ogni
      iterazione
69
70 /* Starting point */
71 //printf ("Starting values are in the file: %s\n", string);
72 /* per adesso 1 pF a tutti, poi cambiare */
73   //gsl_vector_set_all (x, (double) 5E-13);
74   gsl_vector_set_all (x, (double) 1E-13);
75   //gsl_vector_set_all (x, (double) 1E-12);
76 /*
77   for (i = 0; i < n_cap; i += 2)
78     gsl_vector_set (x, i, (double) 1E-12);
79
80   for (i = 1; i < n_cap; i += 2)
81     gsl_vector_set (x, i, (double) 1E-13);
82 */
83
84 /* Leggi i valori iniziali delle capacita' */
85   //sprintf (string, "Initial_Capacitances.txt");
```

```
86   //if ((pfi = fopen (string, "r")) == NULL)
87   //exit (EXIT_FAILURE);
88 // ....
89 // ....
90 // ....
91 // ....
92 // ....
93   //fclose (pfi);
94
95   /* Set initial step sizes */
96   ss = gsl_vector_alloc (n_group);  // Allocazione di un array
       della stessa dimensione di x (numero di diodi da
      ottimizzare)
97   gsl_vector_set_all (ss, 0.9E-12); // Variazione della
      capacita' con il quale la gsl cambia i valori delle
      capacit   in un ciclo iterativo
98   /* Initialize method and iterate */
99   minex_func.n = n_group;   // Numero dei gruppi delle
      capacita'
100  minex_func.f = func_S21;  // Nome della funzione da
      ottimizzare
101
102  /* Initial analysis -- with prints */
103  par[0] = 1;
104  par[2] = fmin;
105  par[3] = fmax;
106  minex_func.params = par;  // Altri parametri da passare alla
       funzione da ottimizzare
107  /* Scan of the whole band */
108  // ........
109  // printf ("Computed %d points in the band %g - %g (GHz)\n",
       nfreq, fmin * 1.0E-9, fmax * 1.0E-9);
110  // printf ("Starting results are in the file: %s\n", string)
      ;
111  S21 = func_S21 (x, minex_func.params);
112  printf ("Calcolo dell'S21 terminato (S21 INIZIALE mediato
      sulla banda = %g \n", -S21);
113
114  if (!solo_analisi)
115    {
116
117      /* Iterative analyses -- without prints */
118      par[0] = 0;
119      par[2] = 3.3E9;
120      par[3] = 4.2E9;
121      minex_func.params = par;  // Altri parametri da passare
      alla funzione da ottimizzare
122
123      s = gsl_multimin_fminimizer_alloc (T, n_group);
124      gsl_multimin_fminimizer_set (s, &minex_func, x, ss);
125
126      /* Ciclo di  ottimizzazione */
```

```
127       do
128   {
129
130     iter++;
131     par[8]++;
132     minex_func.params = par;  // Altri parametri da passare
      alla funzione da ottimizzare
133 /* Per controllo durante l'ottimizzazione a ogni ciclo stampa:
134  * - solo tre valori delle capacit   dei diodi (inizio,
      centrale e fine)
135  * - il valore della funzione da ottimizzare (S21 su tutta la
      banda, compresso in un valore double
136  * - il passo usato nella ricerca delle capacita'
137  */
138
139     status = gsl_multimin_fminimizer_iterate (s);
140
141     printf ("Iterazione: %5zu Capacita' = %10.3e %10.3e %10.3e
      (F) -  f() = %.3f size = %.3f\n\n",
142       iter,
143       gsl_vector_get (s->x, 0),
144       gsl_vector_get (s->x, n_group / 2), gsl_vector_get (s->x
     , n_group - 1), -s->fval, size);
145
146     if (status)
147       break;
148     /*
149        if (gsl_vector_get(s->x, 0) <0)
150        continue;
151        if (gsl_vector_get(s->x, 1) <0)
152        continue;
153        if (gsl_vector_get(s->x, 2) <0)
154        continue;
155        if (gsl_vector_get(s->x, 3) <0)
156        continue;
157      */
158 //    if (gsl_vector_get(s->x, 4) <0)
159 //         continue;
160
161     size = gsl_multimin_fminimizer_size (s);
162     status = gsl_multimin_test_size (size, 1e-13); // forse
      cambiare valore per vedere miglioramenti?? es. 10-13
163
164     if (status == GSL_SUCCESS)
165       {
166         printf ("converged to minimum at\n");
167       }
168           printf ("Iterations: %5zu\n", iter);
169           for (i = 0; i < n_group; i++)
170             printf ("Group %5zu -- capacitance: %10.3e (F)\n",
      i, gsl_vector_get (s->x, i));
171           printf ("Final Value: -s->fval -- Size: %.3f\n", -s
```

```
    ->fval, size);
172   }
173       while ((status == GSL_CONTINUE) && (iter < 200)); // 500
       messo a caso, dipende dai parametri da ottimizzare e dal
      grupp di numero dei diodi che abbiamo
174
175       /* E' uscito, salvare i valori delle capacita' dei diodi
       */
176       // ... Capacita' = gsl_vector_get (s->x, 0);
177       // ...
178       // ... Capacita' = gsl_vector_get (s->x, n_cap - 1);
179
180       /* Final analysis -- with prints */
181       par[0] = 1;
182       par[2] = fmin;
183       par[3] = fmax;
184       par[8]++;
185       minex_func.params = par;  // Altri parametri da passare
      alla funzione da ottimizzare
186       /* Scan of the whole band */
187       // ........
188       // printf ("Computed %d points in the band %g - %g (GHz)
      \n", nfreq, fmin * 1.0E-9, fmax * 1.0E-9);
189       // printf ("Starting results are in the file: %s\n",
      string);
190       for (i = 0; i < n_group; i++)
191         gsl_vector_set (x, i, gsl_vector_get (s->x, i));
192       S21 = func_S21 (x, minex_func.params);
193       printf ("Calcolo dell'S21 terminato (S21 FINALE mediato
      sulla banda = %g \n", -S21);
194     }
195
196   /* Structure free */
197
198   gsl_vector_free (x);
199   gsl_vector_free (ss);
200   gsl_multimin_fminimizer_free (s);
201
202   return status;
203 }
```

It is a program that implements an optimization algorithm for minimizing a function. The function to be minimized is defined by the variable minex func and is called func S21. It is not in the code, but is probably defined in a separate file. The minimization algorithm is implemented using the GSL library (GNU Scientific Library). In particular, the method of finding the minimum using the simplex is used. Initially some parameters and variables used in the program are defined, such as the number of patches, the size of a patch, the minimum and maximum value of the frequency, the initial values of the diode capacitances and so on. The program

reads the initial values of the capacities from the input files and allocates the array containing the capacities x, which will be updated in each iteration. Subsequently, the minimization method is initialized and the iterative cycle for the search for the minimum of the function func S21 is started. Within the iterative loop, the func S21 function is called with the current values of the capacities, the value of the function is calculated, and the values of the capacities are updated for the next iteration. The loop ends when the maximum number of iterations is reached or when the simplex size becomes too small. Finally, the results of the minimization are written to the output file.
This part shows the program for the FDTD calculation of 3D structures:

```
1     void metti_ms_basetta (struct_campoEM * campo, struct_fdtd
      * fdtd, struct_griglia * griglia, double dt, struct_ms *
     ms,
2                            const gsl_vector * v, int *point_cap,
     int fl_diemet, int fl_solopos, int fl_solocond, int rank);
3  void posizione_dipolo (struct_campoEM * campo, struct_fdtd *
     fdtd, struct_griglia * griglia, struct_dipolo * antenna,
4             int tilted);
5  int calcola_fdtd (long int time, double dt, struct_campoEM *
     campo_g1, struct_fdtd * fdtd_g1, struct_griglia *
     griglia_g1,
6                    struct_eccitazione * ecc, struct_piani_sep *
      campi_sep, struct_dim_sep * sep, struct_point_meter *
     origine, struct_onda_piana * op,
7                    int rank);
8  int InizializzaDati_CaricaConf (double dt, struct_griglia *
     griglia_g1, struct_campoEM * campo_g1, struct_campo_max *
     campo_max_g1,
9                                   struct_fdtd * fdtd_g1,
     struct_eccitazione * ecc, struct_onda_piana * op,
10                                  struct_piani_sep * campi_sep,
     struct_dim_sep * sep, struct_point_meter * origine,
     struct_ms * ms,
11                                  double sigma, const gsl_vector
      * v, double * p, int *point_cap, int rank);
12 int stampa_matrix_time_bin (int id_out, char srsim[255], char
     srdir[255], struct_campoEM * campo_tot, double *appo,
13            size_t dimv, int passo, int i_ini, int i_fi, int
     j_ini, int j_fi, int k_ini, int k_fi);
14 #ifdef FDTD_PRINT_TIME
15 void stampa_tempo (double dt, long int time, FILE * correnti,
     struct_campoEM * campo_tot, struct_linea * linea_Tx,
16         struct_linea * linea_Rx);
17 void stampa_tempo_bin (double dt, long int time, FILE *
     correnti, struct_campoEM * campo_tot, struct_linea *
     linea_Tx,
18             struct_linea * linea_Rx);
19 #endif
```

```
20 size_t stampa_buffer (double dt, long int time, struct_campoEM
       * campo_tot, struct_linea * linea_Tx,
21             struct_linea * linea_Rx, double *buffer_stampe,
     size_t count_stampe, size_t npunti);
22 int stampa_max (int flag_x, int flag_y, int flag_z, int flag_t
    , struct_campo_max * campo_max, struct_griglia * griglia,
23     struct_nomi * nomem);
24 double stampa_new (struct_griglia * griglia,
     struct_eccitazione * ecc, double dt);
25 int gethostname (char *name, size_t len);
26
27 #ifdef LIBMPI
28 int
29 main_mpi (double dt, int rank, int size, MPI_Comm comm, double
       *buffer_stampe, size_t dim_buffer, size_t npunti,
30     struct_point_cell * probed_points, size_t num_point_probed
    , const gsl_vector * v, double *p, int nsim)
31 #else
32 int
33 main_mpi (double dt, int rank, int size, double *buffer_stampe
    , size_t dim_buffer, size_t npunti,
34     struct_point_cell * probed_points, size_t num_point_probed
    , const gsl_vector * v, double *p, int nsim)
35 #endif
36 {
37   /* Erano globali ora sono diventate locali, i valori sono
       passati come parametro o puntatori alle funzioni */
38   struct_campoEM campo_g1;
39   struct_campo_max campo_max_g1;
40   struct_fdtd fdtd_g1;
41   struct_eccitazione ecc;
42   struct_griglia griglia_g1;
43   /* Variabili per la separazione campo totale - campo
       riflesso - campo incidente */
44   struct_piani_sep campi_sep;
45   struct_dim_sep sep;
46   struct_point_meter origine;
47   struct_onda_piana op;
48   /* Variabili per simulare un diodo a elementi concentrati */
49   //struct_per_diodo diode;
50   /* Variabili per simulare un induttore a elementi
       concentrati */
51   //struct_per_induttore induttore;
52   struct_ms ms;
53   char sr_sar[255];
54   /* */
55   double tprev_tot = 0.0;
56   double volume;
57
58   /* variabili ausiliarie */
59   long int count_anim;
60   long int anim_passo = 0;
```

```
61   int sim_completa, icap, n_cap, n_group, *point_cap = NULL;
62
63   size_t count_stampe, stampe_step = 1000;
64   long int cont1, cont2;
65   long int time = 0;
66   int len = 255, ris;
67   char hn[len], nt[12];
68   char srsim_anim[255];
69   /* char srsim[255], stringa[255], srdir[255]; */
70   int nx, ny, nz;
71 #ifdef FDTD_PRINT_TIME
72   FILE *correnti = NULL;
73 #endif
74   sim_completa = (int) (p[0] + 0.5);  // Simulazione completa
      prima o dopo l'ottimizzazione con la stampa dell'S21 su
      tutta la banda
75   /* nsim = (int) (p[8] + 0.5); */// Numero della simulazione
      per stampe. - Definita in `supermain` e passata come
      parametro
76   n_cap = (int) (p[9] + 0.5); /* Numero di diodi utilizzati */
77   n_group = (int) (p[12] + 0.5); /* Numero di gruppi di diodi
      utilizzati */
78
79   /* Pointer diode(i) -> group_of_capacitances(j) */
80   point_cap = (int *) malloc (sizeof (int *) * n_cap);
81       for (icap = 0; icap < n_cap; icap++)
82       {
83               point_cap[icap] = (int) (p[100 + icap] + 0.5);
84       }
85
86 printf ("*****\n*****\n*****\n");
87 printf ("Inizio simulazione FDTD, per `nsim` = %d (`
     sim_completa`: %d)\n", nsim, sim_completa);
88 printf ("*****\n");
89
90 /* Stampa i valori delle capacita` dei diodi su file */
91   //if (sim_completa)
92     {
93       char nome_cap[255];
94       FILE *filcap = NULL;
95       sprintf (nome_cap, "./risultati/Capacita_%s%.3d_%.4d",
     PREFISSO_SIM, LABEL_SIM, nsim);
96       if ((filcap = fopen (nome_cap, "w")) == NULL)
97     exit (EXIT_FAILURE);
98       for (icap = 0; icap < n_cap; icap++)
99       {
100              fprintf (filcap, "%e\n",gsl_vector_get (v,
     point_cap[icap]) );
101      }
102      fclose (filcap);
103    }
104
```

```
105 #ifdef FDTD_PRINT_TIME_FIELDS
106   struct_nomi nomet;
107   if (sim_completa)
108     {
109       sprintf (nomet.exmax, "./risultati/CampiTempo_Ex_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
110       sprintf (nomet.eymax, "./risultati/CampiTempo_Ey_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
111       sprintf (nomet.ezmax, "./risultati/CampiTempo_Ez_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
112       sprintf (nomet.emax, "./risultati/CampiTempo_MagE_%s%.3
     d_%.4d", PREFISSO_SIM, LABEL_SIM, nsim);
113     }
114 #endif
115 #ifdef FDTD_PRINT_MAX_FIELDS
116   struct_nomi nomem;
117   if (sim_completa)
118     {
119       sprintf (nomem.exmax, "./risultati/CampiMax_Ex_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
120       sprintf (nomem.eymax, "./risultati/CampiMax_Ey_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
121       sprintf (nomem.ezmax, "./risultati/CampiMax_Ez_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
122       sprintf (nomem.emax, "./risultati/CampiMax_MagE_%s%.3d_
     %.4d", PREFISSO_SIM, LABEL_SIM, nsim);
123     }
124 #endif
125   if (sim_completa)
126     {
127       sprintf (sr_sar, "./risultati/SAR_Wkg_%s%.3d_%.4d",
     PREFISSO_SIM, LABEL_SIM, nsim);
128     }
129
130   /* Non salva i campi in `npunti` punti ad ogni istante
     temporale ma li bufferizza e poi li manda a hdf ogni `
     dim_buffer`
131    * istanti.  Il valore di `dim_buffer` dipende anche da `
     npunti` e deve essere scelto per rendere i tempi di stampa
132    * i piu` brevi possibili */
133   /* Griglia di stampa */
134   if (!sim_completa)
135     {
136       nx = 1;
137       ny = 1;
138       nz = 1;
139     }
140   else
141     {
142       nx = PRINT_NX;
143       ny = PRINT_NY;
144       nz = PRINT_NZ;
```

```
145     }
146   /* nx*ny*nz*3: la griglia (per: Ex, Ey, Ez); 1: Ant probe */
147   /* nx*ny*nz*6: la griglia (per: Ex, Ey, Ez, Hx, Hy, Hz); 1:
      Ant probe */
148   /* nx*ny*nz*4: la griglia (per: Ex, Ez, Hx, Hz); 1: Ant
      probe - Fields on an aperture */
149     //if (((size_t) MAX_DIM_POINTS * 3) != npunti)
150     if (( nx * ny * nz * 3) != npunti)
151       {
152         //printf ("Dimensione del buffer di stampa sbagliato:
      punti letti = %zd, npunti = %zd (rank: %d)\n", (size_t)
      MAX_DIM_POINTS * 3, npunti, rank);
153         printf ("Dimensione del buffer di stampa sbagliato:
      punti letti = %zd, npunti = %zd (rank: %d)\n", (size_t) nx
      * ny * nz * 3, npunti, rank);
154         exit (EXIT_FAILURE);
155       }
156
157 #ifdef FDTD_PRINT_TIME
158   sprintf (srsim, "./risultati/TD_%s%.3d_%.4d", PREFISSO_SIM,
      LABEL_SIM, nsim);
159   /*if (!rank) */
160   printf ("File dati dove saranno scritti i risultati della
      simulazione: %s\n", srsim);
161 #endif
162
163   ris = gethostname (hn, len);
164   if (!rank && sim_completa)
165     printf ("PC dove le simulazioni saranno eseguite: %s\n",
      hn);
166   strncpy (nt, hn, 8);
167   nt[8] = '\0';
168
169   /* Calcola la dimensione dell'array per le stampe temporali
      */
170   /*
171      prt_dimv =
172      ((prt_k_fi - prt_k_ini - 1) / prt_passo + 1) * ((prt_j_fi
       - prt_j_ini - 1) / prt_passo +
173      1) * ((prt_i_fi - prt_i_ini - 1) / prt_passo + 1) * 3;
174    */
175   /* Alloca l'array di appoggio per la stampa dei campi nel
      tempo */
176   /*
177      prt_appo = (double *) malloc (sizeof (double) * prt_dimv)
      ;
178      for (count = 0; count < prt_dimv; count++)
179      prt_appo[count] = 0;
180    */
181
182   if (InizializzaDati_CaricaConf (dt, &griglia_g1, &campo_g1,
      &campo_max_g1, &fdtd_g1, &ecc, &op, &campi_sep, &sep, &
```

```
    origine, &ms, 0.0, v, p, point_cap, rank))
183     {
184       if (!rank)
185   puts ("CONTROLLARE LA MEMORIA O IL FILE DI CONFIGURAZIONE -
     ERRORE\n");
186 #ifdef LIBMPI
187       MPI_Finalize ();
188 #endif
189       return (EXIT_FAILURE);
190     }
191   if (!rank && sim_completa)
192     mostra_info ();
193
194   /* CHIAMATA A COSTANTI_LUN NIENTE FLASG */
195   /*printf("Chiamate a costanti_lun\n"); */
196   costanti_lun (dt, &griglia_g1, &ecc);
197
198   /* CHIAMATA A COSTANTI */
199   /*printf("Chiamata a costanti_form_e_var (g)\n"); */
200   costanti_form_e_var (dt, &fdtd_g1, &griglia_g1);
201   /*printf("Chiamata a costanti_form_h_cos (g)\n"); */
202   costanti_form_h_cos (dt, &fdtd_g1, &griglia_g1);
203
204   if (!costanti_mur2 (dt, &fdtd_g1, &griglia_g1))
205     {
206       puts ("CONTROLLARE LA MEMORIA O IL FILE DI
     CONFIGURAZIONE - ERRORE IN (costanti_mur2)\n");
207       exit (EXIT_FAILURE);
208     }
209   //printf("Chiamata a costanti_abc (g)\n");
210   if (!costanti_abc (dt, &fdtd_g1, &griglia_g1))
211     {
212       puts ("CONTROLLARE LA MEMORIA O IL FILE DI
     CONFIGURAZIONE - ERRORE IN (costanti_abc)\n");
213       exit (EXIT_FAILURE);
214     }
215
216   /* !!! da richiamare dopo la chiamata a costanti !!! */
217   /* geometria dell'oggetto da simulare */
218   /* Da richiamare 4 volte:
219    * Prima per le dimensioni
220    * Seconda per il dielettrico
221    * Terza per il metallo
222    * Quarta per i condensatori
223    * Qui richiamo per il metallo */
224    metti_ms_basetta (&campo_g1, &fdtd_g1, &griglia_g1, dt, &ms
     , v, point_cap, 1, 0, 0, rank);
225   /* Da richiamare 4 volte:
226    * Prima per le dimensioni
227    * Seconda per il dielettrico
228    * Terza per il metallo
229    * Quarta per i condensatori
```

```
230     * Qui richiamo per i condensatori */
231     metti_ms_basetta (&campo_g1, &fdtd_g1, &griglia_g1, dt, &ms
        , v, point_cap, 0, 0, 1, rank);
232
233    //metti_condensatore (34 + RCFDTD_DISTSEP + RCFDTD_DISTOGG,
        10 + RCFDTD_DISTSEP + RCFDTD_DISTOGG, 1 + RCFDTD_DISTSEP +
        RCFDTD_DISTOGG, dt, (double) 1E-15, &fdtd_g1, &griglia_g1,
        0);
234    //metti_induttore(induttore.i0, induttore.j0, induttore.k0,
        dt, &induttore, &fdtd_g1, &griglia_g1, 0);
235
236    /* posizione delle superfici metalliche */
237    /* Piano metallico tra l'antenna della BS e il telefonino */
238    //int spost;
239    //spost = (int) RCFDTD_DISTSEP + (int) RCFDTD_DISTOGG;
240    //crea_quad_pec (&campo_g1, &fdtd_g1, &griglia_g1, spost,
        spost + 100, spost, spost + 100, griglia_g1.dimym1 / 2, 1);
241
242    //
243    /* da richiamare dopo la chiamata a costanti */
244    /* posizione delle superfici metalliche */
245
246    /* Libera gli array per epsr */
247    d_libera_3d (griglia_g1.epsr, griglia_g1.dimx, griglia_g1.
        dimy);
248
249    /*printf("Chiamate a stampa_new\n"); */
250    if (!rank && sim_completa)
251      tprev_tot = stampa_new (&griglia_g1, &ecc, dt);
252
253 #ifdef FDTD_PRINT_TIME
254    /* Apre il file per scrivere i dati */
255    if ((correnti = fopen (srsim, "w")) == NULL)
256      exit (EXIT_FAILURE);
257    if (!rank && sim_completa)
258      {
259        printf
       ("================================================\n");
260        printf ("Dimensioni dei file binari dei punti nel tempo
       :\n");
261        printf ("Tipo di dato: double\n");
262        printf ("Numero di righe: %ld\n", (long int) ecc.tfinale
       );
263        printf ("Numero di colonne: %d\n", (nx * ny * nz * 3));
264        printf
       ("================================================\n");
265      }
266 #else
267    /* Il buffer deve essere lungo per tutti i cicli temporali
       */
268    /* ************* SCOMMENTARE SOLO PER DEBUG ************* */
269    /* ecc.tfinale=3000; */
```

```
270   if (ecc.tfinale > (double) dim_buffer)
271     {
272       if (!rank && sim_completa)
273   printf ("Dimensione del buffer di stampa sbagliato:
       dim_buffer = %zd, iterazioni = %ld  (rank: %d)\n",
274     dim_buffer, (long int) ecc.tfinale, rank);
275       exit (EXIT_FAILURE);
276     }
277 #endif
278 #ifdef FDTD_PRINT_GEO
279   /*if (!rank) */
280   if ((griglia_g1.delta_costante) && (griglia_g1.cella_cubo)
       && (griglia_g1.mezzo_omogeneo))
281     {
282       fdtd_print_geo (&griglia_g1, &fdtd_g1, rank);
283     }
284 #endif
285   count_anim = 0;
286   anim_passo = (long int) ecc.tfinale / (long int) 8000;
287   if (!rank && sim_completa)
288     printf ("The animation is printed for a frame every %ld
       time steps\n", anim_passo);
289 #ifdef DEBUG
290   stampe_step = 1;
291 #else
292   stampe_step = 1000;
293 #endif
294   count_stampe = 0;
295   cont1 = 1;
296   cont2 = 1;
297   if (!rank && sim_completa)
298     puts ("Avanzamento: ");
299 #ifdef DEBUG
300   int ip = 55, jp = 48, kp = 27;
301 #endif
302   /* INIZIO ITERAZIONE TEMPORALE */
303   for (time = 1; time < ecc.tfinale; time++, cont1++, cont2++)
304     {
305       /*printf("************************************
       Iterazione: %ld\n",time); */
306
307       if (calcola_fdtd (time, dt, &campo_g1, &fdtd_g1, &
       griglia_g1, &ecc, &campi_sep, &sep, &origine, &op, rank))
308   {
309     if (!rank)
310       puts ("-----  ERRORE DURANTE LA SIMULAZIONE  -----");
311     return (EXIT_FAILURE);
312   }
313 #ifdef DEBUG
314       printf ("t,i,j,k, Ex,Ey,Ez,Hx,Hy,Hz: %d,%d,%d,%d %g,%g,%
       g,%g,%g,%g\n", time, ip, jp, kp, campo_g1.Ex[ip][jp][kp],
315         campo_g1.Ey[ip][jp][kp], campo_g1.Ez[ip][jp][kp],
```

```
    campo_g1.Hx[ip][jp][kp], campo_g1.Hy[ip][jp][kp],
316         campo_g1.Hz[ip][jp][kp]);
317 #endif
318       if (cont1 == stampe_step)
319   {
320     cont1 = 0;
321     if (!rank && sim_completa)
322       printf ("Iterazioni: %ld\n", time);
323   }
324
325       switch (ecc.tipo)
326   {
327   case 1:
328     count_stampe =
329             stampa_buffer_delta_nolinea (dt, time, &campo_g1,
    buffer_stampe, count_stampe, npunti, nx, ny, nz);
330 #ifdef FDTD_PRINT_TIME
331     if ((dim_buffer == count_stampe) && sim_completa)
332       {
333         fwrite (&buffer_stampe[0], sizeof (double),
    count_stampe * npunti, correnti);
334         count_stampe = 0;
335       }
336 #endif
337     /* if (!(time % anim_passo) && (time < anim_passo * 2000))
     */
338     /* if ((time>((long int) 175000)) && (time<((long int)
    181666 )) && ( !(time%4))) */
339     /* if ((time>((long int) 183776 - (long int) 45000)) && (
    time<((long int) 183776 - (long int) 5000)) && ( !(time%16)
    )) */
340     /* if ((time>((long int) 367552 - (long int) 45000)) && (
    time<((long int) 367552 - (long int) 5000)) && ( !(time%16)
    )) */
341     if ((time < 4000) && (time > 0) && (!(time % 4)) &&
    sim_completa)
342       {
343         /* $$$$$$$$ Codice per scrittura file risultati ad un
    istante di tempo */
344         sprintf (srsim_anim, "./animazioni/ANIM_X_%s%.3d_%.4d_
    %.5ld", PREFISSO_SIM, LABEL_SIM,
345             nsim, count_anim);
346         stampa_matrix_animazione_xcost (srsim_anim, &campo_g1,
     griglia_g1.dimx / 2, 0, griglia_g1.dimy - 1, 0,
347                 griglia_g1.dimz - 1);
348         sprintf (srsim_anim, "./animazioni/ANIM_Y_%s%.3d_%.4d_
    %.5ld", PREFISSO_SIM, LABEL_SIM,
349             nsim, count_anim);
350                  stampa_matrix_animazione_ycost (srsim_anim, &
    campo_g1, griglia_g1.dimy / 2, 0, griglia_g1.dimx - 1, 0,
    griglia_g1.dimz - 1);
351         sprintf (srsim_anim, "./animazioni/ANIM_Z_%s%.3d_%.4d_
```

```
    %.5ld", PREFISSO_SIM, LABEL_SIM,
352             nsim, count_anim);
353                   stampa_matrix_animazione_zcost (srsim_anim, &
    campo_g1, griglia_g1.dimz / 2, 0, griglia_g1.dimx - 1, 0,
    griglia_g1.dimy - 1);
354         count_anim++;
355       }
356      if (!(time % 10) && (time > 1) && (time < 5001) &&
    sim_completa)
357       {
358         /* $$$$$$$$ Codice per scrittura file risultati ad un
    istante di tempo */
359         /* DA FARE
360            sprintf (srdir, "./matrice%.3d/Matrice%.3d_%.3d",
    label_sim, label_sim, ind_anim);
361             ind_anim++;
362             stampa_matrix_time_bin (id_out, srsim, srdir, &
    campo_g1, prt_appo, prt_dimv, prt_passo, prt_i_ini,
363             prt_i_fi, prt_j_ini, prt_j_fi, prt_k_ini, prt_k_fi)
    ;
364           */
365       }
366      break;
367    case 2:
368      count_stampe =
369               stampa_buffer_delta_nolinea (dt, time, &campo_g1,
     buffer_stampe, count_stampe, npunti, nx, ny, nz);
370      break;
371    default:
372      if (!rank)
373        printf ("Tipo di eccitazione non presente\n");
374      break;
375    }
376 #ifdef FDTD_PRINT_TIME_FIELDS
377       if ((time == 375) && (sim_completa))
378    {
379      stampa_campi_tempo (1, 1, 1, 1, &griglia_g1, &campo_g1, 1,
      &nomet);
380    }
381 #endif
382       if (time > ecc.time1)
383    {
384      if (ecc.tipo == 2)
385        {
386          sar (&campo_g1, &campo_max_g1, &griglia_g1);
387        }
388    }
389      }
390    /* Libera gli array dei campi sulla separazione campo Totale
     -Riflesso */
391    Libera_piani_sep (&campi_sep, &griglia_g1);
392 #ifdef FDTD_PRINT_TIME
```

```
393 /* Scrive su HDF i valori dei campi nei punti selezioni
    bufferizzati, ma non ancora scritti */
394   if (count_stampe)
395     {
396       fwrite (&buffer_stampe[0], sizeof (double), count_stampe
    * npunti, correnti);
397     }
398   /* chiudo tutti i file aperti */
399   fclose (correnti);
400 #endif
401 #ifdef FDTD_PRINT_PARS
402 /* Write the transmitted and the reflected signal in the time
    domain
403  *(to be used in alternative to FDTD_PRINT_TIME that prints
    all the grid fields also) */
404   if (sim_completa)
405     fdtd_print_pars (buffer_stampe, ecc.tfinale, dt, npunti,
    rank);
406 #endif
407
408   /* Libera gli array dei campi */
409   Libera_campi (&campo_g1, &griglia_g1);
410 #ifdef FDTD_PRINT_MAX_FIELDS
411   if ((ecc.tipo == 2) && (sim_completa))
412     {
413       //stampa_campi_max (0, 0, 0, 1, &griglia_g1, &
    campo_max_g1, &body_itis, sr_sar, (int) 1, &nomem);
414       //stampa_max (1, 1, 1, 1, &campo_max_g1, &griglia_g1, &
    nome, srsim, hn, nt, id_out);
415       stampa_max (1, 1, 1, 1, &campo_max_g1, &griglia_g1, &
    nomem);
416     }
417 #endif
418   /*free (prt_appo); */
419   /* Libera gli array dei campi massimi */
420   Libera_campi_max (&campo_max_g1, &griglia_g1);
421   /* Libera gli array dei coefficenti */
422   Libera_coef (&fdtd_g1, &griglia_g1);
423   /* Libera gli array della condizione di Mur */
424   Libera_mur2 (&fdtd_g1, &griglia_g1);
425
426   /* Libera gli array per sigma */
427   d_libera_3d (griglia_g1.sig, griglia_g1.dimx, griglia_g1.
    dimy);
428   Libera_griglia (&griglia_g1);
429
430   if ((!rank) && (sim_completa))
431     {
432       puts ("-----  SIMULAZIONE DELLA PARTE FDTD TERMINATA
    -----\n");
433     }
434   return (0);
```

```
435 }
436
437
438 double
439 stampa_new (struct_griglia * griglia, struct_eccitazione * ecc
      , double dt)
440 {
441   double tprev, comodo;
442   double tprev_tot = 0.0;
443
444   puts ("DATI SIMULAZIONE");
445   printf ("%s \n", ecc->nome);
446   switch (ecc->tipo)
447     {
448     case 1:
449       printf ("tg: %13.4g s\n", ecc->tg);
450       printf ("t0: %13.4g s\n", ecc->t0);
451       printf ("Frequenza minima: %13.4g GHz\n", ecc->fmin *
      1.0E-9);
452       printf ("Frequenza massima: %13.4g GHz\n", ecc->fmax *
      1.0E-9);
453       printf ("Frequenza sinusoide (modulante l'impulso):
      %13.4g GHz\n", ecc->freq * 1.0E-9);
454       printf ("Rapporto medio Periodo/dt: %13.4g \n", ecc->
      passi_per);
455       break;
456     case 2:
457       printf ("Frequenza sinusoide: %13.4g GHz\n", ecc->freq *
      1.0E-9);
458       printf ("Rapporto medio Periodo/dt: %13.4g \n", ecc->
      passi_per);
459       break;
460     default:
461       printf ("Tipo di eccitazione non presente\n");
462       break;
463     }
464   puts
      ("-------------------------------------------------------------")
      ;
465   puts ("Dimensioni dell'area di Calcolo");
466   /*  printf("Sottogriglia: %s\n",griglia->srdom1); */
467   printf ("Passo Temporale: %8.4g ps     Totale Iterazioni:
      %8.0f \n", dt * 1.0E12, ecc->tfinale);
468   printf ("Totale Periodi: %ld\n", ecc->Nperiod);
469   comodo = griglia->dimx * griglia->dimy * griglia->dimz;
470   tprev = comodo;
471   printf ("Dimensioni della griglia (Nx*Ny*Nz): %ld * %ld * %
      ld\n", griglia->dimx, griglia->dimy, griglia->dimz);
472   printf ("Numero totale di celle: %g\n", comodo);
473   comodo = griglia->lx * 1e3;
474   printf ("Lunghezza lungo X [mm]: %8.4g\n", comodo);
475   comodo = griglia->ly * 1e3;
```

```
476   printf ("Lunghezza lungo Y [mm]: %8.4g\n", comodo);
477   comodo = griglia->lz * 1e3;
478   printf ("Lunghezza lungo Z [mm]: %8.4g\n", comodo);
479   printf ("Fattore di stabilita' Max - Min : %g  -  %g \n",
       griglia->maxst, griglia->minst);
480   printf ("Rapporto Max - Min lambda/dz: %8.4g  -  %8.4g\n",
       griglia->maxldz, griglia->minldz);
481   printf ("Rapporto Max - Min lambda/dy: %8.4g  -  %8.4g\n",
       griglia->maxldy, griglia->minldy);
482   printf ("Rapporto Max - Min lambda/dx: %8.4g  -  %8.4g\n",
       griglia->maxldx, griglia->minldx);
483   puts
       ("-----------------------------------------------------------------")
       ;
484   comodo = (tprev * ecc->tfinale / 4900000) / 60;
485   printf ("Tempo di calcolo stimato per questo dominio (P4-2
       GHz): %f min.\n", comodo);
486   tprev_tot += comodo;
487   /*printf ("Tempo di calcolo stimato globale  (P4-2GHz): %f
       min.\n", tprev_tot); */
488   puts
       ("-----------------------------------------------------------------")
       ;
489   return (tprev_tot);
490 }
491
492 int
493 stampa_max (int flag_x, int flag_y, int flag_z, int flag_t,
       struct_campo_max * campo_max, struct_griglia * griglia,
494       struct_nomi * nomem)
495 {
496 /* ordine per scrivere il file 3D: */
497 /* dimensione k */
498 /* dimensione j */
499 /* dimensione i */
500 /* cicla prima k, poi j, poi i: valore(i,j,k) */
501   char stringa[255];
502   register int i, j, k;
503   double mvx, mvy, mvz, da;
504   /*long int xin = 30, yin = 30, zin = 30, xfi = griglia->dimx
       - 30, yfi = griglia->dimy - 30, zfi = griglia->dimz -
       30;*/
505   long int xin=0, yin=0, zin=0, xfi=griglia->dimx, yfi=griglia
       ->dimy, zfi=griglia->dimz;
506   FILE *FX, *FY, *FZ, *FT;
507   double sar_loc = 0.0;
508
509   if (flag_x == 1)
510     {
511       if ((FX = fopen (nomem->exmax, "w")) == NULL)
512   return (-1);
513       /* FX = fopen ((*nomi_file).exmax, "w"); */
```

```
514       sprintf (stringa, "%ld\n", zfi - zin);
515       fputs (stringa, FX);
516       sprintf (stringa, "%ld\n", yfi - yin);
517       fputs (stringa, FX);
518       sprintf (stringa, "%ld\n", xfi - xin - 1);
519       fputs (stringa, FX);
520       for (k = zin; k < zfi; k++)
521   for (j = yin; j < yfi; j++)
522     for (i = xin; i < xfi - 1; i++)
523       {
524         sprintf (stringa, "%E\n", ((*campo_max).vsx[i][j][k] -
      (*campo_max).vix[i][j][k]) * (double) 0.5);
525         fputs (stringa, FX);
526       }
527       fclose (FX);
528     }
529
530   if (flag_y == 1)
531     {
532       if ((FY = fopen (nomem->eymax, "w")) == NULL)
533   return (-1);
534       /* FY = fopen ((*nomi_file).eymax, "w"); */
535       sprintf (stringa, "%ld\n", zfi - zin);
536       fputs (stringa, FY);
537       sprintf (stringa, "%ld\n", yfi - yin - 1);
538       fputs (stringa, FY);
539       sprintf (stringa, "%ld\n", xfi - xin);
540       fputs (stringa, FY);
541       for (k = zin; k < zfi; k++)
542   for (j = yin; j < yfi - 1; j++)
543     for (i = xin; i < xfi; i++)
544       {
545         sprintf (stringa, "%E\n", ((*campo_max).vsy[i][j][k] -
      (*campo_max).viy[i][j][k]) * (double) 0.5);
546         fputs (stringa, FY);
547       }
548       fclose (FY);
549     }
550
551   if (flag_z == 1)
552     {
553       if ((FZ = fopen (nomem->ezmax, "w")) == NULL)
554   return (-1);
555       /* FZ = fopen ((*nomi_file).ezmax, "w"); */
556       sprintf (stringa, "%ld\n", zfi - zin - 1);
557       fputs (stringa, FZ);
558       sprintf (stringa, "%ld\n", yfi - yin);
559       fputs (stringa, FZ);
560       sprintf (stringa, "%ld\n", xfi - xin);
561       fputs (stringa, FZ);
562       for (k = zin; k < zfi - 1; k++)
563   for (j = yin; j < yfi; j++)
```

```
564     for (i = xin; i < xfi; i++)
565       {
566         sprintf (stringa, "%E\n", ((*campo_max).vsz[i][j][k] -
      (*campo_max).viz[i][j][k]) * (double) 0.5);
567         fputs (stringa, FZ);
568       }
569       fclose (FZ);
570     }
571
572   if (flag_t == 1)
573     {
574       if ((FT = fopen (nomem->emax, "w")) == NULL)
575   return (-1);
576       /* FT = fopen ((*nomi_file).emax, "w"); */
577       sprintf (stringa, "%ld\n", zfi - zin - 1);
578       fputs (stringa, FT);
579       sprintf (stringa, "%ld\n", yfi - yin - 1);
580       fputs (stringa, FT);
581       sprintf (stringa, "%ld\n", xfi - xin - 1);
582       fputs (stringa, FT);
583       for (k = zin; k < zfi - 1; k++)
584   for (j = yin; j < yfi - 1; j++)
585     for (i = xin; i < xfi - 1; i++)
586       {
587         mvx = (*campo_max).vsx[i][j][k] + (*campo_max).vsx[i][
      j][k + 1] +
588       (*campo_max).vsx[i][j + 1][k] + (*campo_max).vsx[i][j +
      1][k + 1] -
589       (*campo_max).vix[i][j][k] - (*campo_max).vix[i][j][k + 1]
      -
590       (*campo_max).vix[i][j + 1][k] - (*campo_max).vix[i][j +
      1][k + 1];
591         mvy = (*campo_max).vsy[i][j][k] + (*campo_max).vsy[i][
      j][k + 1] +
592       (*campo_max).vsy[i + 1][j][k] + (*campo_max).vsy[i + 1][j
      ][k + 1] -
593       (*campo_max).viy[i][j][k] - (*campo_max).viy[i][j][k + 1]
      -
594       (*campo_max).viy[i + 1][j][k] - (*campo_max).viy[i + 1][j
      ][k + 1];
595         mvz = (*campo_max).vsz[i][j][k] + (*campo_max).vsz[i][
      j + 1][k] +
596       (*campo_max).vsz[i + 1][j][k] + (*campo_max).vsz[i + 1][j
      + 1][k] -
597       (*campo_max).viz[i][j][k] - (*campo_max).viz[i][j + 1][k]
      -
598       (*campo_max).viz[i + 1][j][k] - (*campo_max).viz[i + 1][j
      + 1][k];
599         da = sqrt (mvx * mvx + mvy * mvy + mvz * mvz) * (
      double) 0.125;
600         sar_loc += 0.5 * da * da * griglia->sig[i][j][k] *
      griglia->dx[i] * griglia->dy[j] * griglia->dz[k];
```

```
601         sprintf (stringa, "%E\n", da);
602         fputs (stringa, FT);
603       }
604       /*printf ("Potenza assorbita nel dominio: %g [watt]\n",
    sar_loc); */
605       /*sar_tot+=sar_loc; */
606       fclose (FT);
607     }
608   return (0);
609 }
```

2.3 CONFIGURATION SETUP

In the Fig. 2.1 shows the initial setup of the simulation. It had two antennas, one emitting and one receiving, separated by a metal barrier, and finally a metasurface. Using this type of configuration, problems of wave dispersion have been encountered due to the metal barrier since if the generated wave fell on one of the edges of the PEC, problems with the direction of the wave would arise.

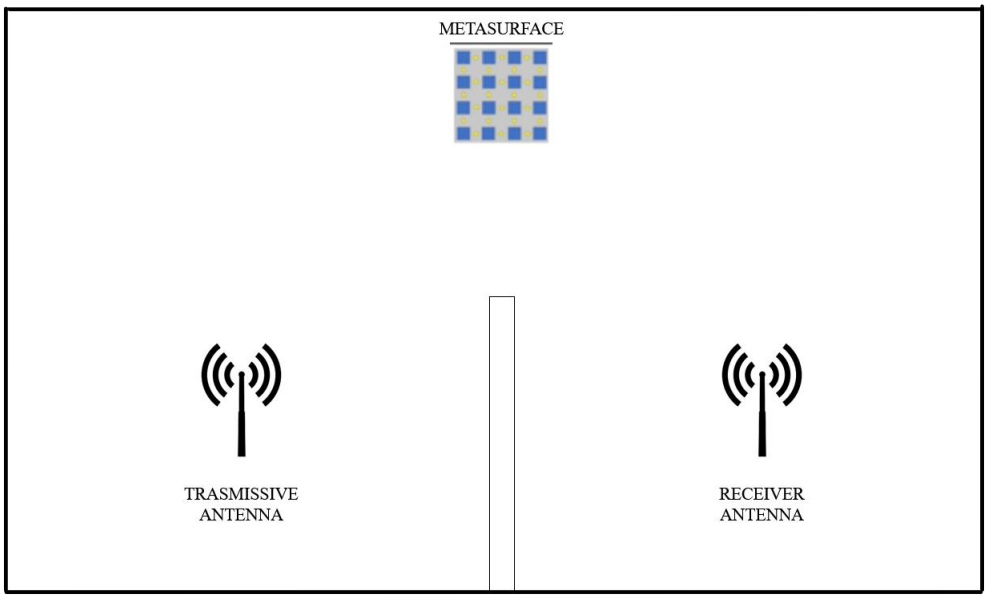

Figura 2.1: First setup

- Metasurface placement;
- Trasmissive antenna placement;
- Receiver antenna placement;
- PEC metal barrier placement;

- PEC METAL BARRIER:
  1 x 60 x 60
  A perfect conductor, also known as a perfect electric conductor (PEC), is an idealized substance with infinite electrical conductivity or, in other words, zero resistance (cf.perfect dielectric). While there are no ideal electrical conductors in nature, the concept is a helpful model when electrical resistance is small in comparison to other effects. Ideal magnetohydrodynamics, the study of completely conducting fluids, is one example. Another example is electrical circuit schematics, which imply that the lines connecting the components are free of resistance. Another example is in computational electromagnetics, where PEC may be simulated quicker since the sections of equations that account for finite conductivity can be ignored.

- ANTENNA CHARACTERISTICS:
  The simulation study of the provided antenna is performed using the FDTD approach, which is implemented in the C computer language. A dipole antenna is a radio frequency energy transmission or receiving antenna with a center-fed driving element. From a physics standpoint, this is the most basic feasible antenna. It is constructed composed of a straight electric conductor made of a conducting metal such as copper that is interrupted at the center, resulting in two poles.
  Frequency band: 5G n77 (3300 - 4200) GHz 3700 GHz for the optimization. For the analysis (2500 - 5000) GHz

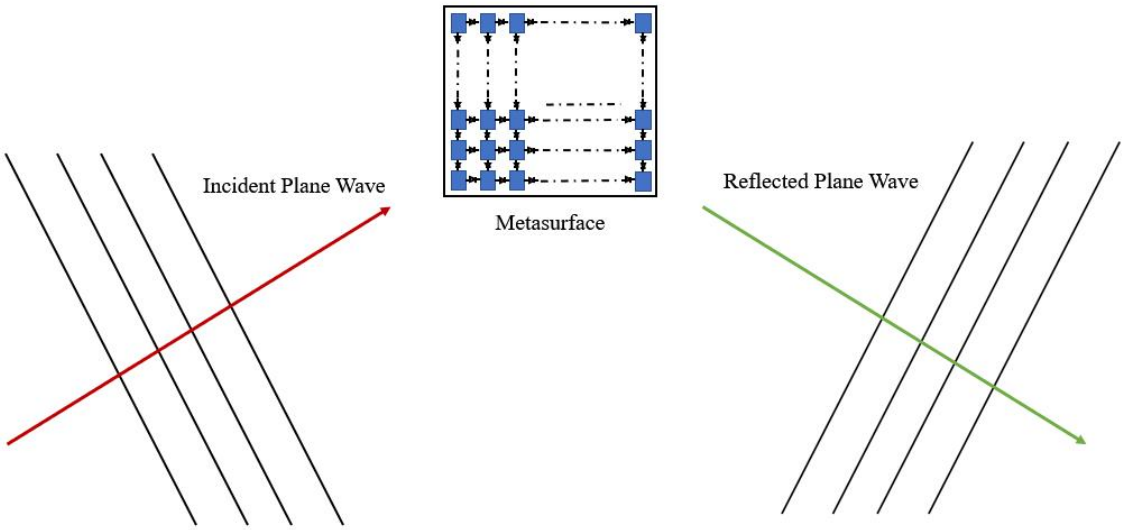

Figura 2.2: Final setup

Due to the problems listed above it was decided to adopt a second setup with which to carry out the experimentation. In this one the antennas and the PEC were removed and it was decided to use a plane wave that reflected on the metasurface. Figure 2.2 shows the final setup used for the experimentation. In this configuration there is an incident plane wave (red arrow), which is reflected (green arrow) hitting the metasurface. The metasurface used within it is composed of the following:

METASURFACE

- 10 x 10 patches
- 1 mm distance between patches
- 56 mm x 1 mm x 56 mm surface

The use of a bigger metasurface allows to reproduce more reliable results.

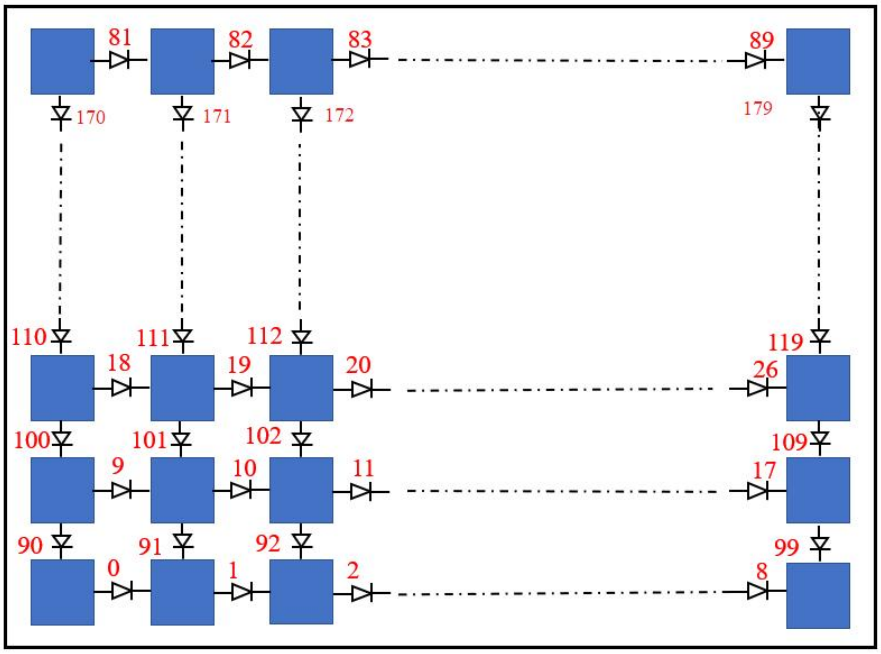

Figura 2.3: Diode numbering

The patches that make up the metasurface are connected to each other by a series of diodes (both horizontally and vertically). In total there are 180 diodes and they have been numbered from 0 to 179, as shown in Fig. 2.3.

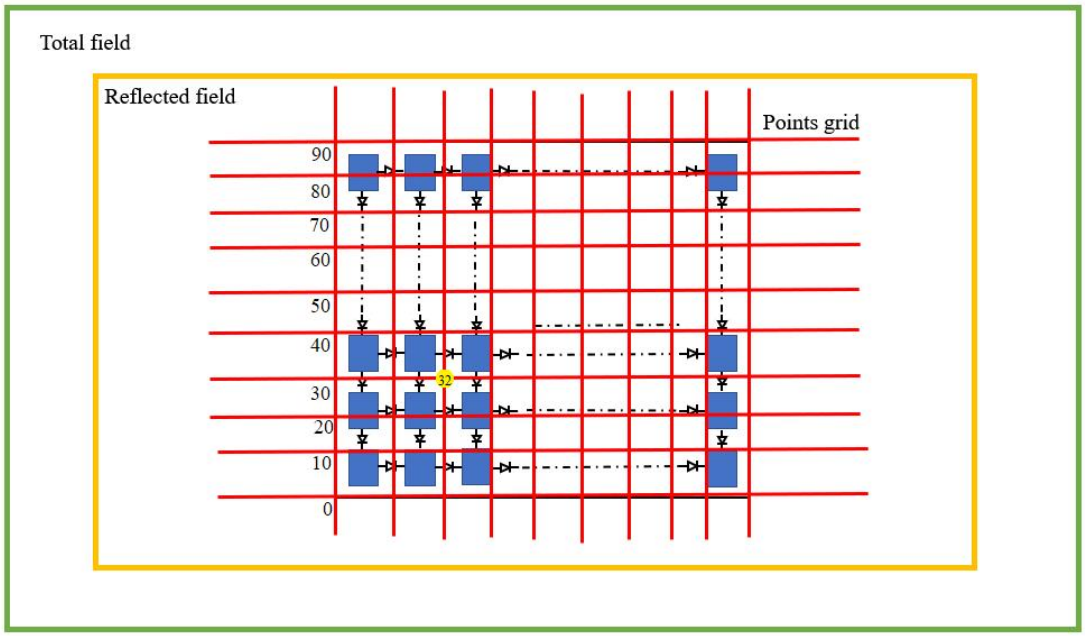

Figura 2.4: Complete setup

Finally, the final setup used for the experimentation is shown in Fig. 2.4.: From the outside towards the inside we find the total field (green box), the reflected field (orange box), the 10x10 grid where the starting points for carrying out the experimentation are placed (red grid) and finally the metasurface. The point of the red grid are numered from 0 to 99, the point used for this study is the point number 32 (yellow point).

Capitolo 3

RESULTS

Starting from the configuration shown in figure 2.4 we started the simulation. The initial capacitances of the diodes have been set at the outset with a value of 1 pF for some simulations and 0.1 pF for others. Following the script reported in the code description chapter, the diodes were reorganized in groups of 10 to change the capacities, since the code did not allow to change the capacities individually. As can be seen from the code, the extraction of the diodes was performed randomly. The ten groups of diodes have been numbered from 0 to 9 and their belonging to the relative group is shown in the following table.

	GROUP									
	0	1	2	3	4	5	6	7	8	9
	12	50	3	1	0	10	5	19	7	2
	13	57	4	11	18	15	21	41	27	9
D	39	65	6	14	20	16	38	46	29	52
I	42	73	8	25	23	17	58	47	40	54
O	66	90	22	28	32	26	68	48	43	63
D	69	93	24	35	34	30	74	49	51	83
E	72	113	33	45	36	31	76	59	53	88
	77	120	44	67	37	86	84	62	61	94
N	82	133	55	70	56	91	110	75	80	98
U	92	144	60	78	64	102	114	85	96	123
M	103	147	79	112	71	106	125	87	107	139
B	104	177	81	121	109	115	129	89	117	140
E	116		97	122	135	118	132	95	119	158
R	126		101	142	143	128	136	99	164	165
	131		108	149	145	130	157	100	168	170
	169		111	151	156	137	167	105	174	
	172		127	173	163	141		124	178	
			152	175	166	153		134		
			160	179		155		138		
			176			159		146		
						161		148		
						171		150		
								154		
								162		

Tabella 3.1: The table shows how the various diodes have been divided into the 10 groups

DISTANCE [m]	α	θ	ϕ
0.2	$\pi/2$	$\pi/4$	$\pi/4$

Tabella 3.2: The table shows the values of the first simulation with a single plane wave

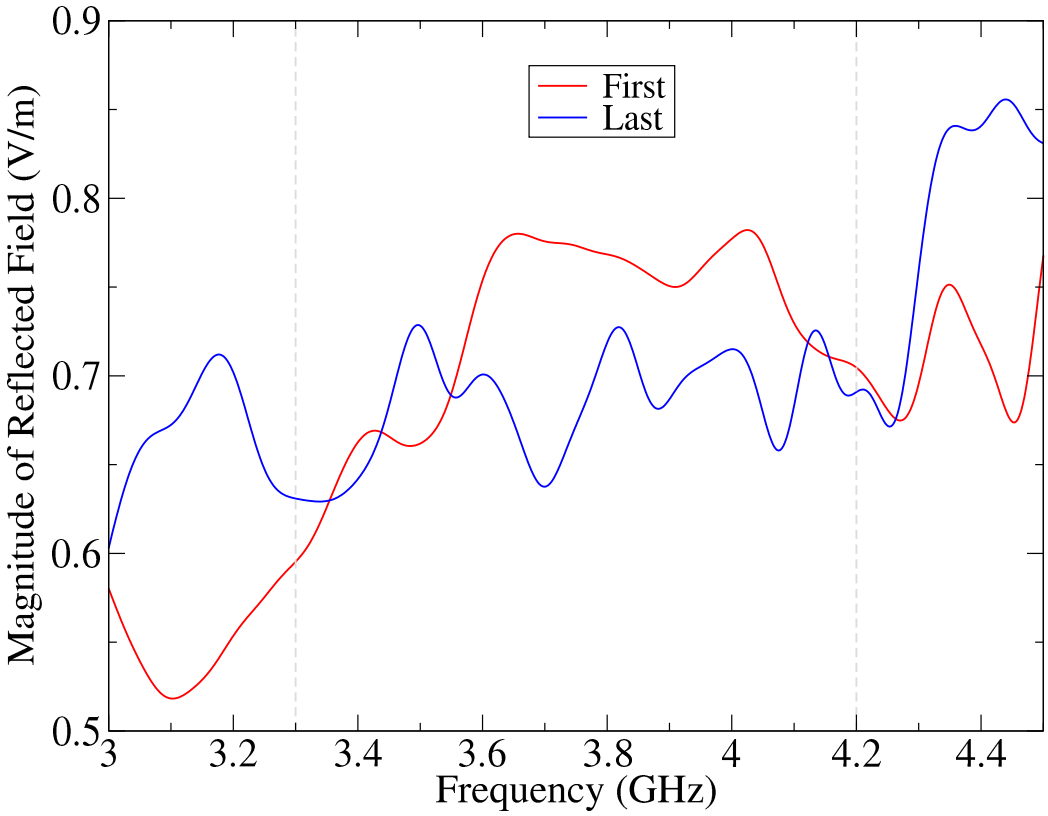

Figura 3.1: Results of Minimization.

The first simulation was performed with a single plane wave having the following values showed in Tab. 3.2. With the data of the single plane wave two simulations were performed, the first of minimization of the reflected field in the desired direction and the second of maximization. Fig. 3.1 and Fig. 3.2 show the results obtained. Figure 3.1 shows the result of minimizing the reflected field in grid point number 32. The analysis frequency of interest is the one between 3.3 GHz and 4.2 GHz. The graph shows two trends: the one in red represents the result before minimization is performed, the one in blue which represents the post-minimization trend. We can note that, in the frequency band of interest, an evident decrease in the magnitude of the reflected field is presented.

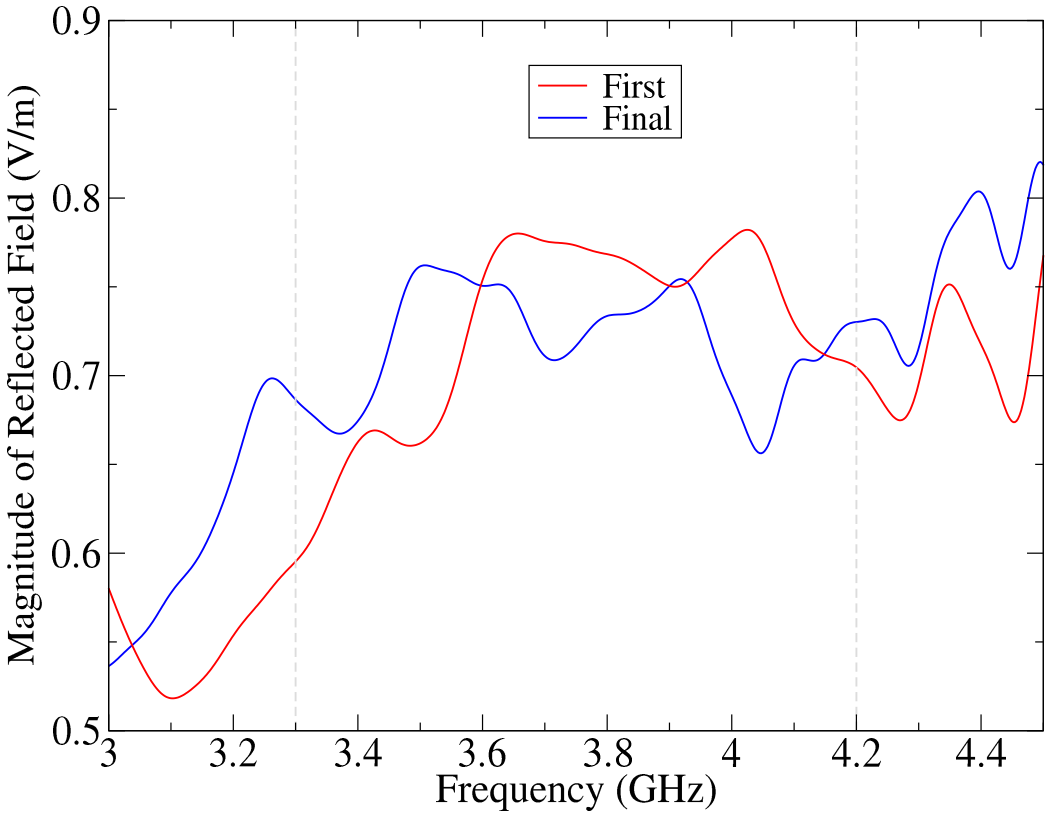

Figura 3.2: Results of Maximization.

Figure 3.2 shows the result of minimization of the reflected field in grid point number 32. The analysis frequency of interest is the one between 3.3 GHz and 4.2 GHz. The graph shows two trends: the one in red represents the result before minimization is performed, the one in blue which represents the post-minimization trend. We can note that, in the frequency band of interest, an evident increase in the magnitude of the reflected field is presented.
Since the grid of points is a 10x10, it is composed of 100 points numbered from 0 to 99. The results shown in Fig 3.1 and Fig. 3.2 represent only those of the point we have chosen, which is point number 32.

DISTANCE [m]	α °	θ °	ϕ °
0.29	347.79	61.30	84.80
0.16	168.73	103.13	43.54
0.20	277.41	101.98	63.60
0.16	134.65	175.90	157.56
0.29	110.58	18.90	144.96
0.22	162.15	60.73	121.47
0.40	268.14	79.64	73.33
0.28	3.78	67.03	40.10
0.35	55.58	55.58	166.16
0.17	89.95	37.81	37.82

Tabella 3.3: The table shows the values relating to the 10 plane waves generated.

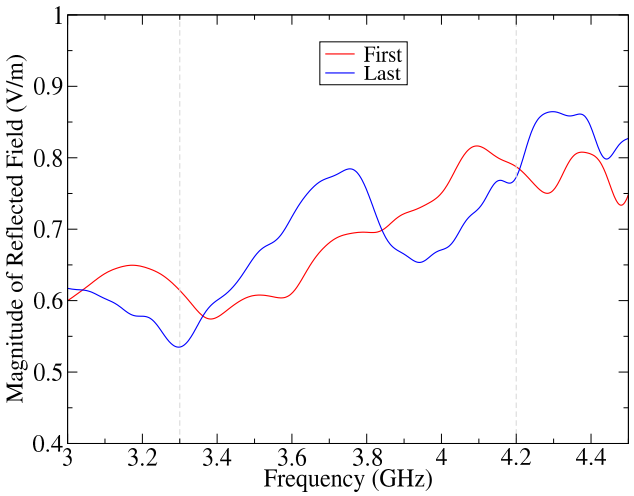

Figura 3.3: Results of Minimization of 10 plane waves.

The second simulation was performed with a beam of plane waves composed of 10 waves. The waves were generated randomly, the values of the single waves are shown in the Tab.3.3. Fig. 3.3 shows the result of maximization of the reflected field in grid point number 32. The analysis frequency of interest is the one between 3.3 GHz and 4.2 GHz. The graph shows two trends: the one in red represents the result before manimization is performed, the one in blue which represents the post-manimization trend. As regards the simulation with 10 plane waves in the graph, we can see that the program has managed to minimize the reflected field in the frequencies between 3.4 and 3.8 GHz, while in the second part of the band of interest there have been increases in the intensity of the reflected field. Regarding the maximization of the simulation with 10 plane waves, two tests were performed. The first in the same previous band which did not bring results, instead going to decrease the band of interest, bringing it between 3.6 GHz and 3.8 GHz, as shown in Fig. it can be seen that the simulation managed to converge, however, after having

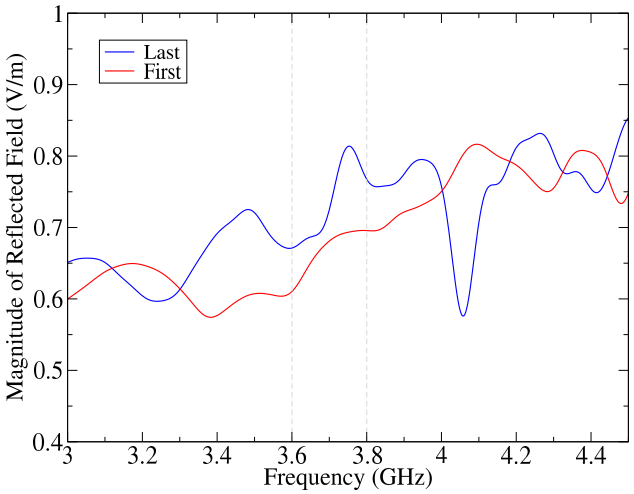

Figura 3.4: Results of Maximization of 10 plane waves.

performed a considerable number of simulations.

Capitolo 4

DISCUSSION

We can see and infer from the previous results that we were able to develop and simulate a manually reconfigurable metasurface using the FDTD simulation method implemented in the C language and evaluate the electric field distributions along it after the emission of a plane wave. We were also able to evaluate the reflection of this wave at any point in the FDTD space lattice. It could be demonstrated that different diode configurations result in various reflections of the plane wave passing through the metasurface, opening up the possibility of giving the varactor diodes a specific configuration in terms of capacitance in order to achieve a desired focus of the reflected signal. The simulation was therefore divided into two parts: an analysis part and a synthesis part. As regards the analysis part, as shown in the code description paragraph, a code capable of developing an intelligent surface FDTD model that can be manually reconfigured has been implemented in C language, evaluating its operating principles, its response to a plane wave accident by evaluating the distribution of the electric field on it. In this part all the values of the diodes with capacitance of 1pF or 0.1pF have been set. The synthesis part was in turn divided into several steps. As a first step, the diodes were divided into 10 groups in random order, as shown by the algorithm present in the code description part (values shown in Tab. 3.1). After that, the starting point on the grid was chosen. Once set on which point and from which point to start, the first test was to generate a plane wave and then go to graph the minimization and maximization of the module the magnitude of the reflected field. And from the results obtained, it can be seen that the synthesis has brought significant results. The second test was to generate 10 random plane waves and then go to graph the minimization and maximization. Here as regards the minimization there have been improvements, but the same was not possible as regards the maximization as the graph did not show any improvements.

Elenco delle figure

Bibliografia

[1] https://www.gnu.org/software/gsl/doc/html/multimin.html.

[2] L. P. Mancera P. Bowen D. R. Smith, O. Yurduseven and N. B. Kundtz. "analysis of a waveguide-fed metasurface antenna,". *Physical Review Applied*, 08(5):054048, 2017.

[3] Julien de Rosny Merouane Debbah Mohamed-Slim Alouini Rui Zhang Ertugrul Basar, Marco Di Renzo. Wireless communications through reconfigurable intelligent surfaces. *IEEE*, page 20 pages, July 2019, 7.

[4] MARCO DI RENZO2 (Senior Member-IEEE) JULIEN DE ROSNY3 MEROUANE DEBBAH4 5 (Fellow IEEE) MOHAMED-SLIM ALOUINI 6 (Fellow IEEE) ERTUGRUL BASAR 1 (Senior Member, IEEE) and IEEE) RUI ZHANG 7, (Fellow. Wireless communications through reconfigurable intelligent surfaces. *IEEE Communications Surveys Tutorials*.

[5] Allen Taflove Susan C. Hagness. Computationai electrodynamics the finite-difference time-domain method third edition.

[6] Q. Cheng J. Y. Dai, J. Zhao and T. J. Cui. Independent control of harmonic amplitudes and phases via a time-domain digital coding metasurface,". *Light: Science Applications*, 7(1):90, 2018.

[7] K. S. Kunz and R. J. Luebbers. "the finite difference time domain method for electromagnetics". 2018.

[8] Yuanwei Liu, Xiao Liu, Xidong Mu, Tianwei Hou, Jiaqi Xu, Marco Di Renzo, and Naofal Al-Dhahir. Reconfigurable intelligent surfaces: Principles and opportunities. *IEEE Communications Surveys Tutorials*, 23(3):1546–1577, 2021.

[9] CNR Nanote Marco Esposito Dipartimento di Matematica Fisica "Ennio De Giorgi" Università del Salento, CNR Nanotec Francesco Todisco Dipartimento di Matematica Fisica "Ennio De Giorgi" Università del Salento. I metamateriali: Invisibilità e rifrazione negativa in mezzi nanochirali.

[10] F. Moglie and A.P. Pastore. Fdtd analysis of plane wave superposition to simulate susceptibility tests in reverberation chambers. *IEEE Transactions on Electromagnetic Compatibility*, 48(1):195–202, 2006.

[11] J. Reddy. Solutions manual for an introduction to the finite element method. page 41, 2006.

[12] J. B. Schneider. Understanding the finite-difference time-domain method. 2021.

[13] D. M. Sullivan. Z-transform theory and the fdtd method. *IEEE Transactions on Antennas and Propagation*, 44(1):28–34, Jan. 1996.

[14] A. Taflove. Application of the finite-difference time-domain method to sinusoidal steady-state electromagnetic-penetration problems. EMC-22(3):191–202, 1980.

[15] A. Taflove and S. C. Hagness. The finite-difference time-domain method third edition.

[16] S. N. Tcvetkova A. D´ıaz-Rubio Y. Ra'di V. S. Asadchy, M. Albooyeh and S. Tretyakov. "perfect control of reflection and refraction using spatially dispersive metasurfaces. *Physical Review Applied*, 94(7):075142, 2016.

[17] R. Ro V. V. Varadan. Analyticity, causality, energy conservation and the sign of the imaginary part of the permittivity and permeability. *IEEE*, pages 09–14, July 09-14 2006.

[18] K. S. Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," ieee trans. antennas propag. 14(3):302–307, 1966.