NTNU – Trondheim
Norwegian University of
Science and Technology

Numerical Methods for Electromagnetic Field Propagation over an Undulating Surface in the Frequency Region of 110 MHz

Kristin Åstebøl

Electronics Engineering
Submission date: June 2013
Supervisor: Torbjørn Ekman, IET
Co-supervisor: Thor Breien, Indra Navia AS

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Instrument Landing System (ILS) is a VHF/UHF radio based approach and landing system guiding aircraft under bad visibility conditions. ILS is sensitive to multipath caused by airport buildings, taxiing aircrafts and the surrounding terrain. Several computer based tools to predict their influence on the ILS signals exist. These tools are based on the electromagnetic principles Physical Optics and Geometrical Theory of Diffraction. The tasks for the master thesis are:

- Find numerical methods that work for electromagnetic field simulation over a humped runway, at the frequency of the ILS localizer, 110 MHz.
- Implement the investigated numerical methods.
- Evaluate the implemented methods.
- Evaluate the need for terrain modeling at the frequency of the ILS localizer, 110 MHz.

The thesis problem is given by Indra Navia, a world-leading ILS manufacturer.

Acknowledgements

First and foremost I would like to thank my supervisor Torbjörn Ekman at NTNU, for support, ideas, and advices during the work of this thesis. Your enthusiasm and our discussions have been invaluable.

I would also like to thank Indra Navia for this very interesting thesis problem, and for motivating, educational and inspiring visits to the office in Oslo. I would especially like to thank Thor Breien for his good advices and invaluable knowledge. In addition, I would like to thank Alf Bakken and Morten Topland for giving me assisting material and feedback.

I would also like to thank my family and friends for supporting me during the work of this thesis. Finally, I would like to thank my fellow students for cheerful days at the university, and for two fantastic years at NTNU.

12.06.2013, Trondheim
Kristin Åstebøl

Abstract

When an aircraft is landing, the use of the Instrument Landing System (ILS) is essential. It provides navigation signals for the landing aircraft. The localizer transmits the signals for horizontal navigation, and is situated at the opposite end of the runway of where the aircrafts are landing. It means that the localizer-signals have to traverse the runway, before the aircrafts receive the signals. The signals may behave differently if the runway is humped. In some cases there might not be line-of-sight between the two ends of the runway. The objective of this thesis was to find numerical methods for computation of the electromagnetic field in the frequency region of the localizer over a humped runway, implement them, and test them. The investigated numerical methods are the Integral Equation Model and the Parabolic Equation Method. The principle of the Integral Equation Model is to compute the field strength at a given point based on direct wave from the transmitter and the induced surface current. The method is not fully implemented due to missing links in the literature used. The principle of Parabolic Equation Method is to solve the standard parabolic equation, a differential equation derived from the scalar wave equation. The Parabolic Equation Method is implemented with two different algorithms; the Split-Step Algorithm (SSA) and the Finite-Difference Method (FDM). Over a flat surface the SSA and FDM results differ somehow. However, as soon as there are some irregularities, up- or downwards inclined plane, wedge, or runway surface profiles, the SSA and FDM give almost identical results. The simulations with SSA and FDM also show that the runway surface profile can influence the electromagnetic field considerably. Therefore, the runway surface profile needs to be taken into account. How suitable the Integral Equation Model is for this application remains subject to further work. However, the Parabolic Equation Method is a numerical method that can be used to simulate electromagnetic field propagation in the frequency region of the localizer over a humped runway.

Sammendrag

Når et fly skal lande bruker det navigasjonssignalene fra instrumentlandingsystemet (ILS) på flyplassen. Localizeren sender navigasjonssignalene for horisontal navigasjon for det landende flyet, og er plassert på motsatt side av rullebanen i forhold til der flyet lander. Det betyr at localizer-signalene må krysse rullebanen før flyet mottar dem. Hvis rullebanen ikke er flat, vil signalene oppføre seg annerledes enn for en flat rullebane. På noen rullebaner er det ikke frisiktlinje fra den ene til den andre enden. Målet med denne oppgaven er å finne numeriske metoder som kan brukes for å beregne det elektromagnetiske feltet i frekvensområdet til localizeren over en ikke-flat rullebane, og implementere og teste dem. De to metodene som ble undersøkt er integralligningsmetoden og metoden med parabolsk ligning. Prinsippet for integralligningsmetoden er å beregne feltstyrken i et gitt punkt basert på den direkte bølgen fra senderen og den induserte overflatestrømmen. I litteraturen som ble brukt manglet det noen vesentlige detaljer, og integralligningsmetoden kunne derfor ikke bli ferdigimplementert. Prinsippet for metoden med parabolsk ligning er å løse "the standard parabolic equation", en ligning utledet fra den skalare bølgeligningen. Metoden er implemtentert på to forskjellige måter, med "Split-Step Algorithm" (SSA) og "Finite-Difference Method" (FDM). Over en plan flate gir SSA og FDM litt forskjellige resultater. For flater med irregulariter derimot, som skråplan, kile og ikke-flate rullebaner, gir SSA og FDM så og si like resultater. Simuleringene med SSA og FDM viser at ikke-flate rullebaner kan påvirke det elektromagntiske feltet vesentlig. Derfor bør rullebaneprofilen tas hensyn til ved elektromagnetiske beregninger. Hvor bra integralligningsmetoden fungerer forblir en oppgave til videre arbeid. Metoden med parabolsk ligning er derimot egnet for elektromagnetiske feltberegninger i frekvensområdet til localizeren.

Contents

List of Figures

Nomenclature

Roman Symbols

$\boldsymbol{E}$	Electric field
$\boldsymbol{H}$	Magnetic field
$\boldsymbol{r'}$	Vector from origin to the source point
$\boldsymbol{r}$	Vector from origin to the observation point
$\boldsymbol{E}^s(\boldsymbol{r})$	Scattered electric field
f	Frequency, $[Hz]$
k	Wavenumber, $k = \frac{2\pi}{\lambda}$
n	Refractive index
t	Time [s]
x	Direction of propagation
z	Vertical direction

Greek Symbols

α	The angle measured from the paraxial direction, [rad]
β	Half-power beamwidth, [rad]
ϵ	Permittivity of the medium
γ	A constant
λ	Wavelength, $[m]$
λ_{max}	The maximum eigenvalue of the matrix system in question
μ	Permeability of the medium

ω Angular frequency, $\omega = 2\pi f$

θ The angle, measured from the paraxial direction [rad]

θ_0 The tilt of the beam [rad]

Superscripts

2D Two dimensions

3D Three dimensions

m meter

Other Symbols

Bold Symbols in bold are vectors

Δx Step-size in the x-direction, $[m]$

Δz Step-size in the z-direction, $[m]$

Acronyms

CST Computer Simulation Technology

FDM Finite-Difference Method

PE Parabolic Equation Method

PML Perfectly Matched Layer

SSA Split-Step Algorithm

UTD Uniform Theory of Diffraction

Chapter 1

Introduction

When an aircraft is landing, the use of the Instrument Landing System (ILS) is essential. The purpose of the ILS is to guide the landing aircraft towards a safe landing, especially under bad weather conditions. This is done by guiding the aircraft towards and along a desired path to the runway. The guiding consists of two signals; one giving the relative position with respect to the desired path in the horizontal direction, and the other the relative position with respect to the desired path in the vertical direction. The horizontal signals are transmitted by the localizer, in the frequency region of 108 to 112 MHz, [Holm, 2002, Chapter 2.1], 110 MHz is used in this thesis. The vertical signals are transmitted by the glide path, in the frequency region of 329-335 MHz, [Holm, 2002, Chapter 3.3]. In order to navigate on the signals towards and along the desired path, 90 and 150 Hz signals are amplitude modulated into the signals, for both the localizer and the glide path. The 90 and 150 Hz components are transmitted each on its side of the desired path, for both the horizontal and the vertical signals. When the landing aircraft receives the same amount of the 90 and 150 Hz component for both the horizontal and vertical signals, the aircraft is at the desired path. For illustration of the principle in the case of the localizer, see figure 1.1.

In order to know the distance left to the runway, three ”base stations” called marker beacons, are placed at known distances ahead of the runway. They transmit signals up in the air, in order to ”notify” the aircrafts of the distance left to the runway. For illustration of the ILS principle with glide path signals the marker beacons, see figure 1.2. For more theory regarding ILS, see Åstebøl [2012].

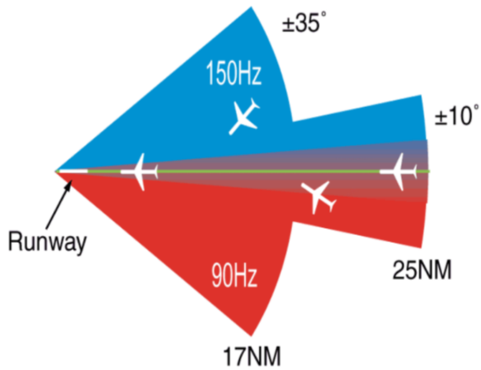


Figure 1.1: Illustration of the principle for navigation using the signals from the localizer, [Holm, 2002, p. 2-4].

The localizer is situated at the opposite end of the runway of where the aircrafts

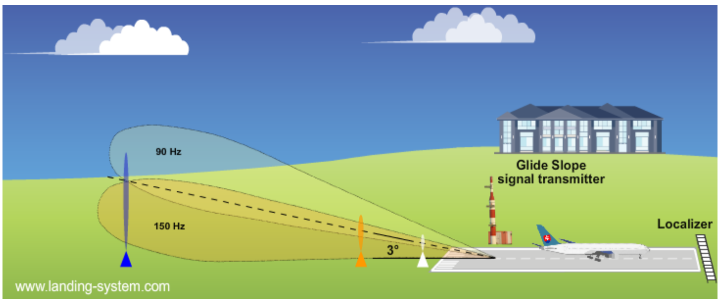


Figure 1.2: Illustration of the ILS with glide path signals and marker beam. The dotted line is the desired path towards the runway, [Landing-Systems].

are landing. It means that the localizer-signals have to traverse the runway, before the aircrafts receive the signals. The field strength over a runway depends on the transmitter power, the antenna gain, the antenna height, and the runway surface profile. This thesis regards the influence of a humped runway surface on electromagnetic waves at the frequency of the ILS localizer, 110 MHz, and numerical methods that can simulate this. It means that the numerical methods should be able to handle slowly varying terrain. It also means that there will not be any back scattering of the signals along the runway, because the terrain is slowly varying.

For a flat runway, there exist very good methods and softwares for computing the propagation of the electromagnetic field along the runway. This also includes the multi-path effects that can occur at an airport. However, the case of a humped runway is not extensively investigated. In the telecom and radio business they simulate fields over irregular terrain for estimating coverage at similar frequencies, for GSM and FM. The distances over which they simulate fields are much larger than for this case, and the methods used are therefore too approximative.

The tasks of the thesis are therefore to find numerical methods for simulation of electromagnetic field propagation at the frequency of the localizer, that can handle a humped runway. In order to find the performance of the numerical methods, they need to be implemented, and then tested. This may give an indication of how humped runways can affect signals at the frequency of the localizer, and whether or not it is necessary to take the terrain profile into account.

This thesis firstly presents analytical models for field strength calculation over a flat surface and source modeling. The analytical models are of interest because the field strength over a flat surface will be a reference for the field propagating over an undulating surface, humped runways. Source modeling is of interest because it shows what the propagated beam in the simulations will look like. Secondly, the principles of the two proposed numerical methods, the Integral Equation Model and the Parabolic Equation Method, are explained. Thirdly, the parameters for simulation of the methods are chosen. They have to be chosen correctly in order to obtain good results. Fourthly, the performance of the implemented methods are tested. This leads to a discussion and a conclusion.

In this thesis, the terms "humped runway", "irregular terrain", and "undulating terrain", all refer to "smoothly varying terrain, relative to the wavelength".

Chapter 2

Analytical Models and Source Modeling

In order to predict the behavior of signals at the frequency of the ILS localizer over a runway, the field propagation needs to be simulated. Depending on the shape of the runway, either an analytical model or a numerical method can be used. For surfaces with a simple shape like a flat surface, flat surface with a knife-edge, or a surface that can be approximated to a canonical form, there exist analytical models for calculation of the field propagation. Although most surfaces are not like that, runways are often very close to being flat, and analytical models can therefore be used to calculate the field over a flat surface. This thesis is about prediction of electromagnetic field over non-flat runways, and the interest of an analytical model for a flat surface is for comparison with numerical methods. If a numerical method gives results consistent with the analytical model, it means that the numerical method works for a flat surface. This does not mean that the numerical method works for a non-flat surface, however, if it does not give good results for a flat surface, it will most likely not for a non-flat surface either.

2.1 Plane Earth Loss and Source Modeling

The interest of an analytical model for a flat surface, is for being a reference for the numerical methods over a flat surface. For a flat surface, the electromagnetic field does only undergo free-space loss and reflection from the ground, called plane earth loss. The reflection from the ground leads to interference, constructive or destructive. For a stationary transmitter and receiver, placed on a flat surface, the receiver will receive two "waves" originating from the transmitter; the direct wave and the wave reflected on the ground. The interference pattern that occurs depends on the distance between and the height of the transmitter and the receiver. The amplitude of the field at the receiver will be the sum of the direct and the reflected wave, equation (2.1), [Saunders and Aragón-Zavala, 2007, p. 99].

$$A_{\text{total}} = A_{\text{direct}} + A_{\text{reflected}} \tag{2.1}$$

For calculations of the plane earth loss[1], the initial field has to be taken into account. It may be directive and have different amplitude in different directions. The direct and reflected wave will therefore not have same initial amplitude from the transmitter. The amplitude of the reflected wave depends on the angle of the ground-reflected wave from the transmitter, which again depends on the height of the transmitter and receiver. For the setup see figure 2.1. For the plane earth loss calculations, the ground is assumed to be a perfect conductor. Therefore, there will not be any loss associated with the reflections on the ground.

The plane earth loss is calculated at each point along the direct line from the transmitter (Tx) to the receiver (Rx), see figure 2.1. The challenge is to find the amplitude of the ground-reflected wave, the amplitude of the initial field in the direction of θ_{Reflect}. For each point along this line, θ_{Reflect} and α_{tx} change because the reflection point, x_{refl} on the ground changes, and therefore d_{tx} and d_{rx} change too. x_{refl}, d_{tx}, and d_{rx} are then unknown quantities. The distance d_{tx} needs to be found in order to calculate θ_{Reflect}. θ_{Reflect} is of interest because the initial amplitude of the reflected wave depends on its direction from the source. The derivation of θ_{Reflect} with transmitter coordinates (t_x, t_z) and receiver coordinates (r_x, r_z) is given in equation (2.2). Note that in the calculations, the receiver, Rx, represents the current point for the plane earth loss calculations, and the distance

[1]Plane earth loss: The loss associated with a wave traversing over a flat surface.

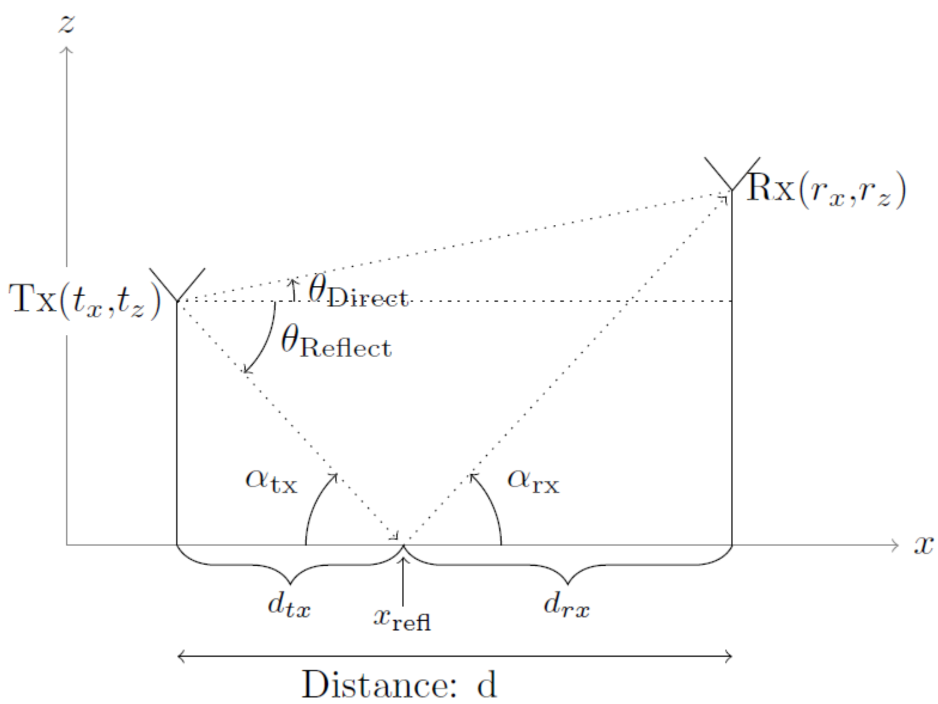


Figure 2.1: Notation for analytical model for plane earth loss.

d, will change accordingly.

$$\begin{aligned}
&\text{From Snell's law: } \alpha_{tx} = \alpha_{rx}\\
&\Rightarrow \tan\left(\frac{t_z}{d_{tx}}\right) = \tan\left(\frac{r_z}{d_{rx}}\right)\\
&\Rightarrow \frac{t_z}{d_{tx}} = \frac{r_z}{d_{rx}}\\
&\Leftrightarrow \frac{t_z}{r_z} = \frac{d_{tx}}{d_{rx}} = a,\ a \in \mathbb{R}\\
&d = d_{tx} + d_{rx} \Leftrightarrow d_{rx} = d - d_{tx}\\
&\Rightarrow a = \frac{d_{tx}}{d - d_{tx}}\\
&\Rightarrow d_{tx} = \frac{ad}{1+a} = \frac{\frac{t_z}{r_z}d}{1+\frac{t_z}{r_z}} = \frac{t_z d}{r_z + t_z}\\
&\Rightarrow \theta_{\text{Reflect}} = \frac{\pi}{2} - \tan^{-1}\left(\frac{d_{tx}}{t_z}\right),\ \frac{d_{tx}}{t_z} = \frac{t_z d}{t_z(r_z+t_z)} = \frac{d}{r_z+t_z}\\
&\Rightarrow \theta_{\text{Reflect}} = \frac{\pi}{2} - \tan^{-1}\left(\frac{d}{r_z+t_z}\right)
\end{aligned} \tag{2.2}$$

For the plane earth loss, the loss of the wave may be expressed in path loss. Due to interference the plane earth loss has dips and peaks. Figure 2.2 shows an example of what the plane earth loss may look like.

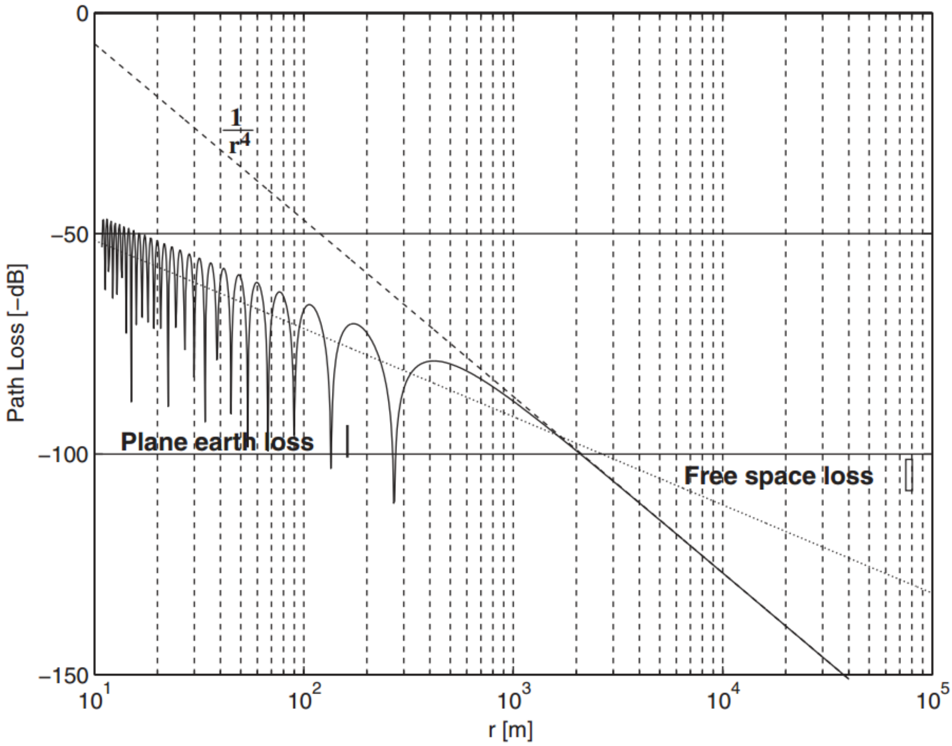


Figure 2.2: The solid line shows the plane earth loss: $f = 900Hz$, transmitter height: $30m$, receiver height: $1.5m$. $r[m]$: the distance along the surface between transmitter and receiver. [Saunders and Aragón-Zavala, 2007, p. 100]

2.2 Numerical Source Modeling

Two sources are proposed; an isotropic source, used in Saunders and Aragón-Zavala [2007, Chapter 5], and a Gaussian source, proposed in Levy [2000, Chapter 5].

2.2.1 Isotropic Source

For an isotropic source the field strength will be the same for all directions. Assuming plane waves, the only difference between the direct and the reflected wave will be phase differences due to different traveling length. The amplitude at the receiver assumes that the ground is a perfect conductor, and can therefore be written according to equation (2.3), [Saunders and Aragón-Zavala, 2007, p. 99], where h_{tx} and h_{rx} are the heights of the transmitter and the receiver, respectively.

$$\begin{aligned} A_{\text{total}} &= A_{\text{direct}} + A_{\text{reflected}} \\ &= A \left| 1 + \exp\left(jk \frac{2h_{rx}h_{tx}}{d} \right) \right|^2 \end{aligned}$$

h_{tx}: Height of transmitter antenna

h_{rx}: Height of receiver antenna

d: Distance between the antennas along the surface, according to figure 2.1

(2.3)

2.2.2 Gaussian Source

A Gaussian source is a source where the field distribution has a Gaussian shape in the far-field, and is given by equation (2.4), [Levy, 2000, p. 40], where A is a normalization constant, β is the half-power beamwidth, and θ the angle from the paraxial direction[1]. This means that the shape of the beam can be entirely determined by the β-parameter. The beam can be tilted by the angle θ_0 by adding an additional term, see equation (2.5), [Levy, 2000, p. 41]. $u(0, z)$ in equation (2.5) is the initial field along the vertical direction. The field strength at the receiver can be found using the relations in equation (2.1) and (2.2).

An advantage of the Gaussian source is that in a polar plot the Gaussian-shaped curve looks exactly like a real antenna beam, without any sidelobes, see figure 2.3 and 2.4. The beam of a real antenna can be modeled by using a combination

[1] Paraxial direction: the direction of propagation

of multiple Gaussian beams with different gain, beamwidth, and tilt. Not all beamshapes can be modeled using a Gaussian beam, however it is quite flexible.

$$B(\theta) = A \exp\left(-2\log(2)\frac{\theta^2}{\beta^2}\right)$$

A: Normalization constant

β: Half-power beamwidth [rad]

θ: The angle from the direction of propagation, from the paraxial direction (2.4)

$$u(0,z) = A\frac{k\beta}{2\sqrt{2\pi\log(2)}}\exp\left(-ik\theta_0 z\right)\exp\left(-\frac{\beta^2}{8\log(2)}k^2(z-z_s)^2\right)$$

z: The height direction

$k = \frac{2\pi}{\lambda}$: Wavenumer, λ is the wavelength (2.5)

θ_0: The tilt of the beam [rad]

z_s: Antenna height

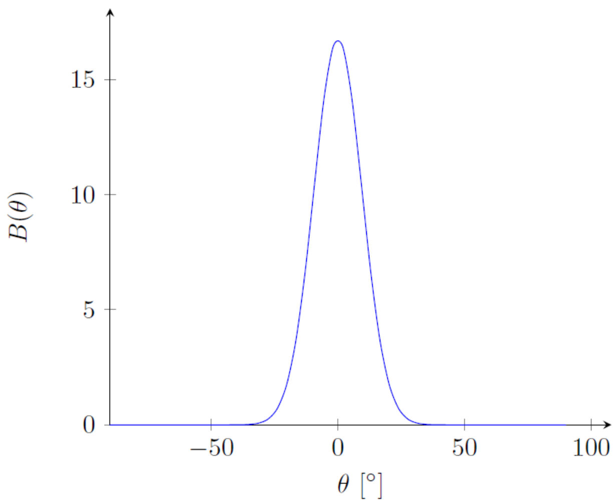


Figure 2.3: Gaussian beam, half-power beamwidth: 40°.

2.2.3 Choice of Source for Numerical Simulations

As will be treated later, the numerical field simulation algorithms have some constraints when it comes to the maximum beam width of the initial field. Therefore,

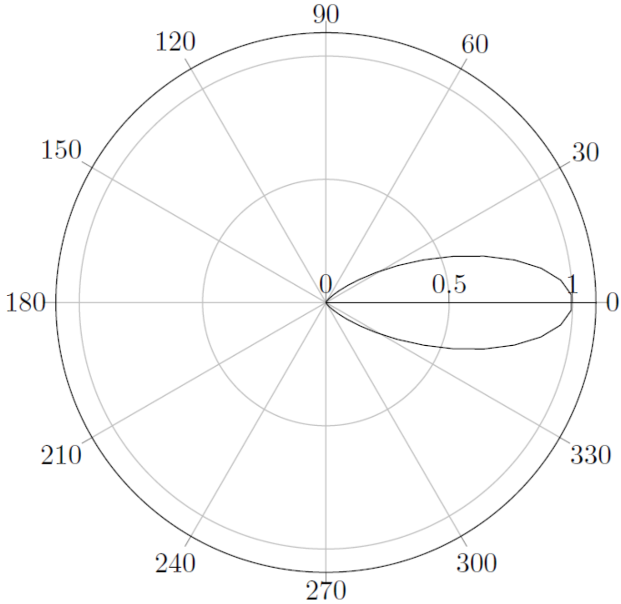


Figure 2.4: Polar plot of the Gaussian beam: half-power beamwidth at ±20°. This is at the edge of the domain of validity for the numerical algorithm, it will be treated later.

a Gaussian source is chosen for the simulations.

Chapter 3

Numerical Methods

There were originally two numerical methods of interest for simulation of electromagnetic fields over irregular terrain, the Integral Equation Model and the Parabolic Equation Method. The Integral Equation Model has a principle that is "intuitive" and easy to understand. However, a few missing links made it not implementable towards the final result, see section 3.1.2. The Parabolic Equation Method on the other hand, uses the scalar wave equation to calculate the electromagnetic field over irregular terrain. This method is implemented with two different algorithms.

Since the surface of a runway can be assumed to be constant in the transverse direction, the numerical methods in this thesis are implemented for simulation of the electromagnetic field in 2D. There is a difference between performing simulations in two and three dimensions. In 2D simulation one dimension is missing, and some of the 2D formulas are therefore different from the 3D ones. The difference between the waves in 3D and 2D is that in 3D the waves are spherical, and in 2D cylindrical. Meaning that in 2D the free-space loss is proportional to $\frac{1}{r}$ instead of $\frac{1}{r^2}$ in 3D. This difference stems from that in the 2D scenario the waves are cylindrical as opposed to the spherical waves in the 3D version.

3.1 The Integral Equation Model

The Integral Equation Model was developed in order to estimate the scattering of electromagnetic waves traveling over irregular terrain, without approximating the terrain to any canonical form.

The principle of the Integral Equation Model is to estimate the field strength of the electromagnetic field based on direct propagation and electromagnetic radiation from induced current on the surface, $\boldsymbol{E}^s(\boldsymbol{r})$, using Maxwell's equations. The induced current at a given point is determined from the direct wave from the transmitter and the current induced on the surface between the given point and the antenna. There will not be any back scattering because the terrain has an undulating profile. Therefore, the given point is the point farthest away from the transmitter that is taken into account.

The Integral Equation Model can be used to predict the field strength for any irregular terrain. However, there are different limitations for the different approaches to solve the problem numerically. Originally, the methods were only stable at low frequencies, at approximately 10 MHz, but the methods have been improved, and today the methods work well up to 144 MHz, [Saunders and Aragón-Zavala, 2007, p. 131]. For higher frequencies, the path loss prediction error increases. The localizer operates in the frequency range of 108 MHz to 112 MHz, the Integral Equation Model should therefore be able to give good results.

The terrain is characterized by a set of sampling points with appropriate separation for describing the terrain sufficiently. In order to fulfill Nyquist's criteria, the maximum distance between two samples is $\frac{\lambda}{2}$, where λ is the wavelength . If a set of sampling points does not fulfill Nyquist's criteria, the set of points needs to be extended. This can be done by interpolating. The way of interpolating depends on the application and assumptions. In the case of a runway, the terrain between two samples is assumed to be plane. The interpolation can therefore be performed using a simple first order interpolation algorithm.

The scattered electric field, $\boldsymbol{E}^s(\boldsymbol{r})$, the field emitted from the induced surface currents, is found using the Maxwell's equations. A common way to calculate $\boldsymbol{E}^s(\boldsymbol{r})$, is to use an intermediate step via the auxiliary vector $\boldsymbol{A}(\boldsymbol{r})$, given by equation (3.1), [Gibson, 2007, p. 12]. $\boldsymbol{E}^s(\boldsymbol{r})$ is given by equation (3.2), [Gibson, 2007, p. 12]. All symbols in the equations are consistent, they have the same meaning in all equations. The explanations of the symbols will only be stated once, and the repeated symbols can also be found in the nomenclature. All symbols in bold are vectors.

$$\begin{aligned}
&\boldsymbol{A}(\boldsymbol{r}) = \mu \iint_S \boldsymbol{J}(\boldsymbol{r'})G(\boldsymbol{r},\boldsymbol{r'})\mathrm{d}\boldsymbol{r'} \\
&G(\boldsymbol{r},\boldsymbol{r'})\text{: Electromagentic Green's function, equation (3.3)} \\
&\boldsymbol{J}(\boldsymbol{r'})\text{: Induced surface current} \\
&\boldsymbol{r'}\text{: Vector from origin to the source point} \\
&\boldsymbol{r}\text{: Vector from origin to the observation point} \\
&\mu\text{: Permeability of the medium}
\end{aligned} \tag{3.1}$$

$$\begin{aligned}
&\boldsymbol{E}^s(\boldsymbol{r}) = -j\omega\boldsymbol{A} - \frac{j}{\omega\mu\epsilon}\nabla(\nabla\cdot\boldsymbol{A}) \\
&\omega = 2\pi f\text{: Angular frequency} \\
&\epsilon\text{: Permittivity of the medium}
\end{aligned} \tag{3.2}$$

For 2-D the electromagnetic Green's function is given by equation (3.3), [Gibson, 2007, p. 10].

$$\begin{aligned}
&G(\boldsymbol{r},\boldsymbol{r'}) \simeq \begin{cases} 1 - j\dfrac{2}{\pi}\log\dfrac{\gamma kr}{2}\ ,\ r\to 0 \\ -\dfrac{j}{4}H_0^{(2)}(kr) = -\dfrac{j}{4}\sqrt{\dfrac{2j}{\pi kr}}\exp\left(-jkr\right)\exp\left(jk\boldsymbol{r'}\cdot\boldsymbol{r}\right)\ ,\ kr\to\infty \end{cases} \\
&\gamma\text{: A constant} \\
&k = \frac{2\pi}{\lambda}\text{: Wavenumber} \\
&H_0^{(2)}(\cdot)\text{: Hankels function of second kind} \\
&r = |\boldsymbol{r'} - \boldsymbol{r}|\text{: The distance between the source and the observation point}
\end{aligned} \tag{3.3}$$

Equation (3.2) includes derivatives that depend on the distance $\boldsymbol{r}$. For large distances, where $kr >> 1$, the terms including the derivatives will be very small compared to the first term, without any derivative, [Gibson, 2007, p. 18]. For most of the points at the surface of the runway $kr >> 1$, therefore, the term including the derivatives can be neglected. The relationship between the scattered electric field and the induced surface current can therefore be calculated using equation (3.4).

$$\boldsymbol{E}^s(\boldsymbol{r}) = -j\omega\boldsymbol{A} \tag{3.4}$$

Numerically, the integrals are carried out as sums. The numerical implementation by Brennan and Cullen [1998] in section 3.1.1.3, uses the relationship from equation

(3.4) directly.

3.1.1 Numerical Implementation

3.1.1.1 Assumptions

In order to be able to give good results, the following assumptions regarding the terrain are taken, [Hviid et al., 1995]:

- 2-D surface, no variations in the transverse direction
- Smooth surface, relative to the wavelength
- The surface is a perfect magnetic conductor
- Vertical polarization
- Grazing incidence angle
- No back scattering

2-D surface means that only the vertical direction and the direction of propagation are considered, there are no surface variations in the transverse direction. Smooth surface means that the height variations relative to the wavelength are slow. In reality, there are no perfect magnetic conductors. However, it is an appropriate assumption for runways. In the case of vertical polarization, the reflection coefficients for perfect magnetic conductors is -1. For a real ground and grazing incidence[1], the reflection coefficient approaches -1, Hviid et al. [1995]. In the case of a localizer transmitting over a runway, the angles will be grazing. According to Hviid et al. [1995] there is hardly any difference between the fields of horizontal and vertical polarization for grazing incidence angle over a perfect magnetic conductor in the microwave region. The localizer signals are horizontally polarized, and they are right outside the microwave region. The microwave region extends from approximately 300 MHz to 300 GHz, depending on the definition. The localizer transmit signals from 108 MHz to 112 MHz. It is therefore a good chance that the method is valid for localizer signals. It is however necessary to compare estimated results with measurements or other verified methods. In the terms of reflection, the surface may be modeled as a perfect conductor. However, the induced current at the surface will not behave as if the surface was a perfect magnetic conductor. The induced current at the surface will be attenuated with increasing distance. The method also assumes that there will be no back scattering, meaning that no point

[1] Grazing incidence: Small incident angle

behind the observation point will contribute to the field strength at the observation point. The same for the induced current at a given point on the surface; it only depends on radiation from induced current at points between the given point and the transmitter, plus the direct wave from the transmitter.

3.1.1.2 Implementation Based on Hviid et al. [1995]

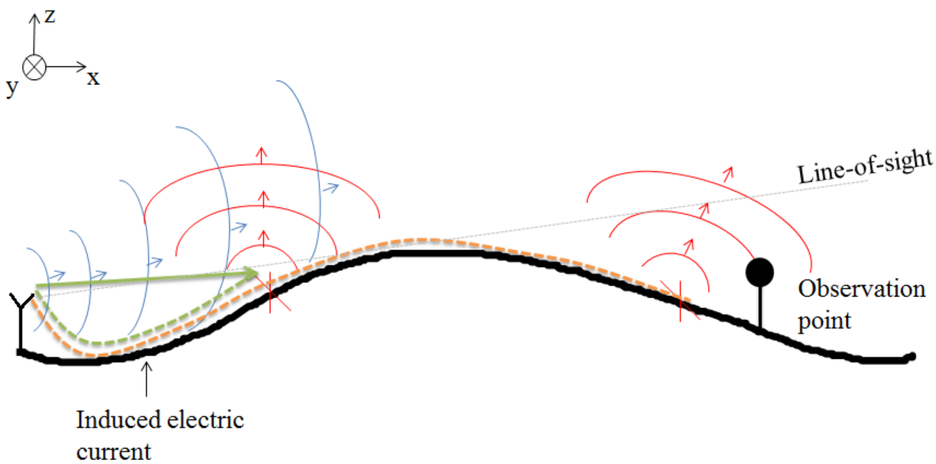


Figure 3.1: Illustration of the principle of the Integral Equation Method.

As already mentioned, the field strength in a given point is determined from the direct electric field and the field radiated from the induced current at the surface, see illustration of the principle, figure 3.1. Equation (3.5) shows the expression for the equivalent source current at a given point n, M_n^s. Equation (3.5) does not include any weighting based on the distance between the two points. Equation (3.5) shows that the induced current at a given point n, is the sum of the direct wave and the induced currents at previous points, weighted by $f(n, m)$, where point m is a previous point. The magnitude of the induced current depends on the angle between the incoming radiation and the surface normal. This applies both for the direct propagation from the antenna and the radiation from induced current, and is determined by $f(n, m)$ in equation (3.5). The $f(n, m)$ is a weighting function based on direction between the two points m and n, and the surface normal in point n. R_1, R_2, r_1 and r_2 in formula (3.5) are according to figure 3.2.

$$M_n^s = TM_{i,n}^s + \frac{T}{4\pi}\sum_{m=0}^{n-1} M_m^s f(n,m)\Delta x_m,$$
$$f(n,m) = (\vec{n}\cdot\vec{r_2})\frac{jk}{R_2}\sqrt{\lambda\frac{R_1R_2}{R_1+R_2}}e^{-j(kR_2+\pi/4)}\frac{\Delta l_m}{\Delta x_m}$$
$k = \frac{2\pi}{\lambda}$: wavenumber

r_2: vector from point m to n.

R_1: Distance from antenna to point m for $z_m = 0$.

R_2: Distance from point m for $z_m = 0$ to point n.

Δl_m: Increment distance along the surface, [Saunders and Aragón-Zavala, 2007, p. 130]

Δx_m: Increment distance along the x-axis

[Hviid et al., 1995]

(3.5)

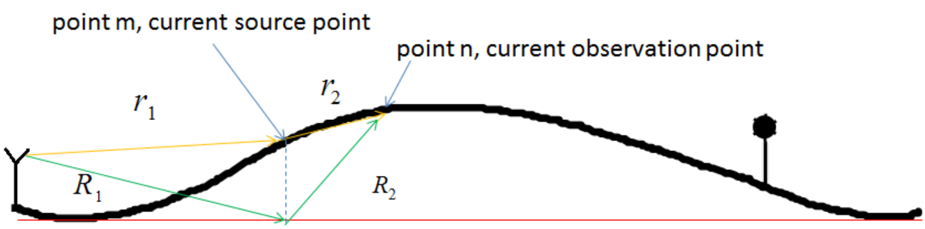


Figure 3.2: Geometry of the scattering problem for the Integral Equation Method.

3.1.1.3 Implementation Based on Brennan and Cullen [1998]

This section describes a method to calculate the scattered electric field in 2D using the method proposed in Brennan and Cullen [1998]. The method uses equation (3.4) directly in order to calculate the induced surface currents. Inserting equation

(3.1) into (3.4) results in equation (3.6).

$$
\begin{aligned}
\boldsymbol{E}^s(\boldsymbol{r}) &= -j\omega\mu \int\limits_S G(\boldsymbol{r},\boldsymbol{r'})\boldsymbol{J}(\boldsymbol{r'})\mathrm{d}\boldsymbol{r} \\
&= \begin{cases} -j\dfrac{\mu}{4}\displaystyle\int\limits_S \boldsymbol{J}(\boldsymbol{r'})\left[1 - j\dfrac{2}{\pi}\log\dfrac{\gamma kr}{2}\right]\mathrm{d}\boldsymbol{r'}, \text{ near-field} \\ -j\dfrac{\mu}{4}\displaystyle\int\limits_S \boldsymbol{J}(\boldsymbol{r'})\sqrt{\dfrac{2j}{\pi kr}}\mathrm{e}^{-jkr}\mathrm{e}^{jk\boldsymbol{r'}\cdot\boldsymbol{r}}\mathrm{d}\boldsymbol{r'}, \text{ far-field} \end{cases}
\end{aligned}
\tag{3.6}
$$

Discretization of the far-field of equation (3.6) leads to equation (3.7).

$$
\begin{aligned}
&\boldsymbol{E}^s(x_r, z_r) = -\frac{j\omega}{4}\sum_{n=0}^{N}\boldsymbol{J}(x_n, z_n)\sqrt{\frac{2}{\pi k d_n}}\mathrm{e}^{-j(jkd_r)}\mathrm{e}^{jk\boldsymbol{d}_r\boldsymbol{d}_s} \\
&[x_r, z_r]\text{: Receiving point coordinates} \\
&[x_n, z_n]\text{: Current transmitter point coordinates} \\
&d_n = \sqrt{(x_r - x_n)^2 + (z_r - z_n)^2} \\
&\boldsymbol{d}_r = [x_r, z_r],\ d_r = \sqrt{x_r^2 + z_r^2} \\
&\boldsymbol{d}_s = [x_n, z_n]
\end{aligned}
\tag{3.7}
$$

According to Brennan and Cullen [1998] this method can also be used to compute the scattered field at successive points at the surface, assuming surface current only. The relationships above also state that by knowing the scattered electric field, it is possible to find the induced surface current. Equation (3.7) shows that the scattered field at a given point is the sum of all "scattering contributions" from surface points with induced current. The far-field expression used for calculating the induced surface current in the Brennan and Cullen [1998] article is very similar to equation (3.7). In Brennan and Cullen [1998] the relationship between the induced surface current at a given point, the induced surface current at the previous surface points, and the incident electric field is stated in a matrix system, given in equation (3.8). The Z-matrix in equation (3.8) is a transition matrix between the induced surface current and the incident electric field, containing coefficients similar to the coefficients in equation (3.7). J_n is a vector containing the induced surface currents, and V_m a vector containing the incident electric fields.

$$\begin{bmatrix} \vdots \\ \vdots \\ V_m \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots\ddots & & & & \\ - & \ddots & & 0 & \\ - & Z_{mn} & \ddots & & \\ - & - & - & \ddots & \\ \vdots\cdots & - & - & - & \ddots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ J_n \\ \vdots \\ \vdots \end{bmatrix} \tag{3.8}$$

3.1.1.3.1 Principle - Forward Scattering Only The surface is divided into D sections of length Δs. Each section is then described by D uniformly separated points, $\rho_1, \cdots, \rho_D$, see figure 3.3.

The field received in point ρ_m is the sum of the direct field from the transmitter,

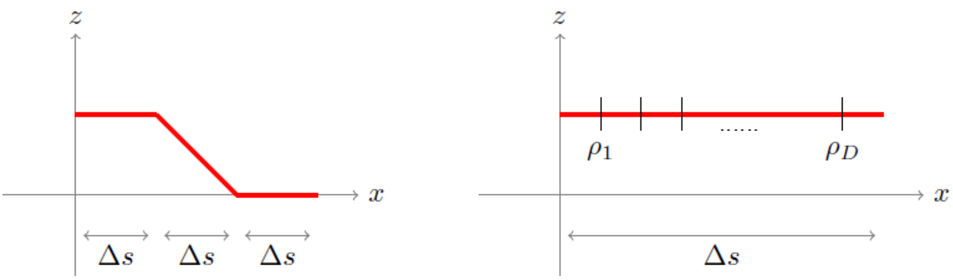


Figure 3.3: The surface is divided into D sections of width Δs. Each of the D sections are described by D uniformly spaced points within each section.

scattered field from far-field regions, and scattered field from near-field regions, see figure 3.4. From [Balanis, 2005, p.34], the near- and far-field regions are given by equation (3.9). The direct field from the transmitter does only have free-space loss, but the field strength also depends on the gain and the directivity of the transmitter. The near-field contribution is determined from the points within the near-field distance from the current point, ρ_m. The determination of the far-field contribution at ρ_m is done by phase- and amplitude shifting of the far-field contribution at the center point of the group, ρ_M.

$$\begin{aligned}&\text{Near-field region: } R < 2\frac{D^2}{\lambda}\\&\text{Far-field region: } R > 2\frac{D^2}{\lambda}\\&R\text{: Distance from radiation source}\\&D\text{: Largest dimension of radiation source}\end{aligned} \tag{3.9}$$

For a group of D collocation points, a vector V_m contains the field strength

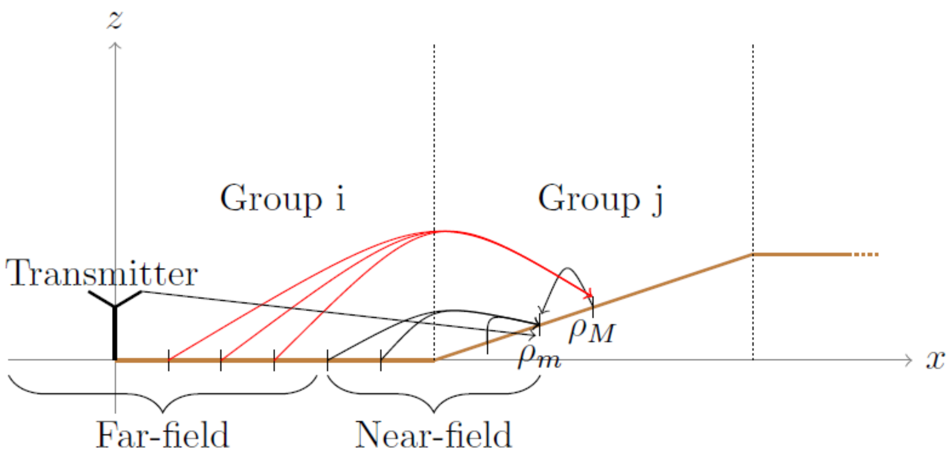


Figure 3.4: The induced surface current at a given point ρ_m depends on the direct field from the transmitter, and the near- and far-field from previous scattering points.

at a given point, ρ_m, in a given group, see equation (3.10), illustrated in figure 3.4.

$$
\begin{aligned}
V_m &= \sum_{n=1}^{N} Z_{mn} J_n \\
&= \sum_{i \in FF_j} F^i_{Mm} \sum_{n \in i} Z_{Mn} J_n + \sum_{i \in NF_j} \sum_{n \in i} Z_{mn} J_n
\end{aligned}
$$

J: Vector of basisfunction coefficients.

Z: matrix, Z_{mn}: field contribution from point n to point m.

F^i_{Mm}: Phase- and amplitude-shifting function for the far-field contribution.

FF_j: Far-field of group j.

NF_j: Near-field of group j.

(3.10)

For forward scattering the Z-matrix is a lower triangular matrix, because only previous points are included in the calculations for the field at a given point. The number of previous points will (obviously) increase by one per point when moving along the x-axis, see equation (3.8). The surface is divided into groups in order to simplify the calculations, the Z_{mn} matrix is therefore composed according to (3.11).

$$
Z = \begin{pmatrix}
Z_{11} & 0 & 0 & 0 & & & & \\
\vdots & \ddots & 0 & 0 & & & \mathbf{0} & \\
\vdots & & \ddots & 0 & & & & \\
Z_{D1} & \cdots & \cdots & Z_{Dk} & & & & \\
Z_{M\,FF_j} & Z_{M\,FF_j} & Z_{mx_l\,NF} & Z_{mx_k\,NF} & Z_{mn} & 0 & 0 & 0 \\
Z_{M\,FF_j} & Z_{M\,FF_j} & Z_{Mx_l\,NF} & Z_{Mx_k\,NF} & Z_{Mn} & \text{Group j} & & 0 \\
Z_{M\,FF_j} & Z_{M\,FF_j} & \cdots & \cdots & \cdots & & \cdots & 0 \\
 & & & & \cdots & \cdots & \cdots & \cdots \\
Z_{M\,FF_j} & Z_{M\,FF_j} & \cdots & \cdots & & & & \\
 & & & & & & & \\
 & & \text{Green: Far-field of group j} & & & & & \\
 & & \text{Red: Near-Field of } Z_m & & & & &
\end{pmatrix}
$$

(3.11)

The near-field coefficients are determined from an equation for the near-field.

However, hardly any points will be in the near-field region because of its extension. In far-field, the element Z_{mn} of Z is given by equation (3.12).

$$Z_{mn} = Z_{Mn} A_{nMm} e^{-j\phi_{nMm}}, \text{ for } n \text{ in the far-field region.}$$
$$A_{nMm} = \sqrt{\frac{R}{R'}} = \left(\frac{R'^2}{R^2}\right)^{-\frac{1}{4}} = \left(1 + \frac{s^2 - 2Rs\cos\alpha}{R^2}\right)^{-\frac{1}{4}}$$
$$\phi_{nMm} = \beta\left(R' - R\right) = \beta R \left(\left(1 + \frac{s^2 - 2Rs\cos\alpha}{R^2}\right)^{\frac{1}{2}} - 1\right) \tag{3.12}$$

All values are according to figure 3.5

α: The angle between s and R in figure 3.5

β: A constant

The ratio $1 + \frac{s^2 - 2Rs\cos\alpha}{R^2}$ is the squared ratio between R' and R, derived

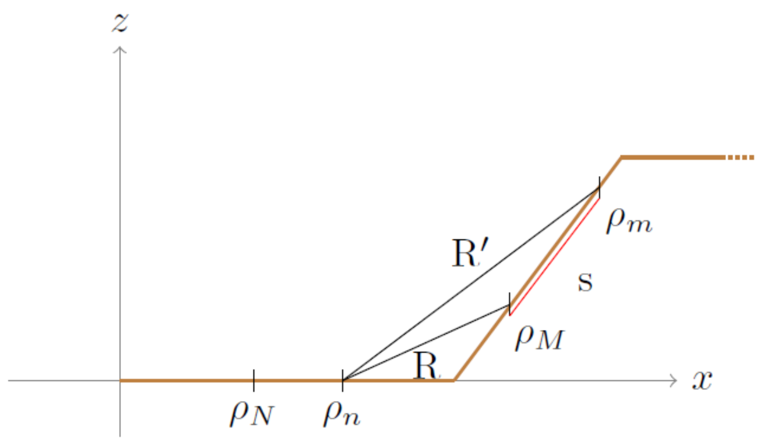


Figure 3.5: Geometrical aid for equation (3.12).

from the cosine-rule, to find R', $R'^2 = R^2 + s^2 - 2Rs\cos\alpha$. This gives $\frac{R'^2}{R^2} = 1 + \frac{s^2 - 2Rs\cos\alpha}{R^2}$, and refers to the difference in distance between ρ_n and ρ_m, and ρ_n and ρ_M, according to figure 3.5.

If two groups are within the close angles, seen from the third group, the two groups can be merged into one group for the far-field contribution calculations for the third group. The angles are measured with respect to the selected center point in a given group. The Z-matrix will the look like the illustration in equation (3.13).

$$
Z = \begin{pmatrix}
Z_{11} & 0 & 0 & & & & & & \\
\vdots & \text{Group i} & 0 & & & \mathbf{0} & & & \\
Z_{D1} & \cdots & Z_{Dk} & & & & & & \\
 & & & \vdots\ddots & 0 & 0 & & & \\
 & & & \cdots & \text{Group j} & 0 & & & \\
 & & & \cdots & \cdots & \cdots & & & \\
Z_{MFF_k} & Z_{MFF_k} & Z_{mx_k NF} & & & & Z_{mn} & 0 & 0 \\
Z_{MFF_k} & Z_{MFF_k} & Z_{Mx_k NF} & & & & Z_{Mn} & \text{Group k} & 0 \\
Z_{MFF_k} & Z_{MFF_k} & \cdots & & & & \cdots & \cdots & \cdots
\end{pmatrix} \tag{3.13}
$$

3.1.2 Remaining Issues

The principle of how to calculate the induced field along the surface is clear, and fully possible to implement. Due to time constraint, two important issues are remaining, and left for future work:

- Which parts of the field along surface contribute to the total field at the receiver? All of the points in line-of-sight from the receiver? Or just the last ones?
- What kind of ”source” will the surface be? Isotropic? Directive?

3.2 The Parabolic Equation Method

The Parabolic Equation Method is a numerical method that can be used to calculate the electromagnetic field over a surface. It can handle both flat and irregular surface. The method calculates the field for all heights of interest and at all points of interest along the surface. The field at a given point along the surface can be entirely determined from the previous point of consideration along the surface. The setup and coordinate system used is according to figure 3.6.

The Parabolic Equation Method is derived from the scalar wave equation, which again is derived from Maxwell's equations given in equation (3.14). The derivation

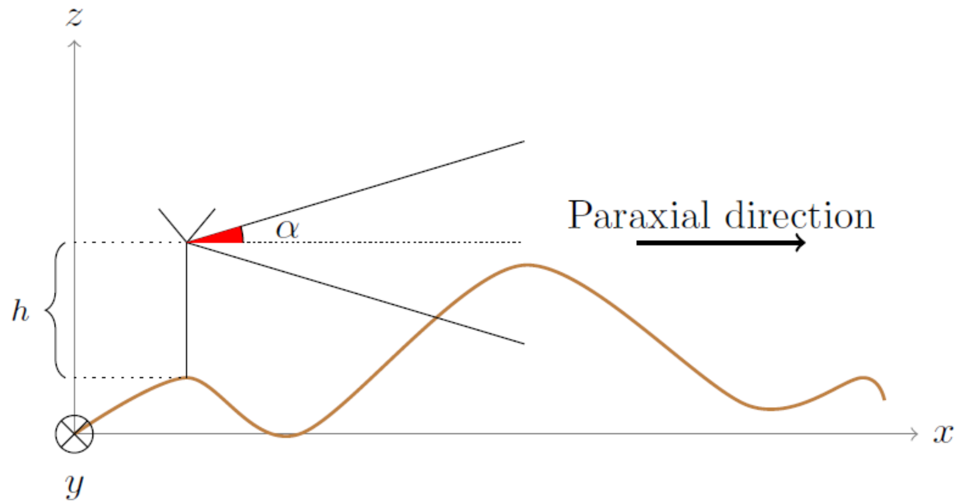


Figure 3.6: The setup for the Parabolic Equation Method.
h: Transmitter antenna height.

of the scalar wave equation is given below, based on Lee [2013, p.389-390]. The propagation of the waves is assumed to be in free-space, without any sources.

$$
\begin{aligned}
&\nabla \times \boldsymbol{H} = \boldsymbol{j_c} + \epsilon \frac{\partial \boldsymbol{E}}{\partial t} \\
&\nabla \times \boldsymbol{E} = -\mu \frac{\partial \boldsymbol{H}}{\partial t} \\
&\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon} \\
&\nabla \cdot \boldsymbol{H} = 0 \\
&\boldsymbol{E} = \boldsymbol{E}(x, y, z; t)\text{: Electric field} \\
&\boldsymbol{H} = \boldsymbol{H}(x, y, z; t)\text{: Magnetic field} \\
&\boldsymbol{j_c}\text{: Current density at the given point} \\
&\epsilon\text{: Absolute permittivity} \\
&\mu\text{: Absolute permeability} \\
&\rho\text{: Charge density at the given point} \\
&\text{Kouzaev [2011]}
\end{aligned}
\tag{3.14}
$$

Assuming time harmonic fields, gives the relationship in equation (3.15).

$$\begin{aligned} \boldsymbol{E}(x,y,z;t) &= \boldsymbol{E}_0(x,y,z)e^{(j\omega t)} \\ \boldsymbol{H}(x,y,z;t) &= \boldsymbol{H}_0(x,y,z)e^{(j\omega t)} \end{aligned} \tag{3.15}$$

Applying the relationship from equation (3.15) in (3.14), results in equation (3.16).

$$\begin{aligned} \nabla \times \boldsymbol{E} &= -\mu \frac{\partial \boldsymbol{H}}{\partial t} \\ &= -j\omega\mu\boldsymbol{H} \end{aligned} \tag{3.16}$$

Applying curl to equation (3.16), results in equation (3.17).

$$\begin{aligned} \nabla \times \nabla \times \boldsymbol{E} &= j\omega\mu\nabla \times \boldsymbol{H} \\ &= j\omega\mu\nabla \times (j\omega\epsilon\boldsymbol{E}), \text{ there is no current source} \\ &= -\omega^2\mu\epsilon\boldsymbol{E} \end{aligned} \tag{3.17}$$

Using vector operator identity gives $\nabla \times \nabla \times \boldsymbol{E} = \nabla\nabla \cdot \boldsymbol{E} - \nabla^2\boldsymbol{E}$. Since there is no source at the given point, $\rho = 0$. Therefore, $\nabla \cdot \boldsymbol{E} = 0$ and $\nabla\nabla \cdot \boldsymbol{E} = 0$. In addition, the Laplace operator, $\nabla^2 = \left(\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2}, \frac{\partial^2}{\partial z^2}\right)$, is independent of time. This leads to equation (3.18).

$$\begin{aligned} &\nabla^2\boldsymbol{E} = -\omega^2\mu\epsilon\boldsymbol{E} \\ &\Leftrightarrow e^{(j\omega t)}\nabla^2\boldsymbol{E}_0 = -\omega^2\mu\epsilon\boldsymbol{E}_0 e^{(j\omega t)} \\ &\Rightarrow \nabla^2\boldsymbol{E}_0 = -\omega^2\mu\epsilon\boldsymbol{E}_0 \\ &\Rightarrow \frac{\partial^2\boldsymbol{E}_0}{\partial x^2} + \frac{\partial^2\boldsymbol{E}_0}{\partial y^2} + \frac{\partial^2\boldsymbol{E}_0}{\partial z^2} + \omega^2\mu\epsilon\boldsymbol{E}_0 = 0 \\ &\text{Accounting for 2D propagation } \Rightarrow \frac{\partial^2\boldsymbol{E}_0}{\partial x^2} + \frac{\partial^2\boldsymbol{E}_0}{\partial z^2} + \omega^2\mu\epsilon\boldsymbol{E}_0 = 0 \end{aligned} \tag{3.18}$$

Exploring the properties of the term $\omega^2\mu\epsilon$ in equation (3.18), leads to the scalar wave equation in equation (3.19). For linear polarization, the electric field will only have one component. It means that the electric field in equation (3.19), $\boldsymbol{E}_0$, is a scalar.

$$
\begin{aligned}
&\text{For propagation in vacuum: } \epsilon_r = \mu_r = 1 \\
&\text{The speed-of-light in vacuum: } c = \frac{1}{\sqrt{\mu_0 \epsilon_0}} \text{ [Lee, 2013, p. 390]} \\
&\text{The speed-of-light in a given medium: } v = \frac{c}{\sqrt{\mu_r \epsilon_r}} \text{ [Lee, 2013, p. 389]} \\
&\text{The index of refraction: } n = \frac{c}{v} = \sqrt{\mu_r \epsilon_r} \\
&\Rightarrow \omega^2 \epsilon \mu = \frac{\omega^2}{c^2} n^2 = \left(\frac{2\pi}{\lambda}\right)^2 n^2 = k^2 n^2 \\
&\Rightarrow \frac{\partial^2 \boldsymbol{E}_0}{\partial x^2} + \frac{\partial^2 \boldsymbol{E}_0}{\partial z^2} + k^2 n^2 \boldsymbol{E}_0 = 0
\end{aligned}
\tag{3.19}
$$

The scalar wave equation in equation (3.19) is formally written in equation (3.20). For horizontal polarization $\psi(x, z)$ describes the electric field, $\psi(x, z) = E_y(x, z)$, and for vertical polarization $\psi(x, z)$ describes the magnetic field, $\psi(x, z) = H_y(x, z)$, [Levy, 2000, p. 4]. Since the localizer transmits horizontally polarized signals, $\psi(x, z)$ will here represent the electric field. In order for the following theory to be correct, it is assumed that the time-dependency of the electric field is e^{jwt}. However, the Parabolic Equation Method deals with how the field propagates in space, not in time.

$$
\begin{aligned}
&\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial z^2} + k^2 n^2 \psi = 0 \\
&x\text{: Direction of propagation} \\
&z\text{: Height with respect to the coordinate system} \\
&k\text{: Wave number in vacuum, } \frac{2\pi}{\lambda} \\
&n\text{: Refractive index, function of x and z, slowly varying}
\end{aligned}
\tag{3.20}
$$

The setup for the method is forward propagation along the x-axis, the z-axis represents the height, and the y-axis represents the transverse direction. The wave propagation problem is assumed to be a 2D problem, which means that there are no variations in the transverse direction, along the y-axis, see figure 3.6. The method also assumes that the field propagation is directive, paraxial propagation, propagation at small angles from the preferred direction. Referring to figure 3.6, this means that α is small.

In order to account for a wave that is slowly varying in the direction of propagation, a reduced function, $u(x, z) = \mathrm{e}^{jkx}\psi(x, z)$ is introduced, [Levy, 2000, p. 5]. Filling $u(x, z)$ into the scalar wave equation, equation (3.20), gives the scalar wave equation for $u(x, z)$, equation (3.21). By factorizing the equation and accounting for forward propagating waves only, the equation can be reduced to the standard parabolic equation, equation (3.22), [Levy, 2000, p. 10]. For the derivation, see appendix B, section B.1. The reason to account for forward propagating waves only, is that in the case of the localizer on a runway there will only be forward propagating waves, because there are hardly any backward reflections from the ground.

$$\frac{\partial^2 u}{\partial x^2} + i2k\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial z^2} + k^2(n^2 - 1) = 0 \tag{3.21}$$

$$\frac{\partial^2 u}{\partial z^2}(x, z) + i2k\frac{\partial u}{\partial x}(x, z) + k^2(n^2(x, z) - 1) = 0 \tag{3.22}$$

The derivation of equation (3.22) includes a Taylor series expansion. The order of magnitude of the error is given by the first neglected term in the series expansion, and is given by equation (3.23), (Levy [2000], p. 10). Equation (3.23) and figure 3.7 show that the error of the approximation increases when α increases. However, as long as α is relatively small, the error will remain small. For $\alpha = 20°, \sin^2(\alpha) \simeq 0.12$. From $\alpha = 20°$ and up, the error increases relatively fast.

$$\frac{1}{k^2}\left|\frac{\partial^2 u}{\partial z^2}\right| = \sin^2(\alpha) \tag{3.23}$$

α: angle according to figure 3.6

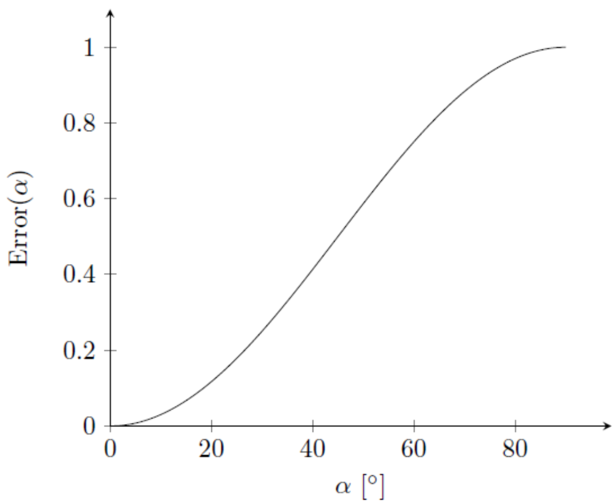


Figure 3.7: The error due to approximation of square-root operator as a function of the angle from the paraxial direction.

3.2.1 Use of Fourier Transform for Solving the Scalar Wave Equation

For plane waves propagating in vacuum, refractive index $n = 1$, the Fourier transform can be used as a mean to solve the differential equation, the standard parabolic equation, equation (3.22). The idea is to transform the problem into the Fourier domain, solve the equation in the Fourier domain, and transform the solution back to the original domain. The reason to change domain for solving the differential equation is that the problem might appear simpler, and for this case it does appear simpler, [Levy, 2000, p. 13]. The definitions of the Fourier transform and the inverse Fourier transform, are given in appendix A, section A.1.

The Fourier transform is a linear operator, and the Fourier transform of the standard parabolic equation for a wave propagating in vacuum can therefore be written according to equation (3.24).

$$\mathcal{F}\left\{\frac{\partial^2 u}{\partial z^2}(x,z) + i2k\frac{\partial u}{\partial x}(x,z) = 0\right\} \Leftrightarrow -4\pi^2 p^2 U(x,p) + i2k\frac{\partial U}{\partial x}(x,p) = 0 \quad (3.24)$$

Equation (3.24) is a homogeneous first order differential equation, and has a closed form solution, see equation (3.25).

$$U(x,p) = \mathrm{e}^{-\frac{i2\pi^2 p^2 x}{k}} U(0,p) \quad (3.25)$$

$u(x,z)$ can be found by taking an inverse Fourier transform of equation (3.25), resulting in equation (3.26). Equation (3.26) shows that the field at any point depends on the initial field, [Levy, 2000, p. 15].

$$u(x,z) = \mathcal{F}^{-1}\left\{\mathrm{e}^{-\frac{i2\pi^2 p^2 x}{k}} \mathcal{F}\left\{u(0,z)\right\}\right\} \quad (3.26)$$

3.2.2 Split-Step Algorithm - Flat Surface

The Split-Step Algorithm (SSA) is based on the solution from the standard differential equation, using the result from equation (3.26) almost directly, in order to calculate the field. The appropriate Fourier transform to use in order to calculate the field, is the Fourier Sine Transform, [Levy, 2000, p. 27]. The expression of the analytical and discrete Fourier Sine Transform can be found in appendix A, section A.2 and A.3, respectively.

The field strength at the current point depends on the previous point. The numerical implementation of the split-step algorithm can be done by using the equations below, equation (3.27) and (3.28), [Levy, 2000, p. 30].

$$u(x+\Delta x) = \mathrm{e}^{\frac{ik(n^2-1)}{2}} \mathcal{S}\{P'\mathcal{S}\{u(x)\}\} \quad (3.27)$$

$$P' = \exp\left(\frac{i\pi^2 l^2 \Delta x}{2kL^2}\right) \quad (3.28)$$

In equation (3.27) the inverse Fourier sine transform is not used, because $\mathcal{S}^{-1} = 4\mathcal{S}$ when using the definition of the Fourier sine transform in appendix A, section A.2, [Levy, 2000, p. 25]. The "interaction" between the different heights are done in the Fourier sine transform, where the Fourier transform is calculated with respect to the z-values, the height.

The algorithm does not converge for a small Δz , because the difference between two neighboring points is so small so that numerically there will not be any, or hardly be any, difference at all.

3.2.3 Finite-Difference Method - Flat Surface

The Finite-Difference Method (FDM) is based on the standard differential equation directly, equation (3.22), using the numerical approximations to the differentials, with finite difference approximations for the first and second derivatives, given in appendix A, section A.4. By inserting the numerical approximations directly, it turns out to be a linear equation with three unknowns, that can be extended into a set of N unknowns with N equations. To simplify the notation $u(x_m, z_j) = u_m^j$.

$$\frac{\partial^2 u}{\partial z^2}(x,z) + 2ik\frac{\partial u}{\partial x}(x,z) + k^2(n^2(x,z)-1)u(x,z) = 0 \quad (3.29)$$

In order for the solution to propagate, the midpoint between two points along the x-axis, ξ_m, is considered, [Levy, 2000, p. 36], see figure 3.8. ξ_m is defined according equation (3.30).

$$\xi_m = \frac{x_{m-1} + x_m}{2} \quad (3.30)$$

Numerically differentiating equation (3.29) in point ξ_m^j, setting $b = 4ik\frac{\Delta z^2}{\Delta x}$ and $a_m^j = k^2({n_m^j}^2 - 1)\Delta z^2$, and then rearranging the equation, leads to equation (3.31). For derivation of the equation see appendix B, section B.2.

$$u_m^{j-1} + u_m^j\left(-2+b+a_m^j\right) + u_m^{j+1} = -u_{m-1}^{j-1} + u_{m-1}^j\left(2+b-a_m^j\right) - u_{m-1}^{j+1} \quad (3.31)$$

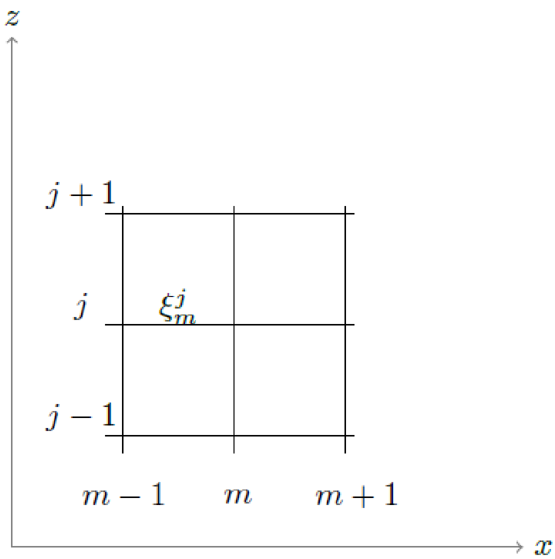


Figure 3.8: The geometry for the Finite-Difference Method.

Equation (3.31) shows that the field at a given point m along the x-axis, can be entirely determined from field at range $m-1$ along the x-axis. Equation (3.31) also shows that there are three heights involved, which means that this is one equation with three unknowns. However, since this relationship is true for any height, it leads to N equations with N unknowns, and can be "summarized" in a linear matrix system, equation (3.32), [Levy, 2000, p. 38], where $\alpha_m^j = -2+b+a_j^m$ and $\beta_m^j = 2+b-a_m^j$. U_{m-1} contains the u-values at range $m-1$ along the x-axis and all heights of interest along the z-axis. U_m is the u-values at range m along the x-axis, and is the unknown that is to be determined.

$$\begin{bmatrix} 1 & & & & \\ 1 & \alpha_1^m & 1 & & \\ & & \ddots & & \\ & & 1 & \alpha_{N-1}^m & 1 \\ & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_m \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ -1 & \beta_1^m & -1 & & \\ & & \ddots & & \\ & & -1 & \beta_{N-1}^m & -1 \\ & & & & -1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_{m-1} \\ \vdots \\ \vdots \end{bmatrix},$$

$$U_m = \begin{bmatrix} u_0^m \\ u_1^m \\ \vdots \\ u_{N-1}^m \\ u_N^m \end{bmatrix}, \; U_{m-1} = \begin{bmatrix} u_0^{m-1} \\ u_1^{m-1} \\ \vdots \\ u_{N-1}^{m-1} \\ u_N^{m-1} \end{bmatrix} \tag{3.32}$$

The field at a given range m, depends on the field at range $m-1$. The field at range $m-1$ depends on the field at range $m-2$, and so on. Therefore, the field at range m, can be entirely determined from the initial field, just by knowing the initial field U_0 and the matrices containing the β_m^j values. In practice, the β-values are range independent along the x-axis, and the field at range m can therefore be expressed as a function of the initial field, according to equation (3.33). This works for a flat surface only, because the surface is constant between the initial field and the point at range m.

$$\begin{bmatrix} \vdots \\ \vdots \\ U_m \\ \vdots \\ \vdots \end{bmatrix} = \left(\begin{bmatrix} 1 & & & & \\ 1 & \alpha_1^m & 1 & & \\ & & \ddots & & \\ & & 1 & \alpha_{N-1}^m & 1 \\ & & & & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 1 & & & & \\ -1 & \beta_1^m & -1 & & \\ & & \ddots & & \\ & & -1 & \beta_{N-1}^m & -1 \\ & & & & -1 \end{bmatrix} \right)^m \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_0 \\ \vdots \\ \vdots \end{bmatrix} \tag{3.33}$$

The simulated results will not be valid unless equation (3.33) converges as $m \to \infty$. The convergence depends on the step size in the x- and z-direction, Δx and Δz, respectively. Numerical results show that equation (3.33) is neutrally stable[1] for any value of Δx and Δz in the relevant interval, Δx and $\Delta z \in [0.5; 1.3]\, m$. Values outside this interval are not tested.

[1]Neutral stability: The absolute value of the largest eigenvalue of the matrix in question equals one, $|\lambda_{max}| = 1$, [Strang, 2006, p. 259].

3.2.4 Mathematical Aspects and Simplifications

The numerical efficiency of the algorithm can be increased by exploring and using the properties of the matrices. The tri-diagonal matrix in equation (3.33) containing the αs is denoted A, and the tri-diagonal matrix containing the βs is denoted B. This results in equation (3.34), provided that A can be inverted.

$$\begin{aligned} U_m &= \left(A^{-1}B\right)^m U_0 \\ &= C^m U_0 \end{aligned} \tag{3.34}$$

A matrix M can be inverted if all the eigenvalues, $\lambda_m \neq 0$, if A is non-singluar. If the inverse does not exist, a pseudo-inverse can be used instead. The computation of C^m can be very simplified by diagonalization of C. C is diagonalizable if all the associated eigenvectors of C are linearly independent. If C can be diagnoalized, it can be written on the following form, equation (3.35), where Λ is a diagonal matrix containing the eigenvalues of C, and S the eigenvectors of C.

$$C = S\Lambda S^{-1} \tag{3.35}$$

$$\begin{aligned} C^m &= \left(S\Lambda S^{-1}\right)\cdots\left(S\Lambda S^{-1}\right) \\ &= S\Lambda^m S^{-1} \end{aligned} \tag{3.36}$$

Equation (3.36) shows that the difference between calculating C^m and C is raising Λ to the m'th power. Since Λ is a diagonal matrix, raising the Λ to the m'th power, equals raising each element to the m'th power, equation (3.37)

$$\begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \ddots & & \\ & & & \lambda_{N-1} & \\ & & & & \lambda_N \end{bmatrix}^m = \begin{bmatrix} \lambda_1^m & & & & \\ & \lambda_2^m & & & \\ & & \ddots & & \\ & & & \lambda_{N-1}^m & \\ & & & & \lambda_N^m \end{bmatrix} \tag{3.37}$$

The diagonalization reduces the number of operations, leading to less memory usage and a more efficient algorithm. The reason is that, if C is a diagonalizable matrix and C^m is to be calculated, only the diagonalized matrix needs to be raised to the power m, see equation (3.36) and (3.37). The diagonalization procedure, does also provide a method to check for stability of the algorithm, before calculating the field. As already mentioned, the matrix system in the FDM is neutrally stable. It means that nothing leaves the system, the energy is preserved.

Another advantage of the simplification above, is that the field points are a function of the initial field only, and the field can be calculated using parallel programming, meaning the field along the surface at different points can be calculated at the same time, rather than being calculated successively, which is more time consuming. Unfortunately this does only work for a flat surface, since for a humped surface the terrain changes the behavior of the field. However, it is useful to calculate the field along a flat surface in order to compare with an accurate analytic model.

3.2.5 Absorption Layer

Since the system is neutrally stable, nothing leaves the system. This means that the waves will not leave the computational domain at the boundaries, they will be reflected instead. At the ground, this is wanted. However, in the sky this is not wanted, and an absorption layer need to be added to the algorithm in order to avoid reflections from the sky. There are two ways to do this; extend the computational domain in the simulations so that the reflections from the sky will not come back within the range of interest, or add an absorption layer to avoid reflections from the sky. The latter one is most interesting, because an absorption layer means a smaller computational domain, therefore less computations and decreased run time.

An absorption layer that works well is the Perfectly Matched Layer (PML) proposed in Bérenger [1993] and the implemented absorption layer is based on this. In the PML the index of refraction is changed in the absorption region by adding an imaginary part so that the wave is damped. Since abrupt change of refractive index numerically creates reflections, [Bérenger, 1993, p.191], the refractive index has to change slowly, starting from zero and increase gradually. The equation used for the absorption layer is equation (3.38), where n is the refractive index and σ_0 is the maximum value of the refractive index in the absorption layer. In Bérenger [1993] Maxwell's equations are solved in time-domain, but in this thesis Maxwell's equations are solved in steady-state, therefore, some adjustments have been done.

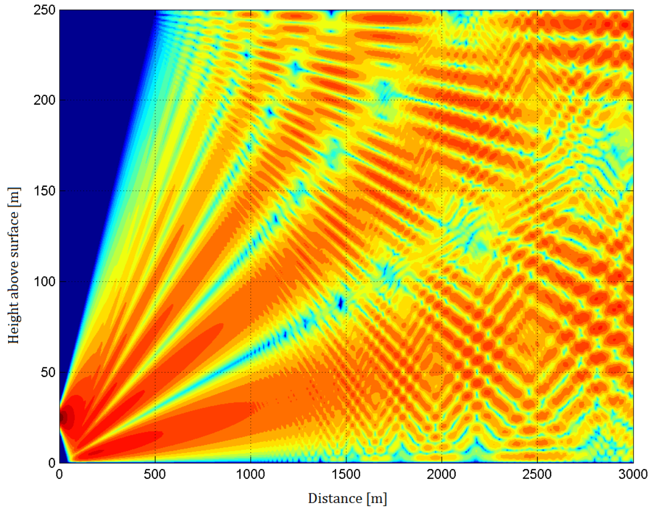
$$\operatorname{Im}\{n\} = \sigma_0 \left(\frac{\text{Point number from beginning of absorption layer}}{\text{Total number of points in the absorption layer}} \right) \tag{3.38}$$

The value of σ_0 is chosen to $\sigma_0 = 0.0015$. σ_0 is limited by an upper and lower boundary. The upper limit is set by the criteria that the waves should not be damped too fast, because it leads to unwanted reflections. The lower limit is set by the criteria that the waves have to be damped fast enough, in order to not be reflected at the boundary of the computational domain, the sky. Reflections from the sky can be seen either by plotting the entire field propagated over a surface, or plotting one height only and compare the path loss with an analytical model. σ_0 was found experimentally, and might not be the optimal value, but it works well.

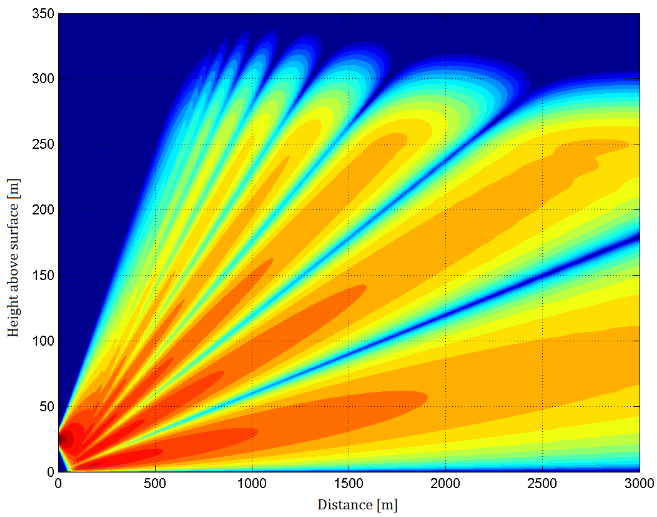
The height of the computational domain and the number of points in the absorption layer can be varied according to the problem. Figure 3.9 shows two examples of the same propagated field, one without and one with absorption layer, figure 3.9a and 3.9b, respectively. The absorption layer clearly damps the wave and reduces the reflections from the top. In figure 3.10 the simulated field is compared with the analytical model for plane earth loss. In figure 3.10a the simulated field starts to oscillate at about 1000 m from the antenna, because of reflections from the top. In this figure, no absorption layer is used. However, in figure 3.10b, an absorption layer is used, and the simulated field does not start to oscillate within the range of 3000 m. This is because the absorption layer damps the waves at the top. In figure 3.9a, 3.9b, 3.10a, and 3.10b the field is propagated over a flat surface. The height of the transmitter antenna is 25 m. In figure 3.10a and 3.10b the field is taken from the same height as the transmitter antenna, 25 m, along the surface.

In figure 3.10a and 3.10b there is a constant difference between the analytical and numerical results. This difference is due to the beam calculation formulas. Equation (2.4) is used for the analytical result and equation (2.5) for the numerical result. They both give the same beam shape. The difference between the equations is that equation (2.4) is a function of the propagation angle, making it suitable for the analytical result. Equation (2.5) is a function of the coordinates, (x, z), making it suitable for numerical applications. Since the difference between the two equations is constant, it does not affect the results, because what is of interest is the slope of the path loss. For accurate results in the terms of the value of the path loss, an appropriate constant can be added. This is also the case for figure 3.12 and the results in chapter 4.

To verify that the absorption layer works, the antenna is "placed" in the air, and nothing will reflect the waves. The verification is performed using the *ParabolicEquation_noGround.m* script in appendix E.1.1. In free-space, the propagating field will only undergo free-space loss. Figure 3.11 shows the simulated field, and figure 3.12 shows the free-space loss along the direction of propagation at the height of the center point of the source. The different height of the curves in figure 3.12

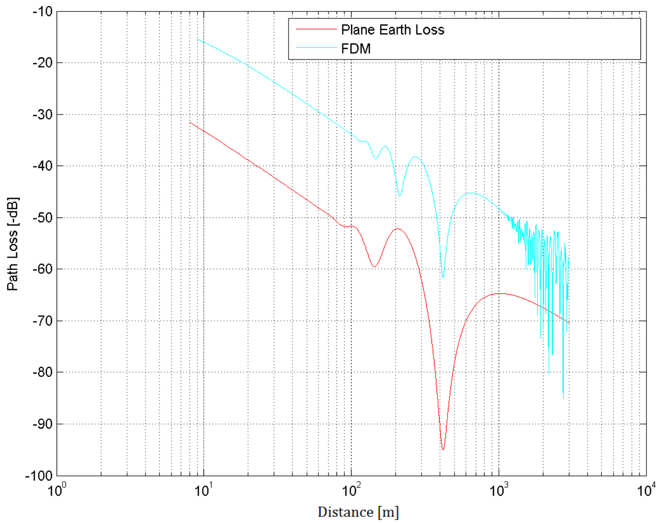


(a) No absorption layer at the top.

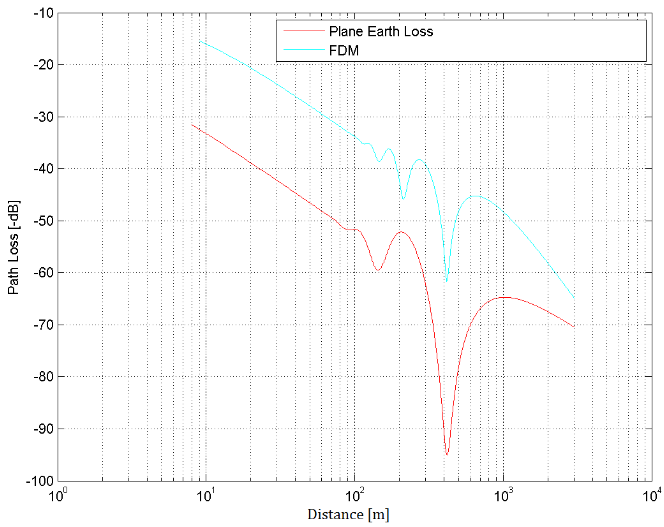


(b) Absorption layer at the top.

Figure 3.9: Field propagated over a flat surface.



(a) No absorption layer at the top.



(b) Absorption layer at the top.

Figure 3.10: Path loss at the height of the transmitter antenna, 25 m, along a flat surface. "FDM" is the simulated path loss. "Plane Earth Loss" is the analytical path loss.

are due to a constant difference and does not matter. What is important is that the two curves have the same slope, meaning that the losses are equal. Figure 3.11 shows that the absorption layer is not perfect, because as the wave propagates, the sphere-like shape of the wave-front has more and more oscillations as the wave-front propagates. Nevertheless, figure 3.12 shows that these oscillations are of less importance. The oscillations may diminish if the absorption layer is thicker, with slower change of refractive index, and/or higher altitude before the absorption layer starts. This will make the computational domain larger, and the runtime of the simulations will increase.

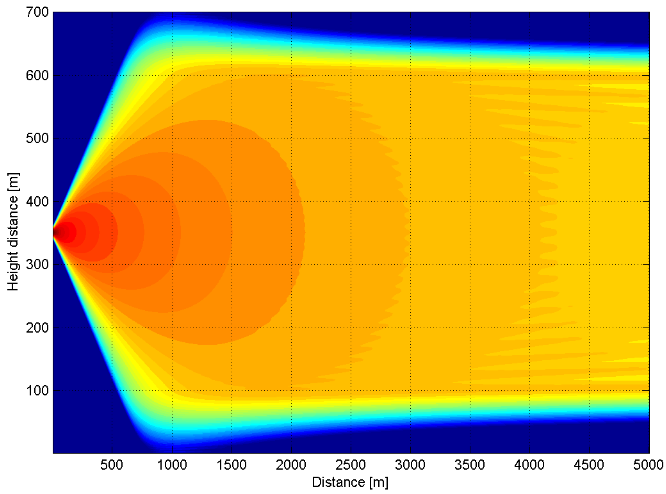


Figure 3.11: Field propagating in free-space with absorption layer at the top and bottom.

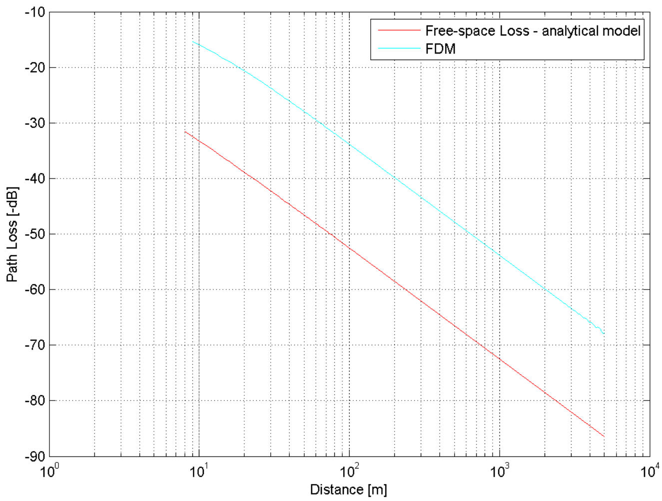


Figure 3.12: Comparison between analytical free-space loss and simulated free-space loss using absorption layer at the top and bottom, same field as in figure 3.11. The comparison is taken at the height of the center of the source. "FDM" is the simulated field. In free-space, path loss is the same as free-space loss.

3.2.6 Non-Flat Surface

One way to model irregularities of a surface is to use the staircase model. In the staircase model, the irregularities in the terrain are modeled as a staircase, meaning that the terrain is flat between two sampling points, see figure 3.13. For ascending

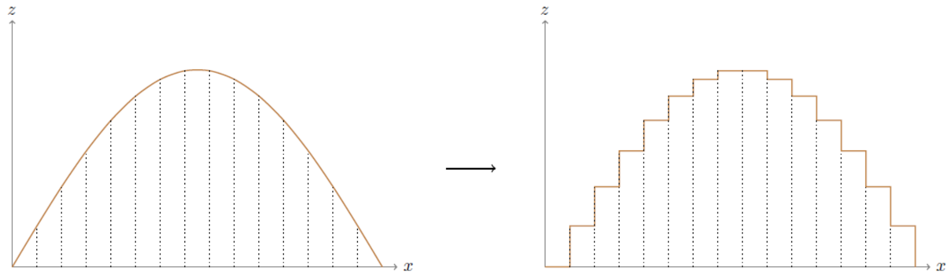


Figure 3.13: Irregular terrain modeled using the staircase model.

terrain the field at range m is calculated from the field at range $m-1$ like if the surface was flat. Then the field at m is truncated to follow the terrain profile, by suppressing the calculated field for points that are below the ground, see figure 3.14. The sharp edges in the "stairs" are disregarded, no corner diffraction is calculated.

For descending terrain, the principle is almost the opposite. The electric field

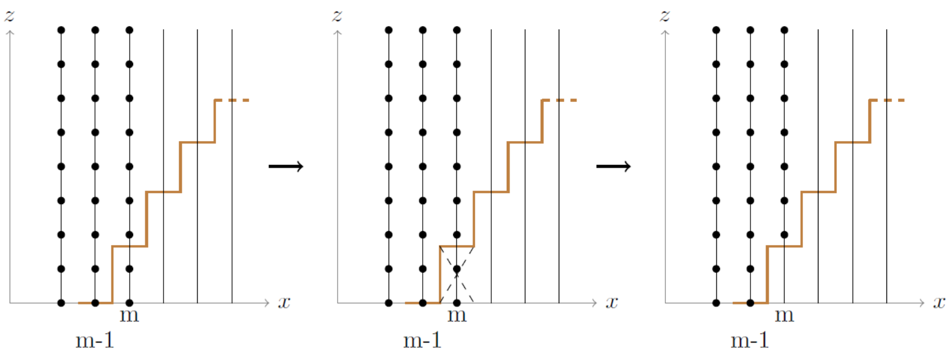


Figure 3.14: Principle of the algorithm for calculation of field in ascending terrain. The dots illustrate the field calculation points.

at range $m-1$ is used to calculate the electric field at range m, pretending that the surface is flat. When this is done, the field closer to the surface at range m is padded with zeros. This method works because for range $m+1$, assuming the terrain is still descending, the field from range m will propagate both upwards and downwards, meaning that at range $m+1$ the field will no longer be zero at at least some of the heights that were padded to zero at range m, see figure 3.15 for illustration of the principle.

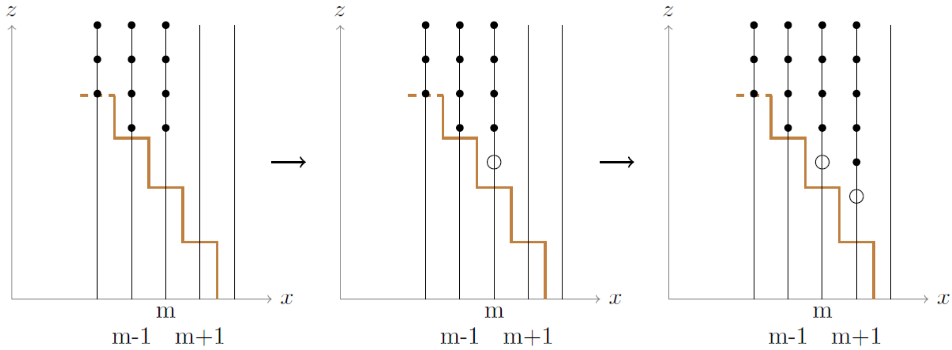


Figure 3.15: Principle of the algorithm for calculation of the electromagnetic field in descending terrain. The dots illustrate the field calculation points, the dots without fill represent the padded dots where the value is zero.

3.3 Verification of the Simulation Results

In order to find out how good the simulation results are, there are different ways of comparing their performance. In this thesis relative field strength and path loss are used.

3.3.1 Relative Field Strength

When comparing the relative field strengths, the variations of the signals within an interval are compared. The field strength values are not of interest. This type of comparison is made when one of the results to compare with cannot be directly compared with the other results. This is the case when the simulated results are compared with an analytical result from Indra.

When there is no analytical result to compare with, relative field strength comparison is used as field strength comparison.

3.3.2 Path Loss

Path loss describes the loss of a signal between the transmitter and receiver, and is given by equation (3.39), [Saunders and Aragón-Zavala, 2007, p. 90]. Path loss is often used as a figure of merit of a radio communication channel.

$$
\begin{aligned}
&L = \frac{P_{TI}}{P_{RI}} \\
&P_{TI}\text{: Effective isotropic transmitter power} \\
&P_{RI}\text{: Effective isotropic receiver power}
\end{aligned} \tag{3.39}
$$

3.3.2.1 Flat Surface

For propagation over a flat surface, the path loss can be calculated analytically. For an isotropic source placed on a perfect reflecting ground using calculations in 3D, the path loss for plane earth is given by equation (3.40), [Saunders and Aragón-Zavala, 2007, p. 99]. P_r is the received power, P_t is the transmitted power, h_b is the height of the transmitter, and h_m is the height of the receiver. Note that equation (3.40) is the inverse of the definition in equation (3.39).

$$
\begin{aligned}
&\frac{P_r}{P_t} = \left(\frac{\lambda}{4\pi d}\right)^2 \left|1 + \exp\left(jk\frac{2h_m h_b}{d}\right)\right|^2 \\
&d\text{: Distance between the transmitter and receiver along the surface} \\
&h_b\text{: Height of the transmitter} \\
&h_m\text{: Height of the receiver}
\end{aligned} \tag{3.40}
$$

For a non-isotropic transmitter, the relations found in section 2.2.2 can be used to find P_r. Then, P_{TI} and T_{RI} can be calculated. In order to find the path loss, equation (3.39) has to be used.

3.3.2.2 Non-Flat Surface

For a non-flat surface, the definition of path loss, equation (3.39), has to be used.

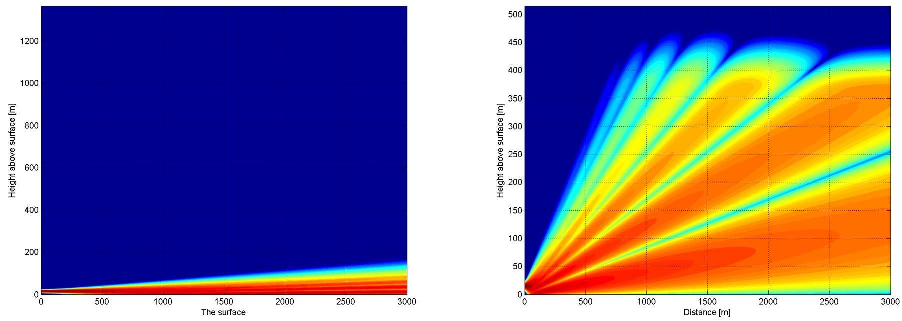
The Hviid et al. [1995] article presents some of their results in path loss. In one test of their algorithm, a simple wedge is used. This makes it possible to compare their results with the results from this thesis, see section 5.3.

Chapter 4

Choice of Parameters

The eligible variables are the step-size in the x- and z-direction, Δx and Δz, respectively. The upper limit is set by the Nyquist's criterion, $\Delta x < \frac{\lambda}{2}$ and $\Delta z < \frac{\lambda}{2}$. At the frequency of the localizer, 110 MHz, $\lambda \simeq 2.73$ m. The lower limit depends on two factors: run time and stability. The run time increases dramatically for small values of Δx and Δz. For the SSA, very small values of Δx and Δz causes instability of the algorithm. For small values of Δx and Δz the distances will be so small that, numerically, there will not be any difference between the current and the previous calculated value. Therefore, the simulated field will propagate "slowly", see figure 4.1a, compared with normal propagation in figure 4.1b. A choice of Δx and Δz that works well, is $\Delta x = \Delta z = 1$ m. Setting Δx and Δz equal can visually be seen as logical, because when the field propagates, it will propagate with the same amount in both the x- and z-direction. The values were chosen by simulating the fields over a flat surface, and then compare the results with the analytical result, "plane earth loss", see figure 4.2, 4.3, and 4.4. The fields are compared at the height of the antenna, 15 m, at each point along the surface. The "correctness" of the algorithms is determined by where the dip of the fields occur, compared with the analytical result. They are supposed to occur at approximately the same distance. The analytical result is obtained using the result from section 2.2.2. Its domain of validity goes from the start to the dip, the dip included. After that this model does not have the correct loss. For explanation of the constant difference between the the analytical and numerical results in figure 4.2, 4.3, and 4.4, see section 3.2.5.

As figure 4.2, 4.3, and 4.4 show, other values of Δx and Δz could as well give



(a) "Slow" propagation, $\Delta x = \Delta z = 0.3$ m. (b) Normal propagation, $\Delta x = \Delta z = 1$ m.

Figure 4.1: Field simulation over a flat surface using SSA. The effect of small different Δx and Δz for the SSA algorithm. Note that the axes are equal in both figures, and that the z-axis in figure 4.1b is the correct one.

the same results. They show that at least at a low height, the FDM algorithm is not affected by various values of Δx and Δz. For simplicity, SSA and FDM use the same parameters.

The test results were obtained using the $ParabolicEquation\_SSA\_FDM.m$ and $ParabolicEquation\_SSA\_FDM\_deltaValueTest.m$ scripts in appendix E.1.2 and E.1.3, respectively. Antenna height is 15 m.

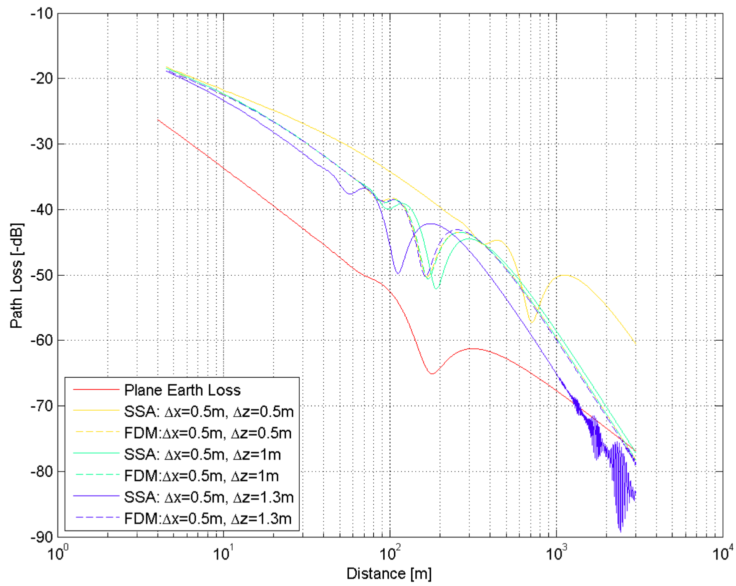


Figure 4.2: Path loss comparison at the antenna height, 15 m, along the surface.

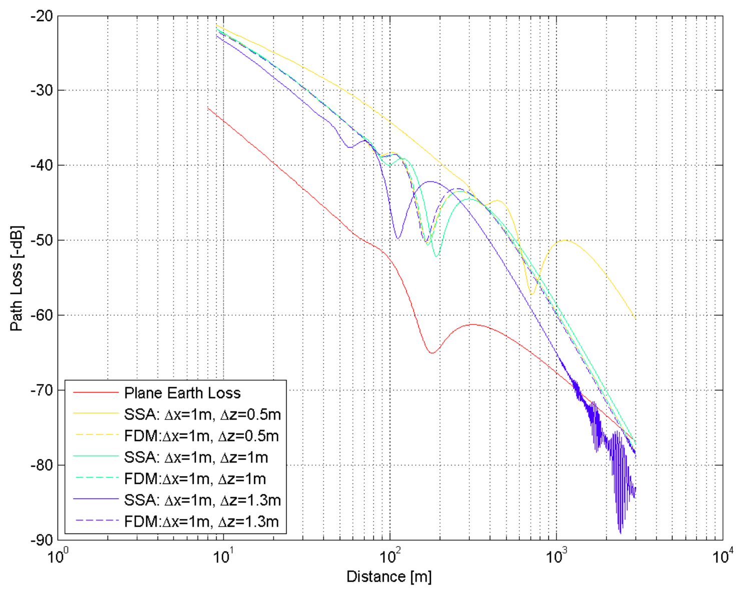


Figure 4.3: Path loss comparison at the antenna height, 15 m, along the surface.

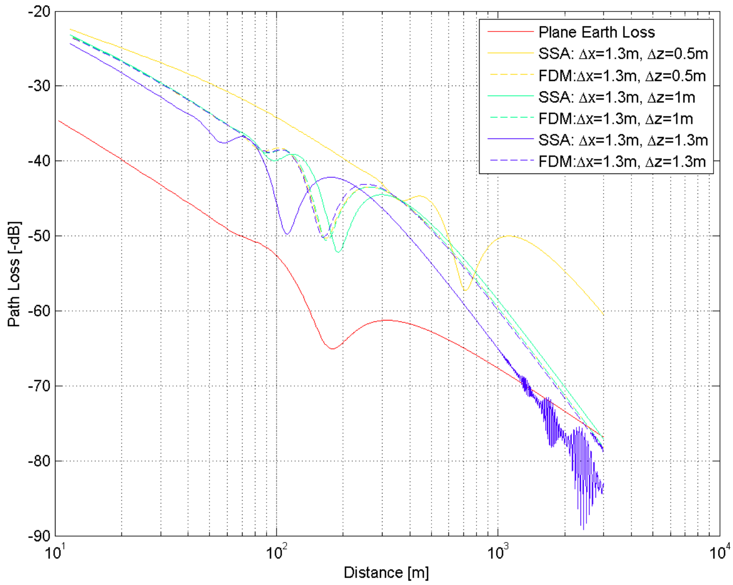


Figure 4.4: Path loss comparison at the antenna height, 15 m, along the surface.

Chapter 5

Results

The results are obtained by simulating the electromagnetic field over different surfaces. The simulation algorithms that are used are the Parabolic Equation methods, the Split-Step Algorithm (SSA) and the Finite-Difference Method (FDM). Due to the number of missing links in the Integral Equation model, this method was not fully implementable, and can therefore not be tested. The goal in this section is to test the performance of the SSA and the FDM over various surface profiles. The simulation surfaces used have the following characteristics:

1. Flat surface
2. Downwards inclined plane
3. Upwards inclined plane
4. Wedge
5. Airport runways: Braunschweig and Luton

The first three cases are of interest because in all these cases, the result can be compared with an analytical result. The fourth case is a wedge used in Hviid et al. [1995]. This is of interest because the simulation results can be compared with the results in Hviid et al. [1995]. The last case, runways of real airports tests the algorithms on real cases. Over the runways there are no results to compare with.

All simulations were run using the frequency of the localizer, 110 MHz, horizontally polarized electric field, and antenna height of 3 m, unless something else is specified. The height of the localizer antenna is usually between 2 to 5 m. Note that in some cases the plot of the simulated field is zoomed in so that it does not cover the entire computational domain. The beamwidth is specified with the half-power beamwidth, either relative to the paraxial direction using the ±-sign, or the total half-power beamwidth, without the ±-sign. Unless anything else is specified, the half-power beam width is 20°, and the beam has no tilt. Half-power beamwidth of 20° means that the maximum beamwidth, which is larger than the half-power beamwidth, will be within 40°. This means that the beam is inside the "validity area". The parabolic equation method described and implemented in this thesis handles preferably a source with a beamwidth of maximum 40°, propagation at ±20° from the paraxial direction. This is due to the error that occurs when approximating the square-root operator in order to obtain the standard parabolic equation, equation (3.23). Referring to figure 3.6 for α, for $\alpha > 20°$ the error will theoretically increase rapidly, see figure 3.7. It is therefore desirable to have a beamwidth of maximum ±20°.

For all plots of simulated fields, the color bar given in figure 5.1 applies.

For all plots of field comparison, the curve labeled "E-field Indra" is the analytical

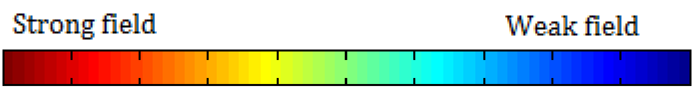


Figure 5.1: Color bar for the field plots.

result. The analytical results may also be referred to as "the results from Indra", because Indra generated the analytical results. Note that in the path loss comparisons, the y-axis of the plot is reversed. This is done for being able to compare the path loss behind a wedge with the results in Hviid et al. [1995].

The code is implemented in Matlab. In order to optimize the speed of the algorithms, they are, as far as possible, implemented on a vectorized form. The implemented scripts and functions can be found in appendix E and in the zip-file[1] attached to the thesis.

[1]The zip-file also includes scripts and functions that are not in use.

5.1 Simulations over a Flat Surface

The simulations were run using the $SSA\_FDM\_indra\_r\_loss.m$ script in appendix E.1.4. This script simulates the electric field over a flat surface of 3000 m. Figure 5.2 and 5.3 show the simulated field using the SSA and FDM algorithm. The half-power beamwidth of the beam is 55°, ±27.5°, in order to have the same width of the beam as the analytical results from Indra. This is outside the "comfort zone" of the algorithm. The gain of the transmitter antenna in the result from Indra is 9dBi. The gain of the Gaussian beam, depends on the beamwidth only, and cannot be changed for a given beamwidth. Therefore, in comparisons with the results from Indra, relative field strengths are used.

In order to test the stability of the FDM, the absolute value of the maximum eigenvalue, $|\lambda_{max}|$, is printed on screen when running the implemented function for FDM on a flat surface, $FDMAbsorptionLayerNumEfficient2$, appendix E.2.1.3. It shows that the FDM is neutrally stable, $|\lambda_{max}| = 1$.

Simulations over a flat surface are of interest because it is most likely necessary

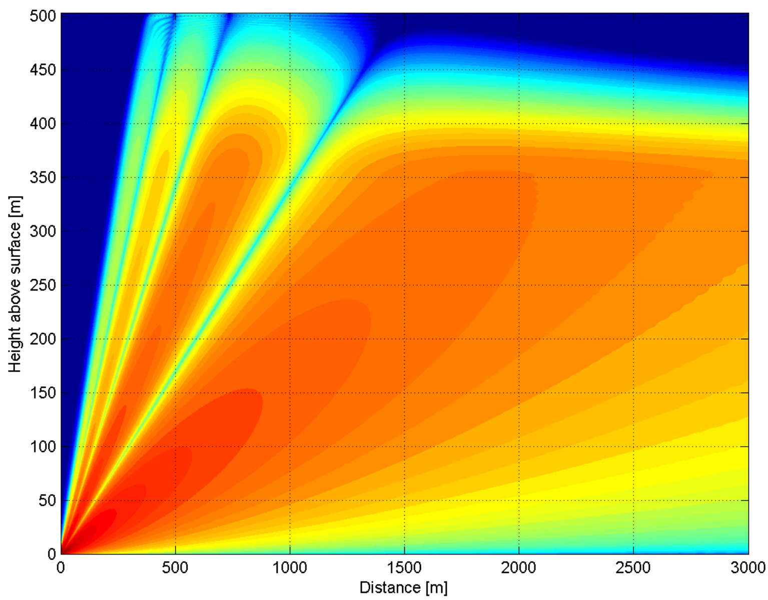


Figure 5.2: Field simulation using the SSA.

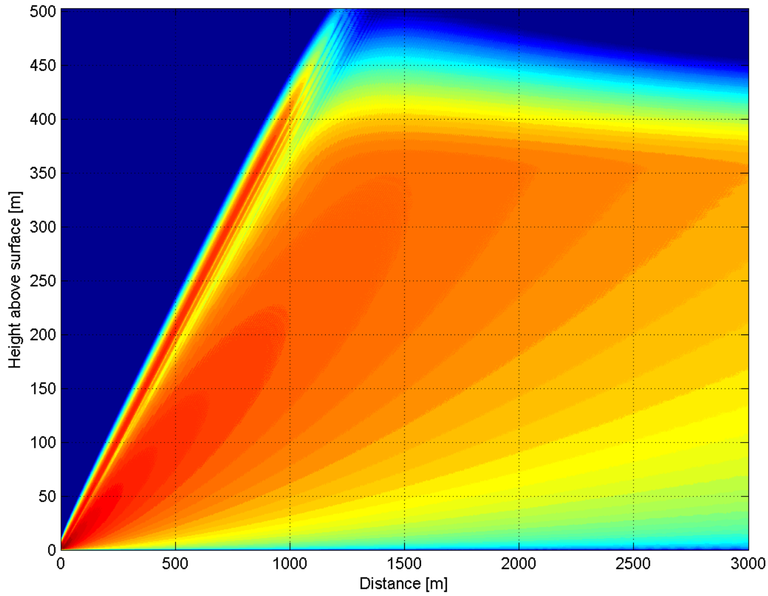


Figure 5.3: Field simulation using the FDM.

that the results are good for a flat surface, in order to good for an irregular surface. This is because the algorithms for flat and irregular surface are similar. However, good results from the flat surface does not mean that the results will be good for an irregular surface. The simulations are performed in 2D. The main difference between 2D and 3D is that the waves are cylindrical and spherical, respectively. This means that in free-space loss a factor of $\frac{1}{r}$ differs between cylindrical and spherical waves. The easiest approach to compare 2D simulation results with 3D simulation results is therefore to add the factor of $\frac{1}{r}$ to the 2D simulation results. This will give an indication of the plane earth loss. However, in the case of plane earth loss, the dips and peaks that will occur due to reflections in the ground, leading to constructive and destructive interference, might not appear at the same spot. This is due to phase differences. Figure 5.4 and figure 5.5 show the simulated field with added $\frac{1}{r}$-loss. These figures show that the shape and appearance of the field is conserved when adding the $\frac{1}{r}$-loss.

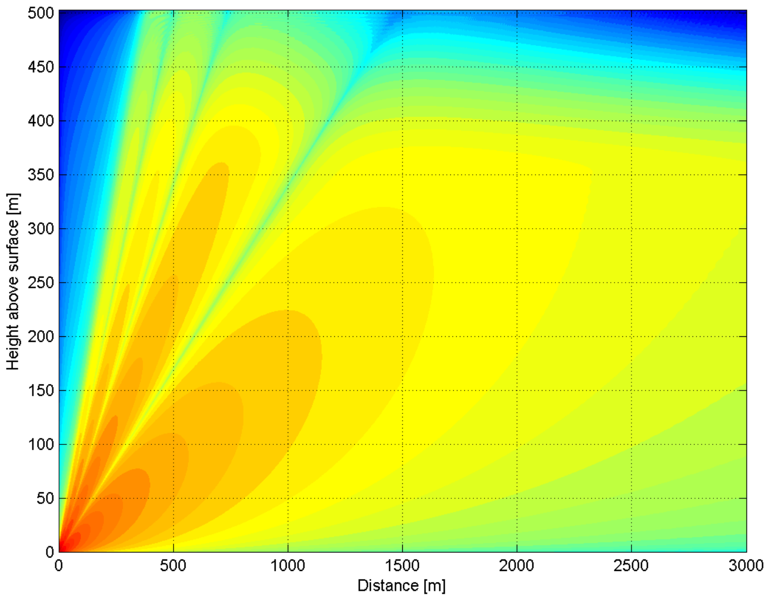


Figure 5.4: Field simulation using the SSA, $\frac{1}{r}$-loss added.

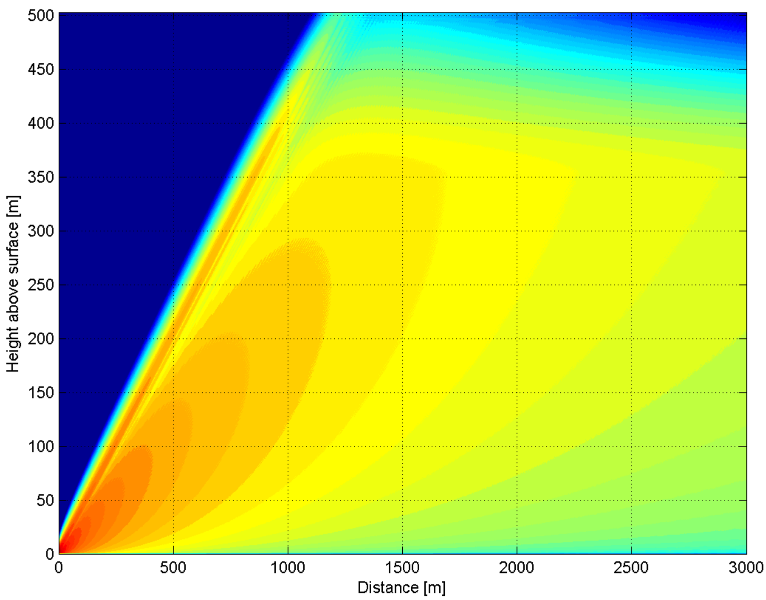


Figure 5.5: Field simulation using the FDM, $\frac{1}{r}$-loss added.

For comparison of the simulated fields, two comparisons are made; horizontal and vertical comparison. In the horizontal comparison the fields are compared at a constant height along the surface. In vertical comparison, the fields are compared at a constant range along the surface, and the height is varying, see illustration in figure 5.6. In addition, comparisons were also made to investigate whether it is necessary to add $\frac{1}{r}$-loss or not.

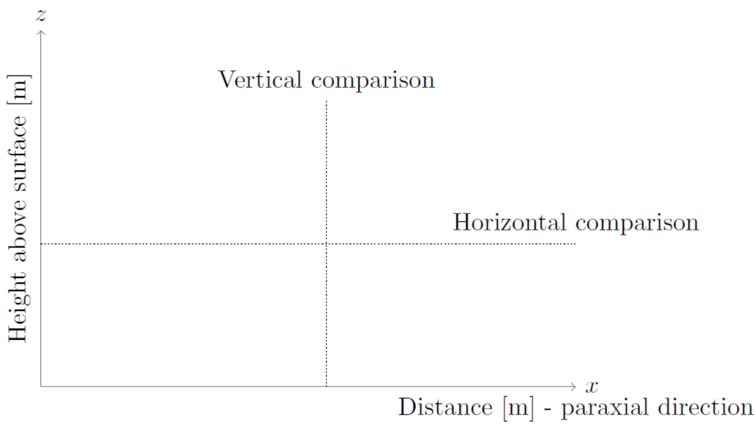


Figure 5.6: Illustration of horizontal and vertical comparison.

5.1.1 Horizontal Comparison - Comparison along the Surface

The horizontal comparisons were made at the antenna height, 3 m above the surface. Two comparisons were made; one without and one with $\frac{1}{r}$-loss added, figure 5.7 and 5.8, respectively. Figure 5.7 shows that the loss of the fields simulated in 2D is less than the analytical result for 3D. However, figure 5.8 shows that adding $\frac{1}{r}$-loss is too much. This is because the transmitted wave is directive and not isotropic. The wave will therefore not spread in a sphere, leading to less free-space loss than a sphere, and therefore less path-loss. The $\frac{1}{r}$-loss-approximation is too simple.

The $\frac{1}{r}$ approximation does have one more simplification that adds inaccuracy to the results; the total field at the receiver is the sum of the direct and the ground-reflected wave. These two waves will have traveled different distances. When adding the $\frac{1}{r}$-loss, this is not taken into account. Since $\frac{1}{r}$ is not constant along the surface, some inaccuracy is added.

Figure 5.7 shows that the field strength dip that occurs at the beginning of the surface does not occur at the exact same distance for SSA and FDM, and not at the same distance as the analytical result either. It would be desirable if they did, but the field of interest is at larger distances, at the end of the runways. Except for different slope, the SSA and FDM results have the same shape as the analytical one.

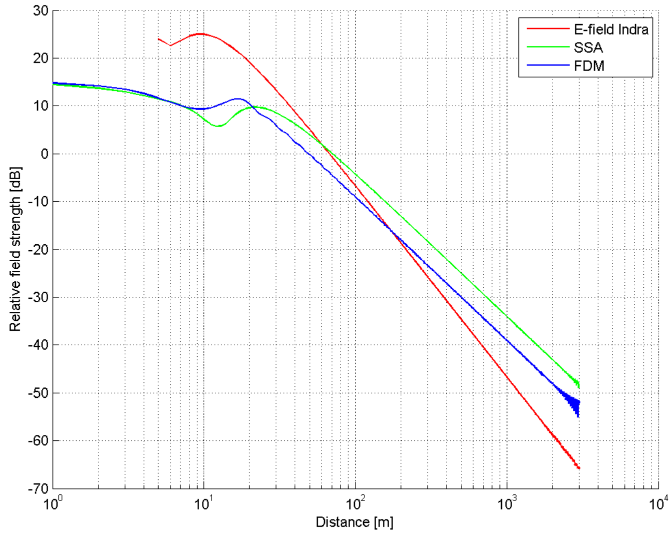


Figure 5.7: Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. No additional loss.

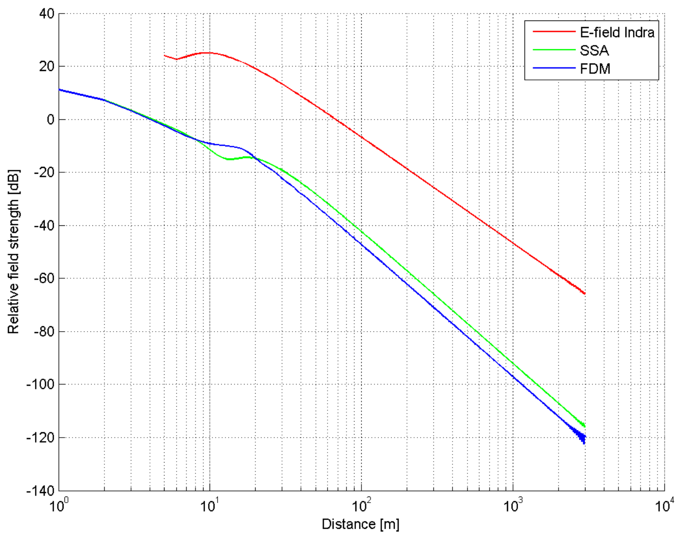


Figure 5.8: Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. $\frac{1}{r}$-loss added.

5.1.2 Vertical Comparison - Comparison in the Height Direction

The vertical comparisons were made at the constant range of 1000 m along the surface. Firstly, one comparison was made; comparing the results with and without additional $\frac{1}{r}$-loss with the analytical result, figure 5.9. The figure shows that there is hardly any visible difference between the results with and without additional $\frac{1}{r}$-loss. This is because when comparing in the vertical direction, there is not much variation in the distance difference between the source and the different points along the vertical. Therefore, the additional loss for spherical waves will be close to constant in the vertical direction. Since the $\frac{1}{r}$-loss is close to constant because of small differences, any additional inaccuracy will also be very small. It is therefore no need to add $\frac{1}{r}$-loss for vertical comparisons. Figure 5.9 shows that the FDM results overlap with the analytical result. The SSA results do not overlap the analytical result, but they have similar shape.

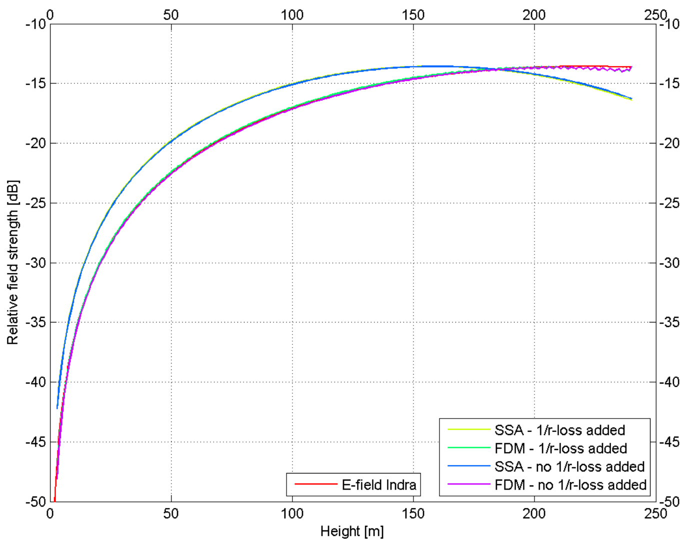


Figure 5.9: Relative field strength at the distance of 1000 m along a flat surface. Both SSA algorithms overlap each other, and both FDM algorithms overlap each other.

Figure 5.11 and 5.10 show a zoomed version of figure 5.9, the maximum height is 50 m. In figure 5.10 the field strengths are aligned to have the same value at the lowest point of consideration. This can be done because the plot shows relative field strengths. Figure 5.10 shows that at low heights, the FDM increases faster, and the SSA slower than the analytical result. As shows both figure 5.10 and 5.11, all of the results have approximately the same slope from 15 m up to 50 m. In figure 5.11 all relative fields strengths are shifted to have the same maximum value.

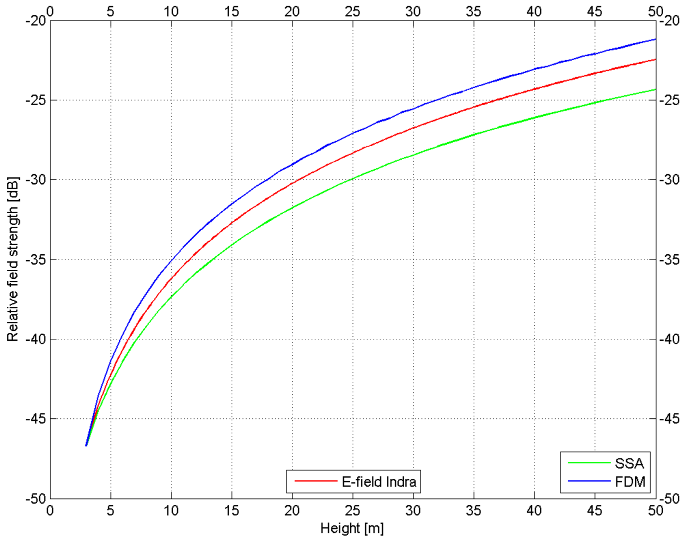


Figure 5.10: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. No $\frac{1}{r}$-loss added. Aligned at the lowest height.

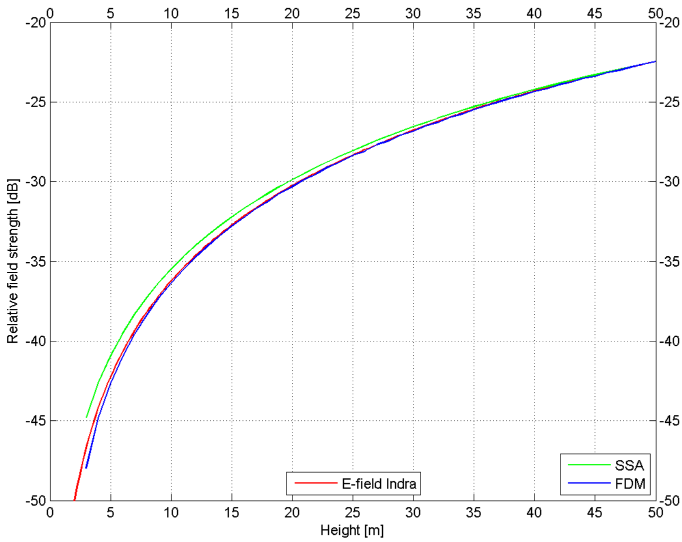


Figure 5.11: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. No $\frac{1}{r}$-loss added. Aligned at the maximum value.

5.1.3 Flat Surface Summary

When comparing in the vertical direction there is no difference between 2D and 3D. For horizontal comparison, 2D and 3D results cannot be compared with each other because of different loss. $\frac{1}{r}$-loss adds inaccuracy and is not the correct loss when the transmitted field is directive. It is not tested whether it works for an isotropic source or not, because the algorithm cannot handle wide-angle propagation. Therefore, for vertical comparisons, 2D results can be compared with 3D results. For horizontal comparison, 2D results should be compared with 2D results.

The results show that in vertical comparison the FDM has the same shape as the analytical result. The SSA have similar shape as the analytical result. In horizontal comparison, the field strength of SSA and FDM have decreases linearly in dB from the distance of approximately 30 m and up. In this region the analytical result also decreases linearly. This shows that the Parabolic Equation methods works for a flat surface.

5.2 Inclined Plane

An upwards and downward inclined plane can be used to test the algorithms and still being able to compare the result with the analytical result for a flat surface. This can be done by steering the transmitted field parallel to the inclined plane. For vertical comparison, the vertical direction is the direction perpendicular to the plane. Figure 5.12 illustrates the principle for a downwards inclined plane. The principle is the same for an upwards inclined plane.

A runway will hardly never have larger height difference than 20 m. Therefore, the height difference on the inclined plane will be 20 m. Any difference between the simulated fields over a flat surface and the inclined plane would most likely be due to quantization error. The resolution has to be the same as Δx and Δz in both the x- and z-direction, and is therefore 1 m. This means that the planes are implemented as stairs, where the height difference of each stair step is 1 m. The half-power beam width used is 55°, in order to be able to compare with the analytical result from Indra.

The black line near the end of the simulated fields show the "vertical" direction for the inclined planes, perpendicular to the plane. The line shows where the field values are taken from in the vertical comparison.

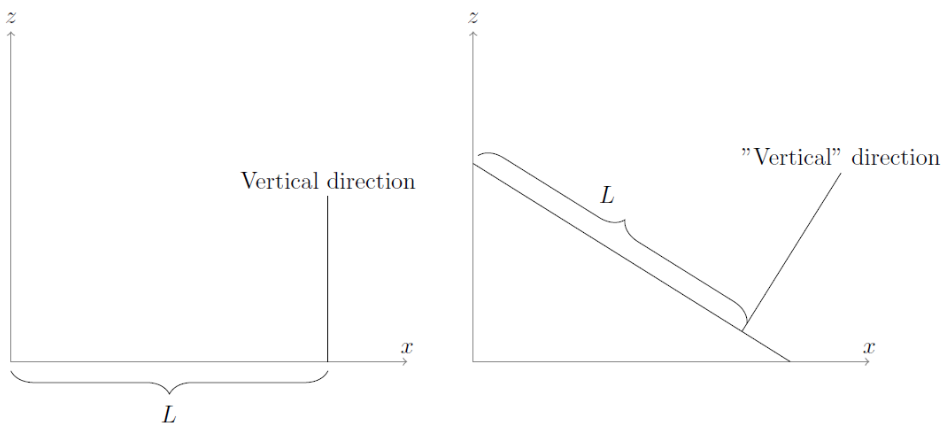


Figure 5.12: Flat surface and downwards inclined plane, field comparison in the vertical direction. The distance between transmitter and the line-of-comparison along the surface is L. The transmitter is assumed to be along the z-axis at the same height relative to the ground.

5.2.1 Downwards Inclined Plane

The simulations were run using the *DownwardsInclinedPlane.m* script in appendix E.1.5. Figure 5.19 and 5.20 show the simulated field over a downwards inclined plane, using the SSA and FDM, respectively.

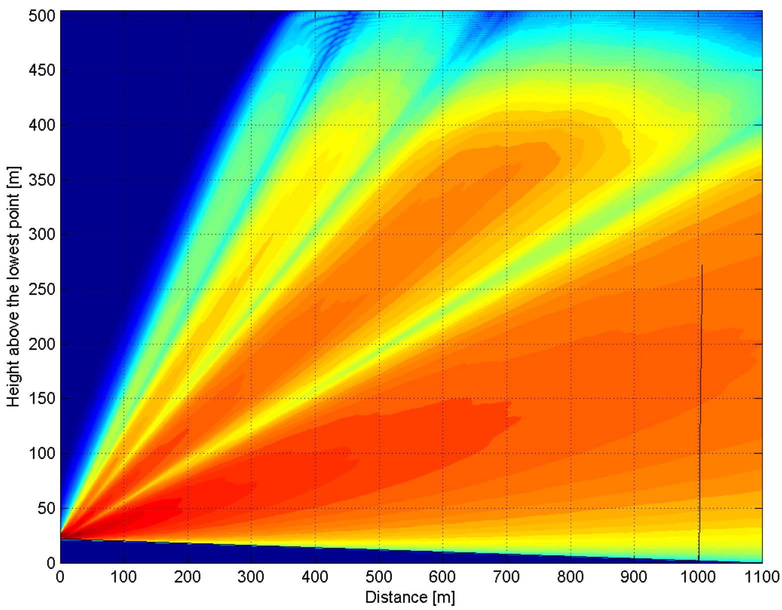


Figure 5.13: Field simulation using the SSA on a downwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

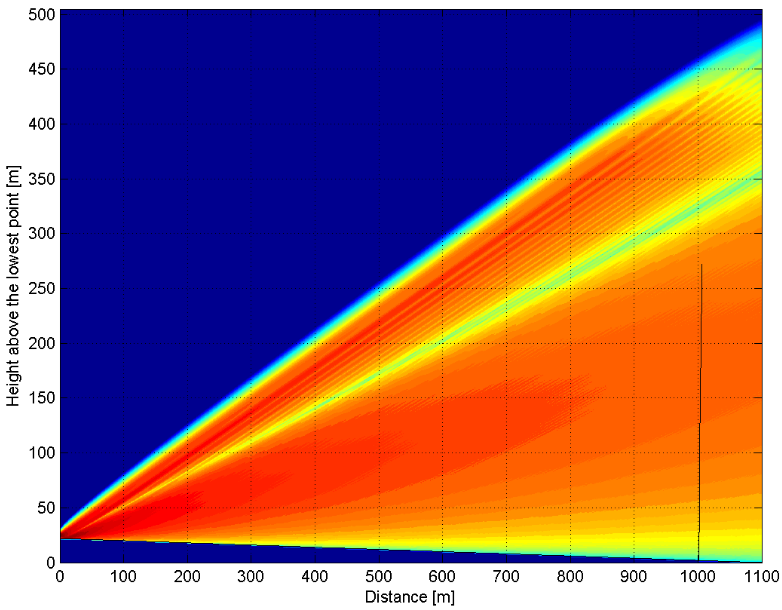


Figure 5.14: Field simulation using the FDM on a downwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

Figure 5.15 shows a vertical comparison of relative field strengths between the simulated fields on the inclined plane, the simulated fields on a flat surface, and the analytical result. The results are shifted to have the same maximum value as the analytical result. The shape of the results is preserved. The figure shows that the SSA and FDM from the inclined plane have the same shape. Their shape is something between the shape of the SSA and FDM for a flat surface.

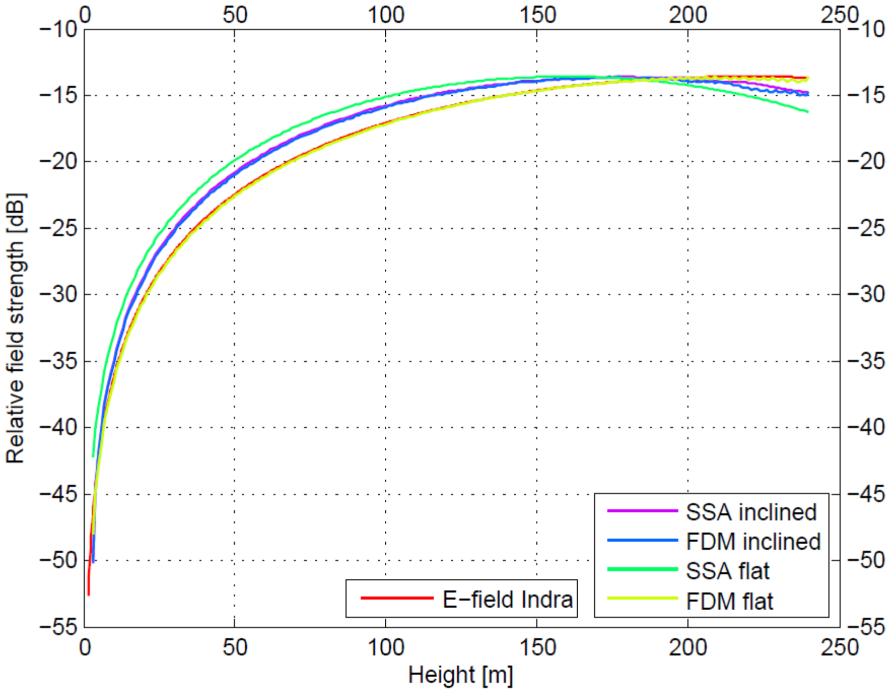


Figure 5.15: Vertical comparison of the relative field strength at the distance of 1000 m along a downwards inclined plane.

Figure 5.16 and 5.17 are zoomed versions of figure 5.15. In figure 5.16 the field strengths are shifted to have the same minimum value as the analytical result, and in figure 5.17 they are shifted to have the same maximum as the analytical result. Figure 5.16 shows that the field strength of the fields simulated along the inclined plane increases faster than the analytical result and the simulated results over a flat surface, for lower heights. This may be because of the staircase model for downwards propagation, with zero-padding for downwards step. Due to the staircase model, not all points will be within line-of-sight from the transmitter. Both figure 5.16 and 5.17 show that the inclined results differ from the flat results up to somewhere between 5 and 10 m. 5.17 show that the field strengths have approximately the same slope from somewhere between 10 and 15 m up to 50 m. The SSA and FDM for the inclined plane follow each other closely in the entire domain.

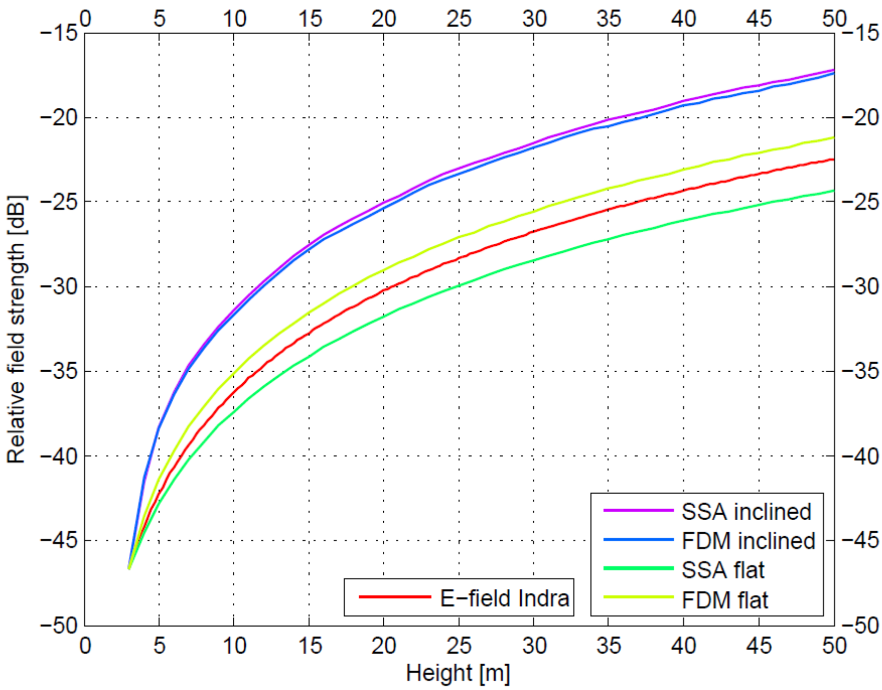


Figure 5.16: Vertical comparison of the relative field strength at the distance of 1000 m along a downwards inclined plane. The relative field strengths are aligned to the minimum value of the analytic field.

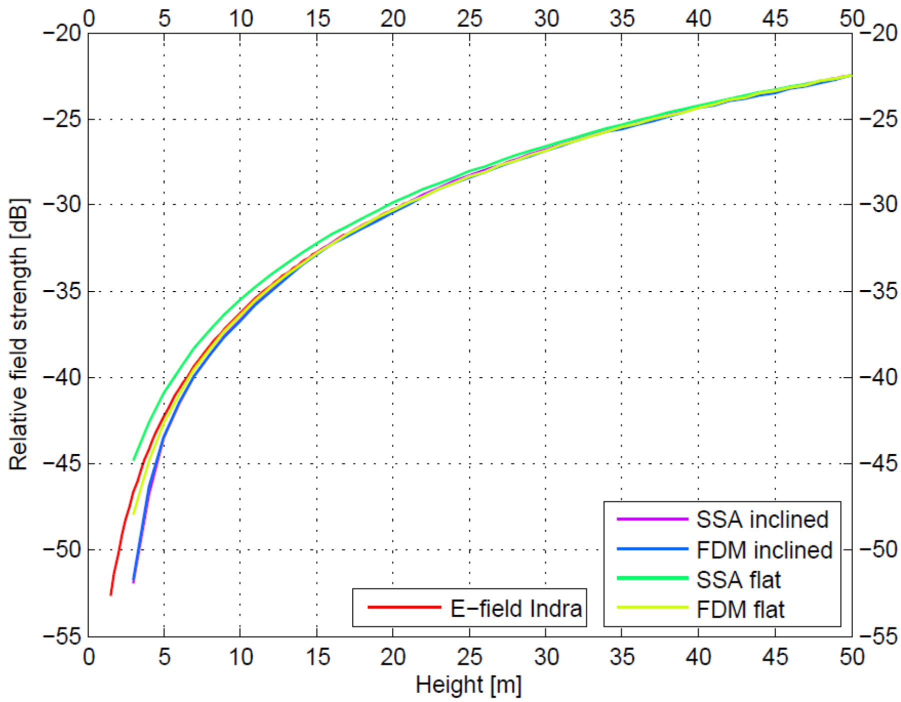
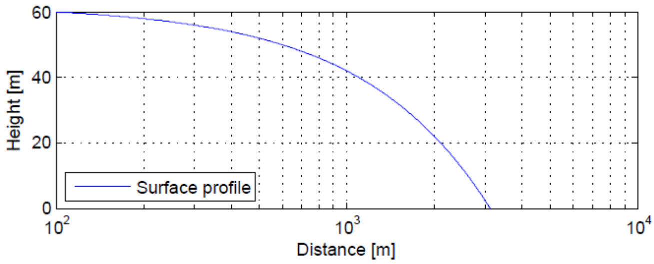
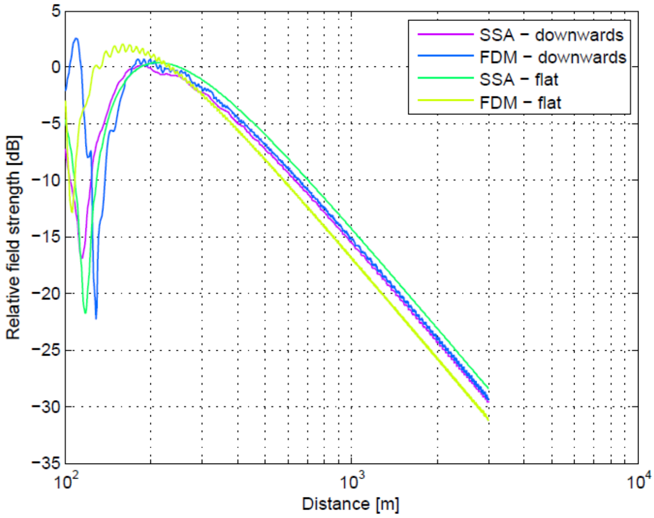


Figure 5.17: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. The relative field strengths are aligned to the maximum value of the analytic field.



(a) Surface profile of the downwards inclined plane.



(b) Horizontal comparison at 15 m above the surface at each point.

Figure 5.18: Horizontal comparison between the field over the runway and over a flat surface. Figure 5.18a is for surface reference.

Figure 5.18b shows horizontal comparison at the height of 15 m above the surface, along the inclined plane, between the results from the inclined plane and the results from a flat surface. The results are not shifted, so the comparison is the same as field strength comparison. For shorter distances, up to 200 m, the simulated results differ quite a bit. From 200 m and up, all the results follow each other. In this region the SSA and FDM from the inclined plane almost overlap each other, and their values are somewhere between the values of the SSA and FDM for a flat surface.

5.2.2 Upwards Inclined Plane

The simulations were run using the $UpwardsInclinedPlane2.m$ script in appendix E.1.6. Figure 5.19 and 5.20 show the simulated field over an upwards inclined plane, using the SSA and FDM. The black line near the end of the simulated field show the "vertical" direction for the inclined plane, perpendicular to the plane.

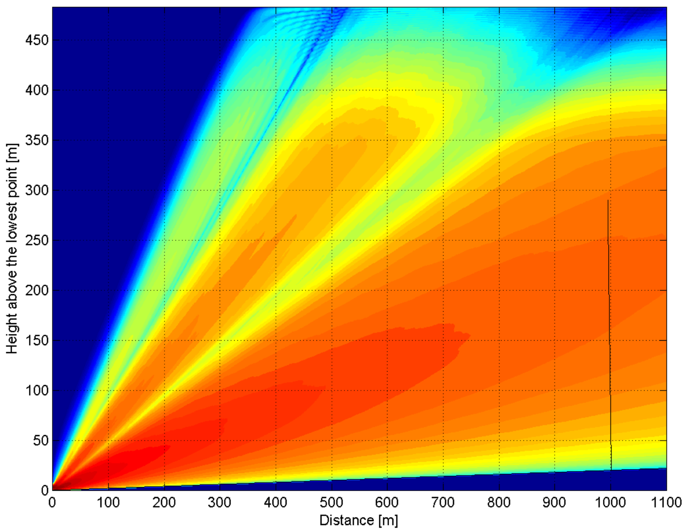


Figure 5.19: Field simulation using the SSA on an upwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

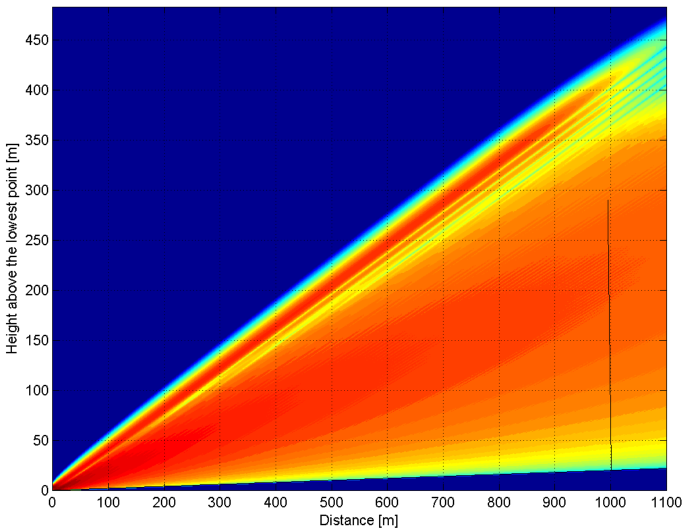


Figure 5.20: Field simulation using the FDM on an upwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

Figure 5.21 shows a vertical comparison between the simulated results over the inclined plane and a flat surface, and the analytical result. The simulated results are shifted to have the same maximum value as the analytical result. The shapes are preserved. The figure shows that the results from the inclined plane have very similar shape, and their shape is something between the shape of the SSA and FDM results from a flat surface.

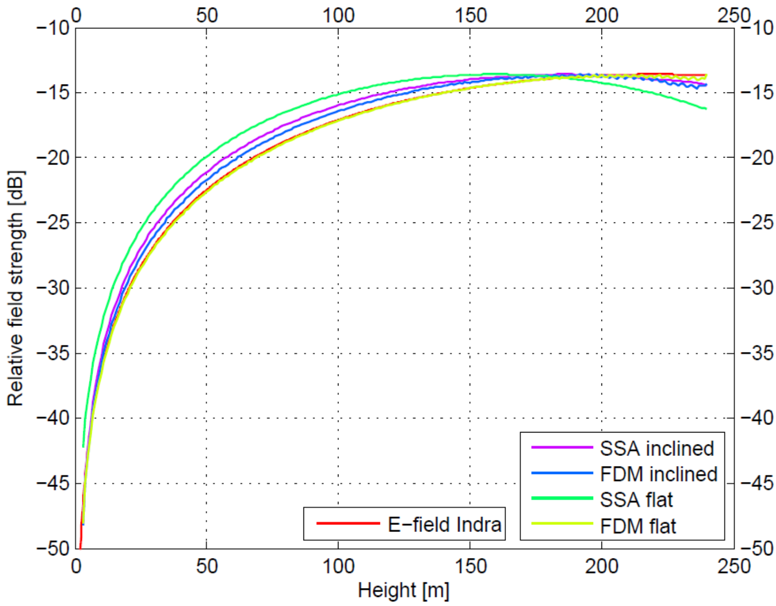


Figure 5.21: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane.

Figure 5.22 and 5.23 are a zoomed versions of figure 5.21. In figure 5.22 the field strengths are shifted to have the same value as the analytical result at the lowest point of consideration. The figure shows that the results from the inclined plane increase faster than than the other results at lower heights. In figure 5.23 all relative fields strengths are shifted to have the same maximum value as the analytical result, at the highest point of consideration. As shows figure 5.23, all the results have approximately the same slope from somewhere between 10 and 15 m and up to 50 m, this is also the same as for flat surface. The SSA and FDM results from the inclined plane follow each other closely in the entire domain. The reason that the results from the inclined plane increases fastest at lower heights, may be due to the algorithm for handling undulating surface.

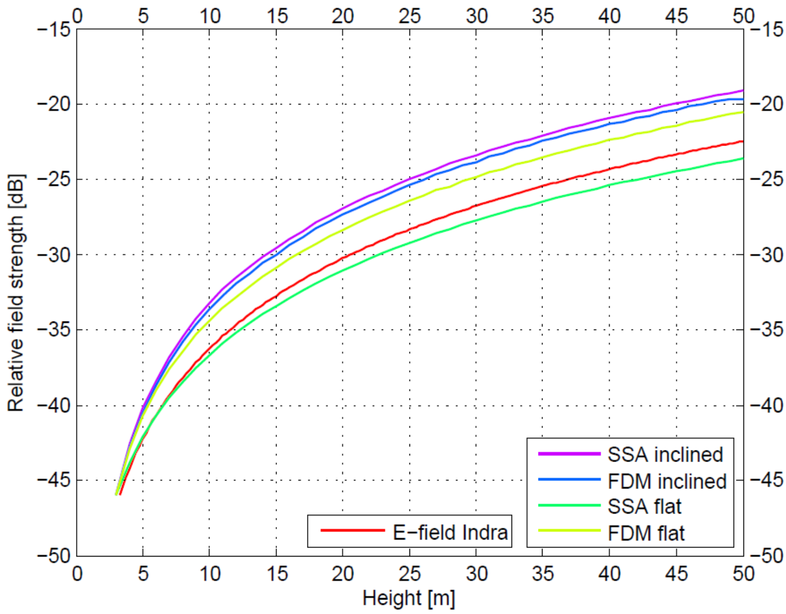


Figure 5.22: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane. The relative field strengths are aligned to the minimum value of the analytic field.

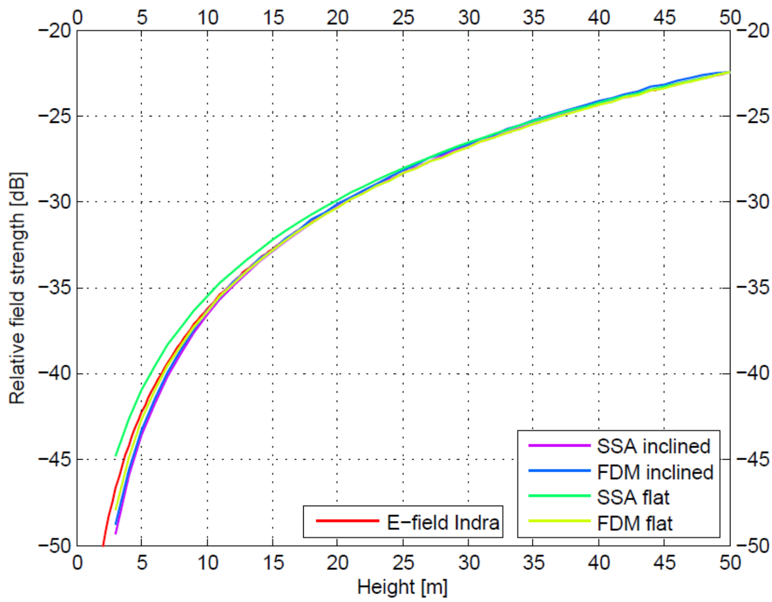
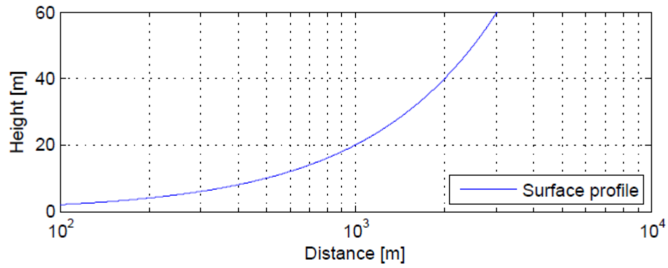
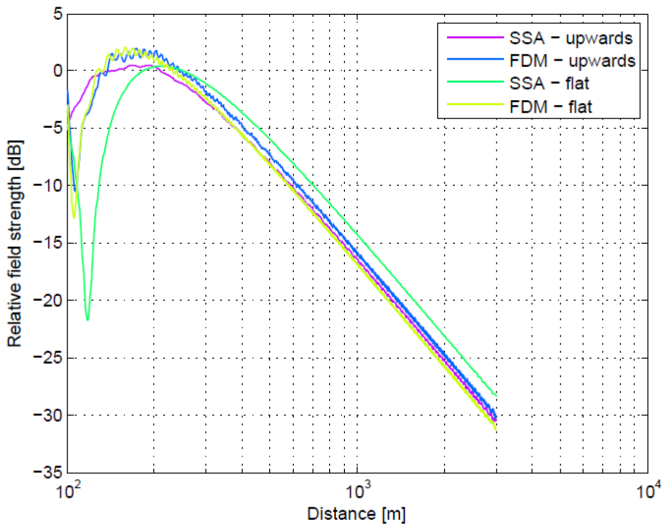


Figure 5.23: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane. The relative field strengths are aligned to the maximum value of the analytic field.



(a) Surface profile of the upwards inclined plane.



(b) Horizontal comparison at 15 m above the surface at each point.

Figure 5.24: Horizontal comparison between the field over the upwards inclined plane and over a flat surface. Figure 5.24a is for surface reference.

Figure 5.24b shows horizontal comparison at the height of 15 m above the surface, along the inclined plane, between the results from the inclined plane and the results from a flat surface. The results are not shifted, so the comparison is the same as field strength comparison. For shorter distances, up to 200 m, all the simulated results differ quite a bit. From 200 m and up, all the results follow each other. In this region the SSA and FDM from the inclined plane almost overlap each other, and their values are somewhere between the values of the SSA and FDM for a flat surface.

5.2.3 Inclined Surface Summary

Both for vertical and horizontal comparison, the results for SSA and FDM over an inclined plane lie in between the SSA and FDM for a flat surface. For vertical comparison, the shape of the SSA and FDM for inclined surface is something between the SSA and FDM for flat surface, except at lower heights, up to approximately 10 m, where the results from the inclined surface decrease faster. This difference may be due to the algorithm for irregular surface. For horizontal comparison at shorter distances, up to approximately 200 m, all results, both from flat and inclined plane, differ. At larger distances, from 200 m and up, the results from the inclined plane are something between the SSA and FDM for a flat surface. An aircraft landing on a runway will receive the signals that have traversed the runway, within the larger distances from the localizer. Therefore, algorithm performance on larger distances most relevant for this thesis.

The results for inclined plane show that the SSA and FDM can handle both up- and downwards surfaces. The SSA and FDM result for the up- and downwards planes are more similar than the simulation results over a flat surface.

5.3 Simulations over a Wedge

The simulations were run using the $WedgeComparison\_Hviid.m$ script in appendix E.1.7. The wedge in question is taken from Hviid et al. [1995], and is given i figure 5.25. The frequency used is 100 MHz, for being able to compare with the results in Hviid et al. [1995].

Figure 5.26 and 5.27 show a zoomed version of the simulated fields over the

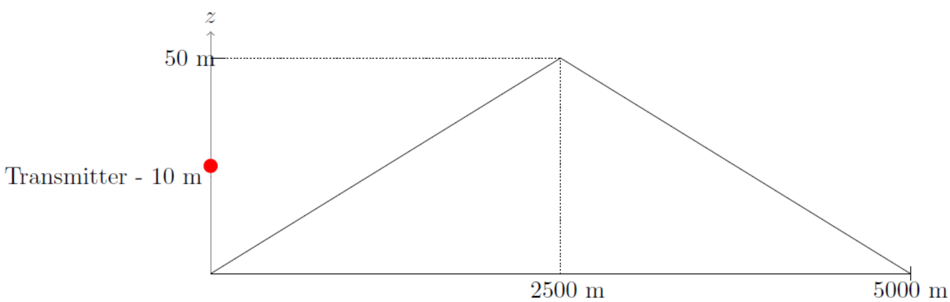


Figure 5.25: The setup for the wedge.

wedge, using the SSA and FDM, respectively. The computational domain had to be very large in order to avoid spurious numerical effects. The total simulated fields can be found in figure C.1 and C.2 in appendix C.

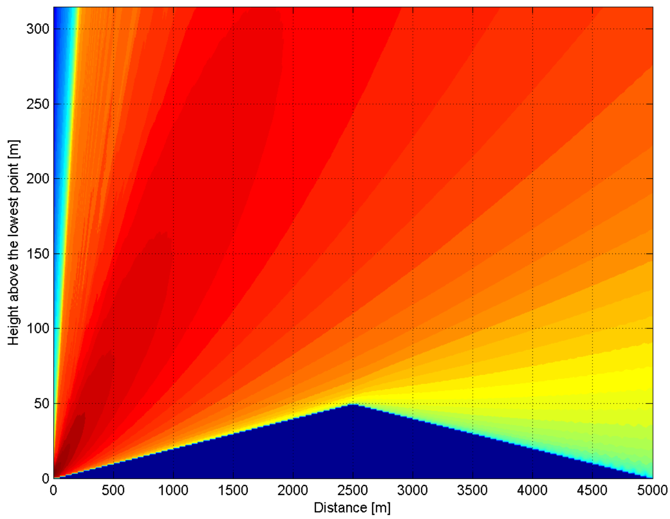


Figure 5.26: Field simulation over the wedge given in figure 5.25 using the SSA. Frequency: 100 MHz.

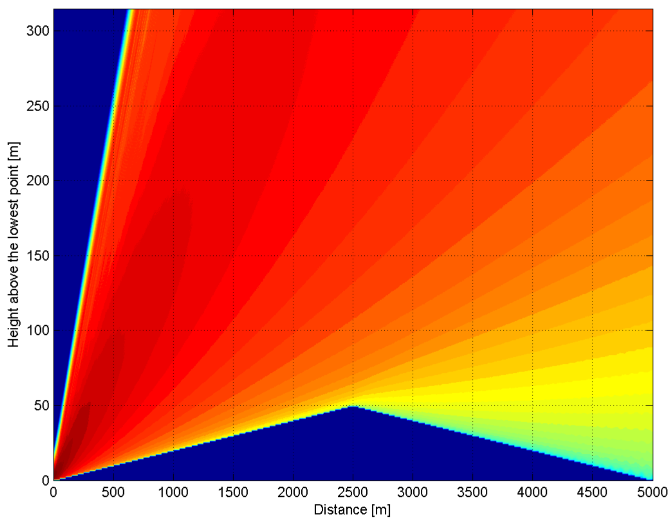
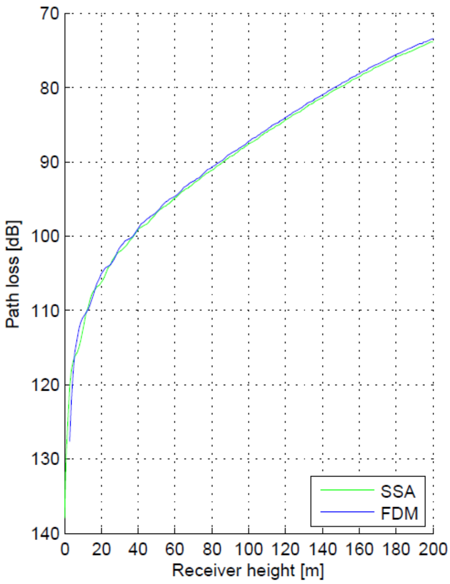
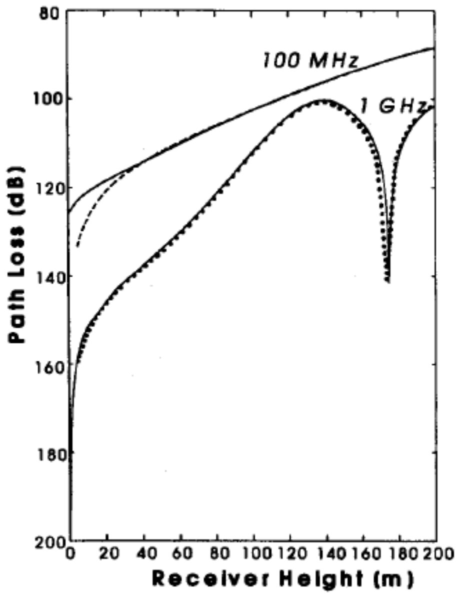


Figure 5.27: Field simulation over the wedge given in figure 5.25 using the FDM. Frequency: 100 MHz.

Figure 5.28 shows vertical comparison of the path losses at the distance of 5000 m. Figure 5.28a shows the path losses from the simulated fields, and figure 5.28b shows the path losses using the Uniform Theory of Diffraction (UTD) and the Integral Equation Model from Hviid et al. [1995]. The simulations were run at 100 MHz, so that is the frequency of comparison. Figure 5.28a shows that the path losses for the SSA and FDM are approximately equal. Figure 5.29 shows that the path loss results of the wedge and the flat surface will approach the same value, meaning that these results are consistent. For lower heights, up to 20 m, there is a large difference between the simulated path losses in figure 5.28a and 5.28b. However, from approximately 20 m up to 200 m, the path losses in both figures are linear in dB. In this interval, the SSA and FDM path losses have approximately the same slope as Hviid et al. [1995]. The path loss decreases with approximately 30 dB in this interval. The values of the path loss in figure 5.28a and 5.28b are different. The "transmitter antenna" in Hviid et al. [1995] is a dipole, however, which kind of dipole is not specified. This means that the beam shape remains unknown. The "transmitter antenna" gain is also unknown. As shows the equation for path loss, equation (3.39), path loss is independent of the type of antennas used. In the region of altitude from 20 to 200 m, the path loss value differences between the simulated fields and Hviid et al. [1995] may be due to constant difference, leading a constant difference.



(a) Path loss of simulated fields; SSA and FDM



(b) Results from Hviid et al. [1995]; UTD (dashed line) and Integral Equation Model (continuous line).

Figure 5.28: Vertical comparison of path loss at the distance of 5000 m, for field propagated over the wedge in figure 5.25.

Figure 5.29 compares the path loss behind the wedge, at the distance of 5000 m, between the path loss over a flat surface. This figure shows that the path loss behind the wedge is larger than along a flat surface, which is as expected because there is no line-of-sight from the transmitter. The wedge will influence the waves above its height, as long as the wedge is within the first Fresnel zone of the receiving point. With a wedge height of 50 m and antenna height of 3 m, the lowest point with line-of-sight from the transmitter, at the distance of 5000 m, is at 96 m, see equation (5.1). This means that the wedge affects the signal high up in the air, higher than 96 m. As already mentioned, figure 5.29 also shows that as the altitude gets higher, the signals from the wedge and the flat surface approaches the same path loss value. This is as expected because the influence of the wedge on the signal will gradually decrease.

$$\begin{aligned} h_{\text{Line-of-sight}} &= 5000\ m \cdot \tan\left(\frac{47\ m}{2500\ m}\right) + 3\ m \\ &= 96\ m \end{aligned} \tag{5.1}$$

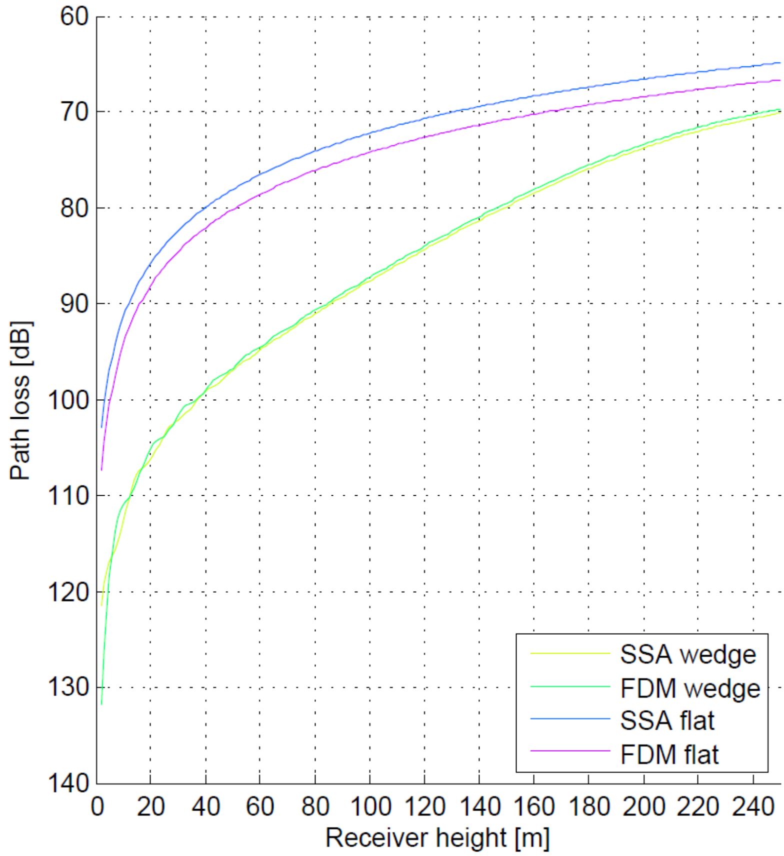
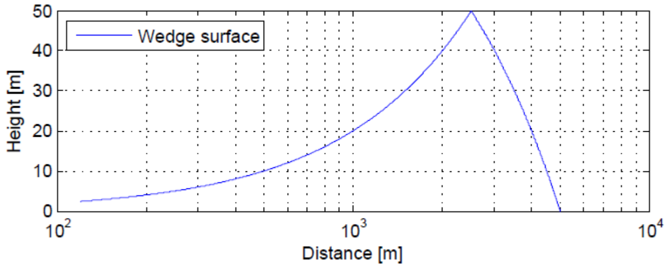
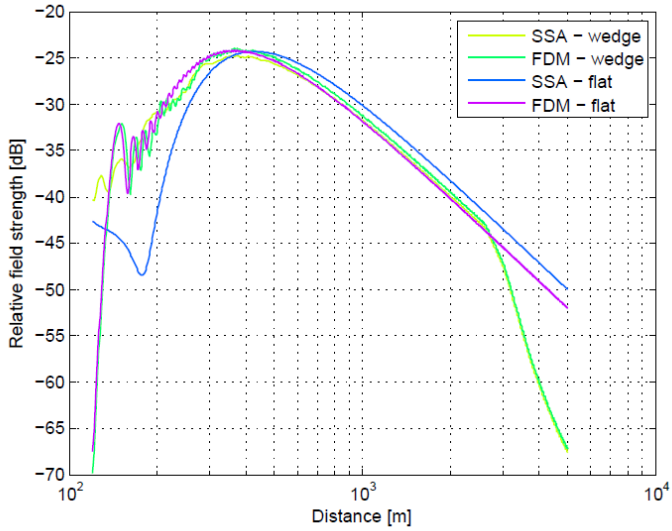


Figure 5.29: Vertical comparison of path loss of the field propagated along a flat surface and behind the wedge, at 5000 m.



(a) Wedge profile



(b) Horizontal comparison at 15 m above the surface at each point.

Figure 5.30: Horizontal comparison between the field over the wedge and over a flat surface. Figure 5.30a is for the wedge surface reference.

Figure 5.30b shows horizontal comparison, at 15 m above the surface, following the surface, between the the fields simulated over the wedge and a flat surface. The results are not shifted, so the comparison is the same as field strength comparison. The figure show that from approximately 300 to 2500 m, all the results follow each other, with almost the same field strength value and slope. The SSA and FDM from the wedge simulations almost overlap each other from the distance of 300 m, and their values are somewhere between the SSA and FDM values from the flat surface. The figure shows a drastical decrease of field strength slope behind the wedge for the SSA and FDM over the wedge. This is as expected because it is within the shadow region of the transmitter, there is not line-of sight from the transmitter. Figure 5.30a is for surface reference.

5.3.1 Wedge Summary

The path loss comparisons behind the wedge, between the simulated results and the results from [Hviid et al., 1995], show that both the SSA and FDM can handle a wedge. The statement is supported by the the horizontal comparisons which show that the field strength decreases faster behind the wedge than within line-of-sight from the transmitter.

5.4 Simulation over Runways

Fields are simulated over two runways, Braunschweig and Luton. The both have a smooth irregular terrain characteristic. Vertical and horizontal comparisons are made. The vertical comparisons takes place at the end of the runways. The horizontal comparisons takes place at 15 m above the surface, follows the terrain. The choice of 15 m above the surface, is made in order to follow the surface, and at the same time avoid numerical effects due to ondulating terrain. The interest of both comparisons is to see how the terrain affects the signals. Indra does not have any measurements form the airports that are suitable for comparisons. Therefore, the comparisons are of simulated fields only.

5.4.1 The Braunschweig Airport Runway

The simulations over the Braunschweig airport runway were run using the *Braunschweig.m* script in appendix E.1.8. The surface profile of the Braunschweig airport is given in figure 5.31. The calculated fields using the SSA and FDM methods are given in figure 5.32 and 5.33. Both vertical and horizontal field comparisons are made.

Figure 5.34 shows the vertical comparison of the path losses at the end of

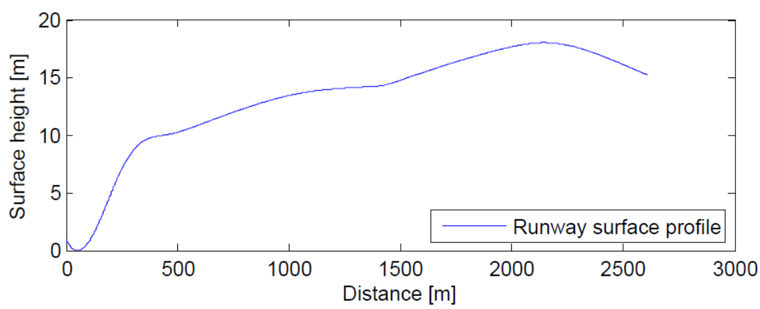


Figure 5.31: Surface profile of the runway at the Braunschweig airport. Please note the scaling difference between the x- and z-axis.

the runway, compared with the path loss over a flat surface. The figure shows that at lower heights, the path loss of the field over the runway is larger than the path loss for a flat surface. This is as expected because of diffraction effects over the runway, and there no line-of-sight from the transmitter. However, at higher altitudes the path losses over the runway and the flat surface approach the same

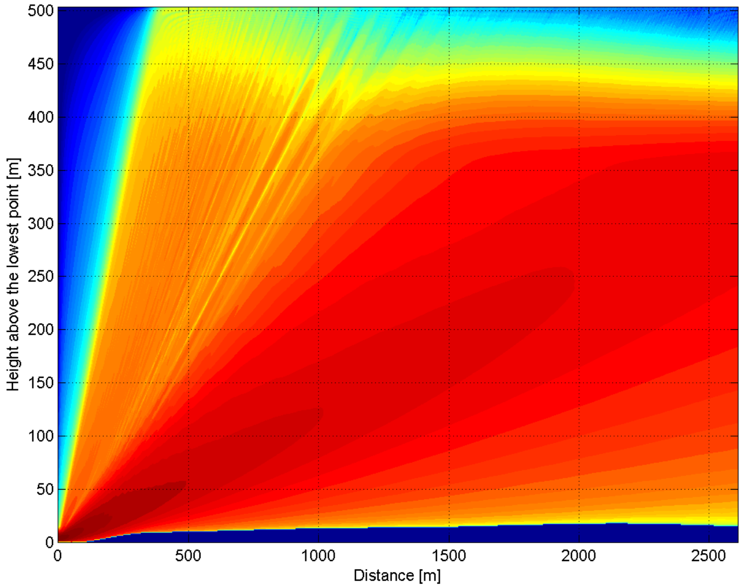


Figure 5.32: Field over the Braunschweig airport, using SSA.

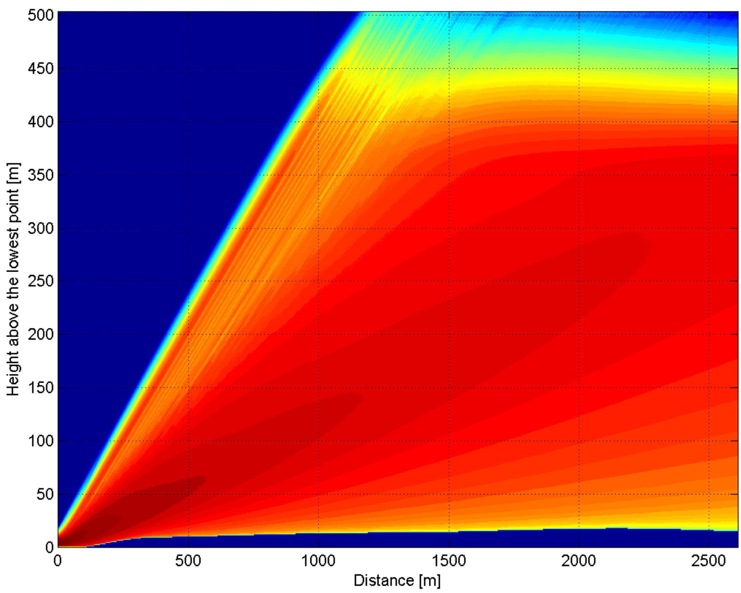


Figure 5.33: Field over the Braunschweig airport, using FDM.

value. This is because the effect of the ondulating terrain will gradually decrease. The path losses obtained using the SSA and FDM are very close to equal. They are closer than for a flat surface.

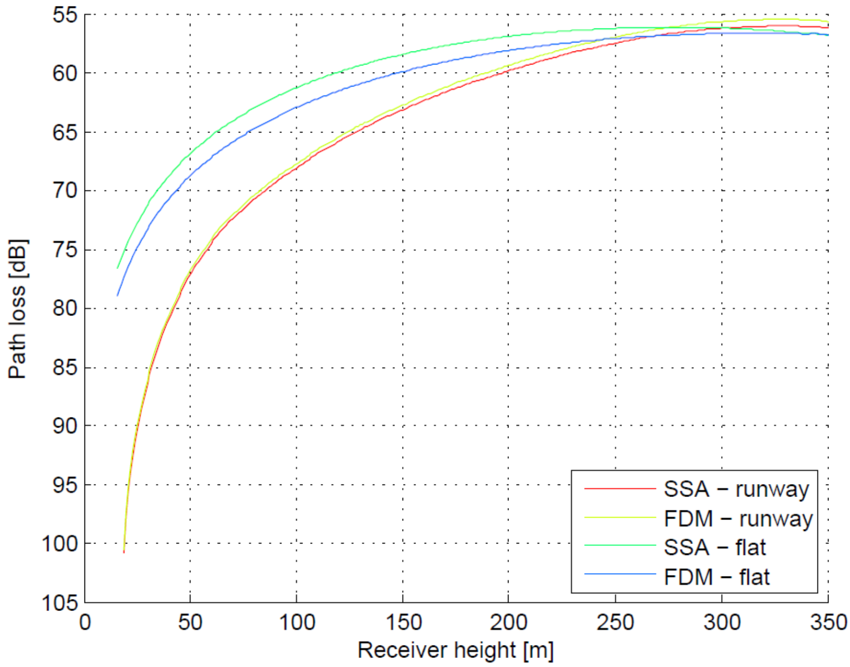
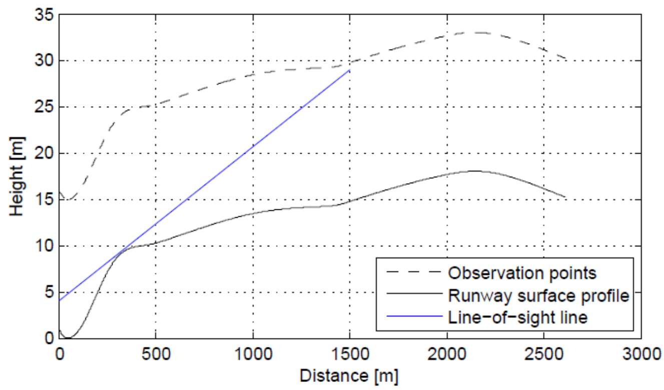


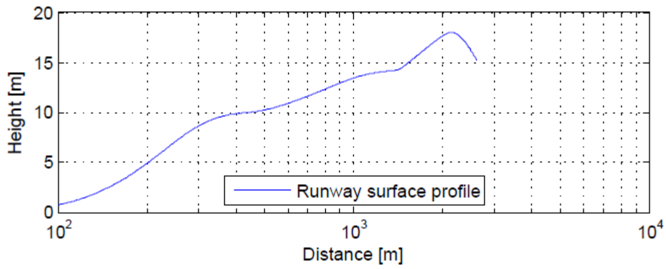
Figure 5.34: Vertical comparison of the path losses of the SSA and FDM at the end of the runway. Compared with flat surface as well.

Figure 5.35c shows horizontal comparison with height of comparison of 15 m above all surface points, see figure 5.35a. The results are not shifted, so the comparison is the same as field strength comparison. Figure 5.35c shows that until approximately 500 m, the field strength of the runway and the flat surface have similar pattern. This is within line-of-sight from the transmitter. From approximately 500 m, still within line-of-sight, the field strength of over the runway starts to decrease faster than the field strength on the flat surface. Near the end of the runway, the runway profile goes down. Here, the field strength drops. The signals over the runway are affected by runway profile even when the observation points are within line-of-sight, most likely because the surface of the runway is within the first Fresnel zone of the observation points. This means that diffraction effects from the surface affects the signals.

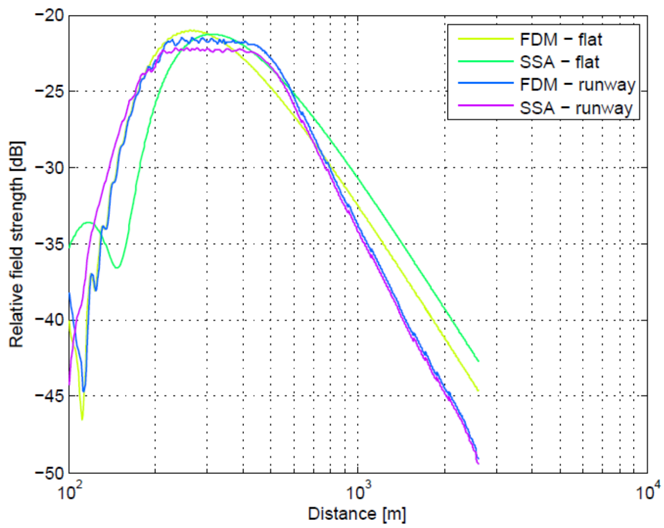
The SSA and FDM over the runway follow each other closely, especially when the field strength starts to decrease, at approximately 500 m. From 500 m, they almost overlap. The SSA and FDM over the runway follow each other closer than SSA and FDM over a flat surface.



(a) Runway surface profile, line-of-comparison, and line-of-sight line.



(b) Runway surface profile



(c) Horizontal comparison at 15 m above the surface at each point.

Figure 5.35: Horizontal comparison between the field over the runway and over a flat surface. Figure 5.35b is for runway surface reference.

5.4.2 The Luton Airport Runway

The simulations over the Luton airport runway were run using the *Luton2.m* script in appendix E.1.9. The runway at Luton airport has the profile given in figure 5.36. The simulated fields using the SSA and FDM methods are given in figure 5.37 and 5.38.

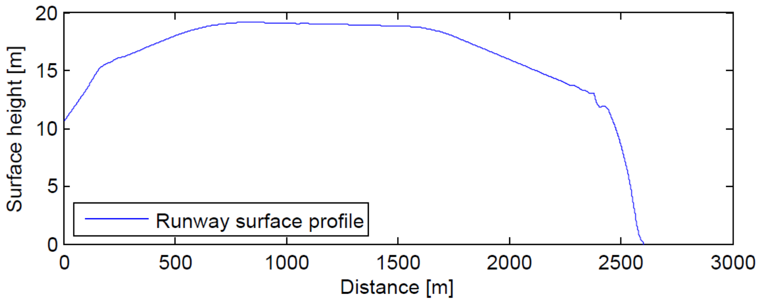


Figure 5.36: The surface profile of the runway at the Luton airport. Please note the scaling difference between the x- and z-axis.

The vertical field comparison between the SSA and FDM at the end of the runway is given in figure 5.39. The figure shows that the simulations from SSA and FDM are consistent, they almost overlap in the entire region. The end of the runway is behind a hump, seen from the localizer. For the vertical comparison, the height of which there will be line-of-sight from the localizer to the points of comparison, is at approximately 70 m, see figure 5.40a. This can somehow be seen in figure 5.39 because the difference between the path loss of a flat surface and the runway is relatively large. When the altitude gets higher, the effect of the runway surface will gradually decrease, and the path loss will approach the path loss for a flat surface. This is as expected since the hump will no longer affect the signals. The signals continue to be affected by the runway surface, even when the observation point is within line-of-sight, because some part of the surface is within the Fresnel zones.

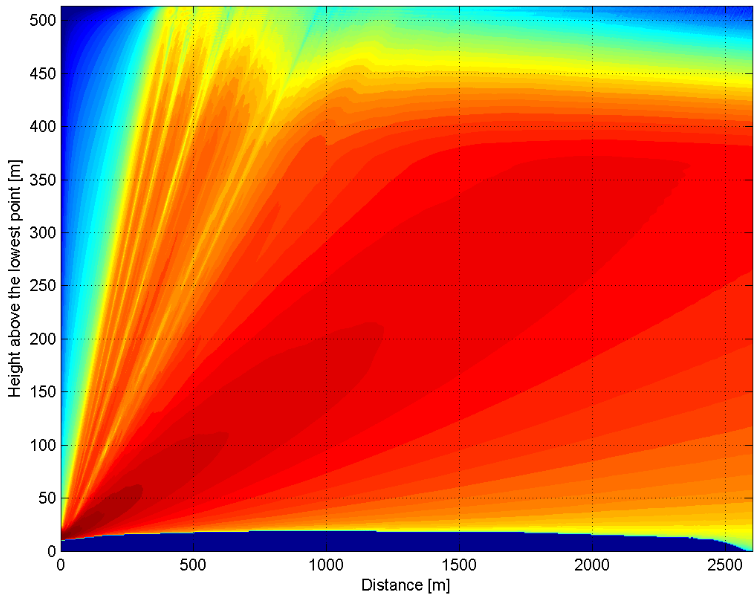


Figure 5.37: Field over the Luton airport, using SSA.

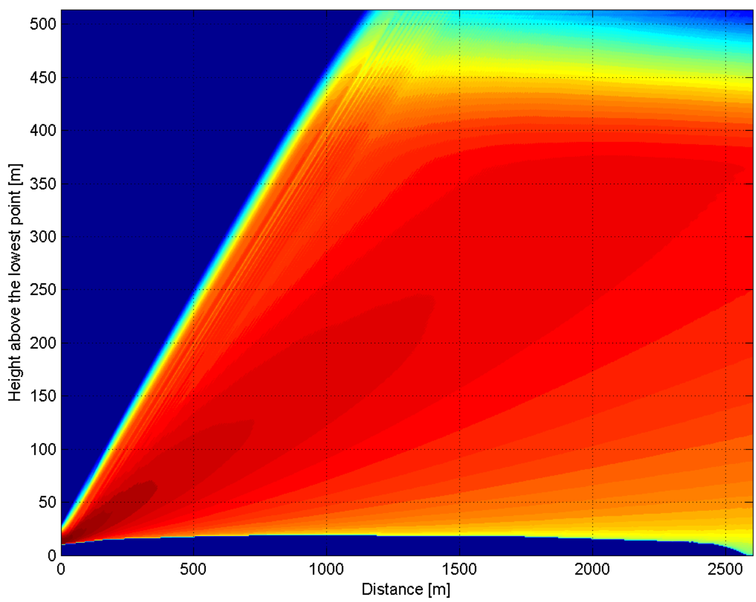


Figure 5.38: Field over the Luton airport, using FDM.

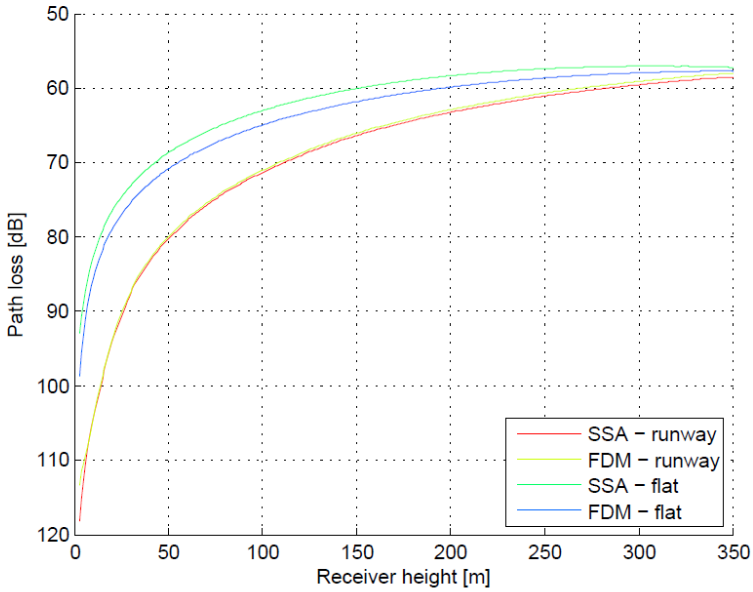
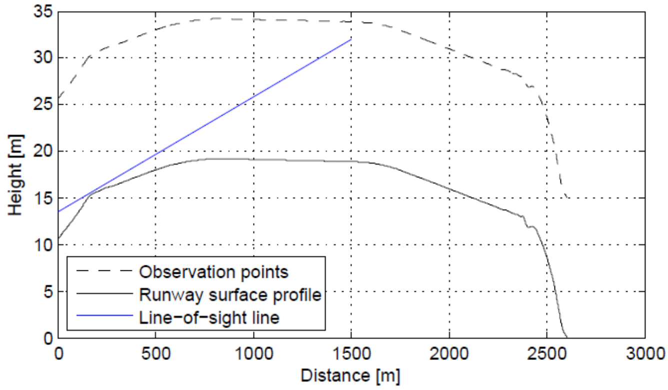


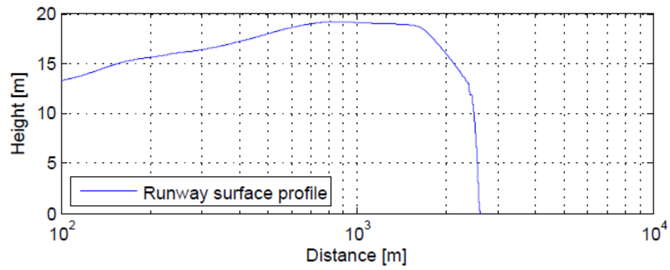
Figure 5.39: Vertical comparison of the path losses of the SSA and FDM at the end of the runway. Compared with flat surface as well.

Figure 5.40c shows horizontal comparison at the height of 15 m above the surface, along the runway. The results are not shifted, so the comparison is the same as field strength comparison. The figure shows that the hump shape of the runway affects the signals, even when the observation point is within line-of-sight from the transmitter, see figure 5.40a. The hump starts to affect the signals at a distance of approximately 400 m, see figure 5.40c, which is within line-of-sight from the transmitter. This is most likely because some part of the terrain is within the first Fresnel zone. The field strength on the runway starts to differ from the field strength on a flat surface at the distance of approximately 400 m. From this point and on, the field strength starts to decrease gradually faster than for a flat surface. When the surface profile drops near the end, the field strength drops too. Figure 5.40b is for surface reference.

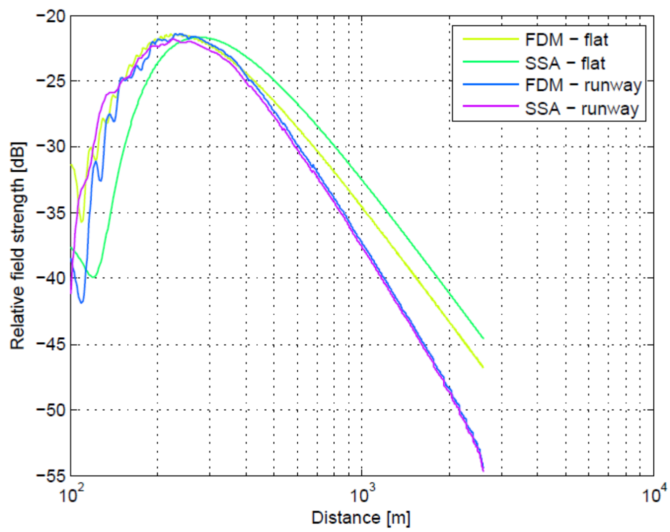
The SSA and FDM for the runway follow each other closely over the entire runway. Up to 300 m they have similar pattern. From 300 m and up, they almost overlap each other.



(a) Runway surface profile with observation point, and line-of-sight line.



(b) Runway surface profile. Note that the x-axis is not linear.



(c) Horizontal comparison at 15 m above the surface at each point. The SSA and FDM from the runway overlap each other at larger distances.

Figure 5.40: Horizontal comparison between the field over the runway and over a flat surface. Figure 5.35b is for runway surface reference.

5.4.3 Runway Simulation Summary

The simulations over the runways show that the results using the SSA and FDM follow each other closely over most part of the runways. When seen from the "transmitter antenna", for shorter distances the SSA and FDM differ slightly, and for larger distances they almost overlap. In vertical comparison they almost overlap each other in the entire domain. The results also show that the path loss is larger, the field strength is lower, when propagating over ondulating terrain, which is like expected. At higher altitudes, when the waves will no longer be affected by the terrain, the results from the simulations over the runway will approach the same value as the results from a flat surface, which is also like expected.

Based on the results from the previous sections, it is likely that the simulations over the runways give a realistic indication of the field strength and path losses. Except at very low heights.

5.5 Results Summary

For a flat surface, the FDM results are consistent with the analytical results. The SSA has the same shape, but does not overlap the analytical results like the FDM. For the inclined planes, the wedge, and the runways, the SSA and FDM overlap at larger distances for horizontal comparison. In vertical comparison they have the same shape in almost the entire domain. Close to the surface the results from the inclined surface decrease faster than the analytical result and the results from flat surface. The SSA and FDM from the inclined surface follows each other in the entire domain. For the wedge, the path loss comparisons showed that in the interval of altitude 20 to 200 m, the simulated results, SSA and FDM, have the same slope as the results in Hviid et al. [1995]. Over the runways, for horizontal comparison, the SSA and FDM follow each other closely from approximately 200 m. For vertical comparison, they follow each other in the entire domain. The comparisons with flat surface show that there can be significant differences between the field strength over a flat surface and a humped runway. Provided that the results from the runways can be trusted.

Based on the results, the SSA and FDM can handle undulating terrain, except a very low heights, and it is likely that the results that give realistic indications of the field strengths.

A $\frac{1}{r}$-compensation for simulation of 3D propagation in 2D is too conservative. It consistently underestimates the field strength, overestimates the loss.

Chapter 6

Discussion and Further Work

Based on the results, the Parabolic Equation Method is suitable for field propagation simulation of signals at 110 MHz, the frequency of the ILS localizer, for signals propagating over a humped runway.

6.1 Field Strength Near the Surface for Inclined Plane

The results show that for an inclined plane, the field decreases faster close to the ground than for a flat surface. For downwards propagation it can be explained by the zero-padding when the terrain goes down. When the terrain goes down one step, the field values of the points within the height of this step will be zero. When the field continue to propagate at this height, the zero-padded points will get their values from the propagating field. Therefore, the field values close to the ground will be less accurate, and the values smaller, since the zero-padded points influence the propagation. For upwards propagation, the principle is the opposite, and why the field decreases fast close to the ground remains subject to further investigation. Due to time constraint, there was not time for that in this thesis.

6.2 Modeled Surface Resolution

When modeling a surface, its resolution will be the same as the step size in the x- and z-direction, Δx and Δz, for the field propagation algorithms, SSA and FDM. In this thesis, a step size of 1 m in both directions is used. This means that for small height differences, the terrain modeling can be quite rough. Smaller step size would therefore be preferable, however, this will drastically increase the run time of the algorithm. In addition, according to chapter 4, only the FDM algorithm can handle smaller step sizes. For small step sizes, the SSA does not converge. There exist other methods for handling irregular terrain, like "piecewise linear terrain" and "conformal mapping", [Levy, 2000, p. 97,100]. They are not implemented and remain subject to further work.

6.3 SSA and FDM Differences - Flat and Non-Flat Surface

The results show that there is less difference between the SSA and FDM when the surface is non-flat than for a flat surface. Both the SSA and the FDM solve the standard parabolic equation. The SSA by solving the equation in the Fourier domain, and the FDM by solving the equation directly, by discretization of the equation. Since they both solve the same equation, their results should be quite similar. This is the same for both flat and irregular surface. The reason for this difference between flat and irregular surface, remains a subject to further investigation. Due to time constraint, it is beyond the limits of this thesis.

6.4 Localizer Signals and Wide-Angle Propagation

The results show that the runway surface profile can affect electromagnetic waves at the frequency of the localizer significantly. For simulation of how the localizer signals will be affected, a wide-angle propagation algorithm has to be used. The implemented algorithm is a narrow angle propagation algorithm, where the preferred beamwidth is 40°, ±20°, or narrower. The implemented algorithm has been tested for signals of half-power beamwidth of 55°, ±27.5°, with good results. It is therefore possible that the results can be reasonable with a beamwidth of ±40°. However, wide-angle propagation algorithms would be preferable. The wide-angle propagation algorithms are more complex extensions of the implemented narrow-angle propagation algorithms. They were not implemented because it was necessary to know that the implemented algorithms works well, before extending them. It is beyond the scope of this thesis, and is subject for further work.

6.5 Runtime

When running the SSA and the FDM, the runtime of both algorithms is approximately equal. It is hard to compare the number of computations for the algorithms, because of matrix inversion in the FDM. The algorithms give similar results. For a flat surface the FDM is closer to the analytical result than the SSA. However, over undulating terrain they give the same result.

6.6 Commercial Software

There exists commercial software for electromagnetic simulations, like Computer Simulation Technology (CST). The software was not used because it is very complex, and due to time constraint, there was not enough time to learn how to use it. In addition, it is not used in this industry, simulation of ILS and the propagation of its radio waves. Indra does not use it, and Airbus recently developed their own simulation software, ELISE. The ELISE software does only work for a flat runway.

6.7 3D Loss in 2D

It should be possible to introduce "artificial" loss in a 2D model, in order to compensate for the additional loss in 3D. As already stated in the results, the difference between the losses is not as large as $\frac{1}{r}$. At least not when the beam is directive. One way to do this may be to transform the 3D source of interest into an equivalent 2D source, using a path loss correction factor. This is proposed for the Finite-Difference Time-Domain algorithm in Wu et al. [2008]. Due to time constraint, there was not time to work on it within this thesis. It is left for future work.

6.8 3D - Parabolic Equation

The parabolic equation method can be extended to 3D. When doing so the method can adapt to irregular terrain in the transverse direction as well. The algorithm will then become more complex, the number of computations will increase dramatically, and so will the run time of the algorithm as well. At the same time, the simulations will be a lot closer to the reality because the propagated wave will be a spherical wave, and the terrain will have extension in the transverse direction. Correct wave propagation in 3D will be a challenge, because the wave will spread in two dimensions at the same time.

Similar for the sky in the case of 2D, the end of the computational domain in the transverse direction will as well need an absorption layer on the sides, in order to avoid numerical reflection.

If the terrain has the same assumption as in 2D, no variations in the transverse direction, the number of computations can be reduced by introducing a plane of symmetry at $y = 0$, according to figure 6.1. This works if the source is centered at $(x, y) = (0, 0)$, and the source is not tilted in the transverse direction.

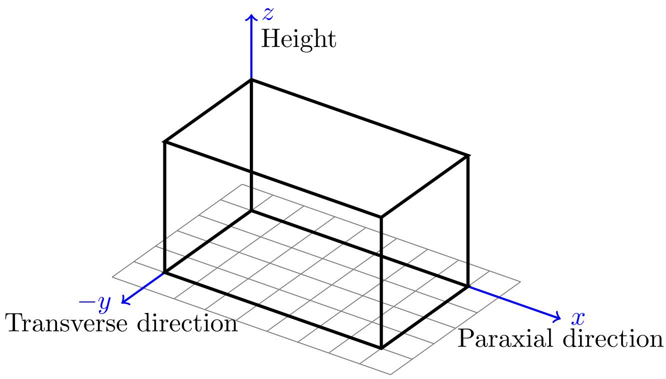


Figure 6.1: Computational domain for simulation using the Parabolic Equation method in 3D.

6.9 Integral Equation Model

The Integral Equation Model is implemented as far as possible in this project. Some more research on this method can resolve the remaining issues. Since this method also was "invented" for irregular terrain, it could be very interesting to compare the performance of this method with the Parabolic Equation Method. What remains on the implementation of the Integral Equation Model is the calculation of the electric field at the receiver, based on the calculated induced currents along the surface. In order to do so, further research on the following points are necessary:

- Which points along the surface contribute to the total field at the receiver point?
- What kind of source is the surface?

The implemented code is available via NTNU.

Chapter 7

Conclusion

The Parabolic Equation Method is suitable for simulation of electromagnetic field propagation over a surface with undulating terrain, at the frequency of the ILS localizer. The computational domain for the simulations consists of boundary conditions for a perfect conductor at the ground, free-space propagation, and an absorption layer for damping of the waves at the top of the computational domain. The simulation results over a flat surface and up- and downwards inclined plane are consistent with the analytical results. The algorithms can also handle a wedge. It is therefore likely that the results over the humped runways are reasonable. They show that a humped runway surface can affect electromagnetic signals at the frequency of the ILS localizer considerably. In order to predict the propagation of electromagnetic signals at the frequency of the ILS localizer over a humped runway surface, the runway surface profile needs to be taken into account. This can be done using the Parabolic Equation Method. The suitability and performance of the Integral Equation Model remains unknown.

The Parabolic Equation Method is implemented in 2D, for narrow beam propagation. It is a building block for wide extension possibilities like wide-angle propagation, 3D loss in 2D, and 3D implementation.

References

C.A. Balanis. *Antenna Theory: Analysis and Design.* John Wiley & Sons, 2005. 20

Jean-Pierre Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114(2):185–200, 1993. 35

C. Brennan and P.J. Cullen. Application of the fast far-field approximation to the computation of uhf pathloss over irregular terrain. *Antennas and Propagation, IEEE Transactions on*, 46(6):881 –890, jun 1998. 15, 18, 19

W.C. Gibson. *The Method of Moments in Electromagnetics.* Taylor & Francis, 2007. 14, 15

R Holm. *Grunnleggende ILS teori, revisjon 6.* Avinor, 2002. ix, 1, 2

J.T. Hviid, J.B. Andersen, J. Toftgard, and J. Bojer. Terrain-based propagation model for rural area-an integral equation approach. *Antennas and Propagation, IEEE Transactions on*, 43(1):41 –46, jan 1995. 16, 18, 43, 49, 50, 74, 77, 78, 82, 93, 150

G. Kouzaev. Microwave techniques TTT4205_2. Lecture notes in TTT 4205, fall 2011, 2011. 25

Landing-Systems. Welcome to the educational ils program. `"http://instrument.landing-system.com/ils-tutorial-animation/`. Visisted: 08.06.2013. ix, 2

Y.H. Lee. *Introduction to Engineering Electromagnetics.* Springer London, Limited, 2013. 25, 27

M.F. Levy. *Parabolic Equation Methods for Electromagnetic Wave Propagation.* Electromagnetic Waves Series. Institution of Electrical Engineers, 2000. 10, 27, 28, 30, 31, 32, 96, 105, 106, 107, 108, 109

S. Saunders and A. Aragón-Zavala. *Antennas and Propagation for Wireless Communication Systems: 2nd Edition.* John Wiley & Sons, 2007. ix, 6, 9, 10, 14, 18, 43

K Åstebøl. Datamaskin-assistert analyse av omgivelsenes påvirkning på signalytelsen til flylandingssystemet ILS. 2012. 1

G. Strang. *Linear algebra and its applications.* Thomson Brooks/Cole Cengage learning, 2006. 33

The MathWorks Inc. dst, idst. URL `http://www.mathworks.se/help/pde/ug/idst.html`. Visited: March 7th 2013. 106

Yan Wu, Min Lin, and I. Wassell. Path loss estimation in 3d environments using a modified 2d finite-difference time-domain technique. In *Computation in Electromagnetics, 2008. CEM 2008. 2008 IET 7th International Conference on*, pages 98–99, 2008. 97

Appendix A

Mathematical Tools

A.1 Fourier Transform

The definitions of the Fourier transform and the inverse Fourier transform, respectively, are given in equation (A.1) and (A.2), [Levy, 2000, p. 13].

$$U(x,p) = \mathcal{F}\left\{u(x,z)\right\} = \int\limits_{-\infty}^{\infty} u(x,z)\mathrm{e}^{-i2\pi pz}\mathrm{d}z \tag{A.1}$$

$$u(x,z) = \mathcal{F}^{-1}\left\{U(x,p)\right\} = \int\limits_{-\infty}^{\infty} U(x,p)\mathrm{e}^{i2\pi pz}\mathrm{d}p \tag{A.2}$$

A.2 Fourier Sine Transform

The Fourier Sine Transform, equation (A.3), [Levy, 2000, p. 25]:

$$U(x,p) = \mathcal{S}\{u(x,z)\} = \int_{0}^{+\infty} u(x,z)\sin(2\pi pz)\mathrm{d}z \tag{A.3}$$

A.3 Discrete Fourier Sine Transform

Discrete Fourier Sine Transform, equation (A.4), [The MathWorks Inc.]:

$$U(x,p) = \mathcal{S}\{u(x,n)\} = \sum_{n=1}^{N} u(x,n) \sin\left(\pi \frac{p \cdot n}{N+1}\right), p = 1, \cdots, N \tag{A.4}$$

Inverse Discrete Fourier Sine Transform, equation (A.5), [The MathWorks Inc.]:

$$u(x,n) = \mathcal{S}^{-1}\{U(x,p)\} = \frac{2}{N+1} \sum_{n=1}^{N} U(x,p) \sin\left(\pi \frac{p \cdot n}{N+1}\right), p = 1, \cdots, N \tag{A.5}$$

A.4 Approximations of Differentials

The first order derivative is given by equation (A.6), and the second order derivative is given by equation (A.7), [Levy, 2000, p.36].

$$\frac{\partial u}{\partial x}(\xi_m, z_j) = \frac{u(x_m, z_j) - u(x_{m-1}, z_j)}{\Delta x_m} \tag{A.6}$$

$$\frac{\partial^2 u}{\partial x^2}(\xi_m, z_j) = \frac{u(\xi_{m+1}, z) + u(\xi_{m-1}, z_j) - 2u(\xi_m, z_j)}{\Delta x^2} \tag{A.7}$$

Appendix B

Derivations

B.1 Derivation of the Standard Parabolic Equation

$$\begin{aligned} &\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial z^2} + k^2 n^2 \psi = 0 \\ &x\text{: direction of propagation} \\ &z\text{: height} \\ &k\text{: wave number in vacuum, } \frac{2\pi}{\lambda} \\ &n\text{: refractive index, function of x and z, slowly varying} \end{aligned} \tag{B.1}$$

Introducing $u(x,z) = \mathrm{e}^{ikx}\psi(x,z)$, [Levy, 2000, p. 5], and filling $u(x,z)$ into the scalar wave equation, equation (B.1), gives the scalar wave equation for $u(x,z)$, equation (B.2).

$$\begin{aligned} &(E)\text{: } \frac{\partial^2 u}{\partial x^2} + i2k\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial z^2} + k^2(n^2-1) = 0 \\ &(E) \Longleftrightarrow \left\{\frac{\partial}{\partial x} + ik(1-Q)\right\}\left\{\frac{\partial}{\partial x} + ik(1+Q)\right\} u = 0 \\ &Q = \sqrt{\frac{1}{k^2}\frac{\partial^2}{\partial z^2} + n^2(x,z)} \end{aligned} \tag{B.2}$$

Q is a pseudo differential operator, meaning that the operator itself contains partial derivatives and regular functions of the variables. The Q operator is valid for the set of functions $u(x,z)$ satisfying equation (B.3), [Levy, 2000, p. 6].

$$Q(Q(u)) = \frac{1}{k^2}\frac{\partial^2 u}{\partial z^2} + n^2 u \tag{B.3}$$

The motivation for simplifying the scalar wave equation for u, like in equation (B.2), is to discover that u is a sum of forward and backward propagating waves, see equation (B.4).

$$\begin{cases} u = u_+ + u_- \\ \dfrac{\partial u_+}{\partial x} = -ik(1-Q)u_+\text{: Forward propagating wave} \\ \dfrac{\partial u_+}{\partial x} = -ik(1+Q)u_-\text{: Backward propagating wave} \end{cases} \tag{B.4}$$

Forward propagating waves are the only waves of interest. In order to find u_+, an approximation of the differential equation for u_+, the standard parabolic equation, can be found by approximating the square-root in the Q operator with a first order Taylor series expansion, leading to the standard parabolic equation, see equation (B.5).

$$\begin{aligned} &\text{Taylor series expansion around } x = 0\text{: } \sqrt{1+x} \simeq 1 + \frac{1}{2}x \\ &Q = \sqrt{\frac{1}{k^2}\frac{\partial^2}{\partial z^2} + n^2(x,z)} = \sqrt{\frac{1}{k^2}\frac{\partial^2}{\partial z^2} + n^2(x,z) - 1 + 1} \\ &\Rightarrow Q \simeq 1 + \frac{1}{2}\left(\frac{1}{k^2}\frac{\partial^2}{\partial z^2} + n^2 - 1\right) \\ &\Rightarrow \left\{\frac{\partial}{\partial x} + ik(1-Q)\right\}u = 0 \Leftrightarrow \frac{\partial u}{\partial x} + ik\left(1 - 1 - \frac{1}{2}\left(\frac{1}{k^2}\frac{\partial^2}{\partial z^2} + n^2 - 1\right)\right)u = 0 \\ &\Leftrightarrow \frac{\partial^2 u}{\partial z^2} + i2k\frac{\partial u}{\partial x} + k^2(n^2-1) = 0\text{: The standard parabolic equation} \end{aligned} \tag{B.5}$$

B.2 Derivation of the Numerical Standard Parabolic Equation

To simplify the notation, $u(x_m, z_j) = u_m^j$. When "converting" the original SPE, equation (B.6), to a numerically implementable form, the first differential term, $\frac{\partial^2 u}{\partial z^2}(\xi_m, z_j)$, is given in equation (B.8), and the second differential term, $\frac{\partial u}{\partial x}(\xi_m, z_j)$ is given in equation (B.9). In order for the solution to propagate, the midpoint between the current point m and the previous point $m-1$ along the x-axis is considered, ξ_m^j, [Levy, 2000, p. 36]. ξ_m^j is defined according to equation (B.7).

$$\frac{\partial^2 u}{\partial z^2}(x,z) + 2ik\frac{\partial u}{\partial x}(x,z) + k^2(n^2(x,z)-1)u(x,z) = 0 \tag{B.6}$$

$$\xi_m = \frac{x_{m-1}+x_m}{2} \tag{B.7}$$

$$\frac{\partial^2 u}{\partial z^2}(\xi_m, z_j) = \frac{u_{m-1}^{j+1} + u_m^{j+1} + u_{m-1}^{j-1} + u_m^{j-1} - 2u_{m-1}^j - 2u_m^j}{2\Delta z^2} \tag{B.8}$$

$$\frac{\partial u}{\partial x}(\xi_m, z_j) = \frac{u_m^j - u_{m-1}^j}{\Delta x} \tag{B.9}$$

Inserting equation (B.8) and (B.9) into equation (B.6), results in equation (B.10).

$$\frac{u_{m-1}^{j+1} + u_m^{j+1} + u_{m-1}^{j-1} + u_m^{j-1} - 2u_{m-1}^j - 2u_m^j}{2\Delta z^2} + 2ik\frac{u(x_m, z_j) - u(x_{m-1}, z_j)}{\Delta x} + \frac{k^2}{2}(n^2(\xi_m, z_j) - 1)(u_{m-1}^j + u_m^j) = 0 \tag{B.10}$$

By Setting $b = 4ik\frac{\Delta z^2}{\Delta x}$ and $a_m^j = k^2({n_m^j}^2 - 1)\Delta z^2$, and inserting this into equation (B.10), leads to equation (B.11).

$$u_{m-1}^{j+1} + u_m^{j+1} + u_{m-1}^{j-1} + u_m^{j-1} - 2u_{m-1}^j - 2u_m^j + b\left(u_m^j - u_{m-1}^j\right) + a_m^j\left(u_{m-1}^j + u_m^j\right) = 0 \tag{B.11}$$

Rearranging equation (B.9) such that all terms including point m are at the right-hand side, and all terms including point $m-1$ are at the left-hand side leads to equation (B.12).

$$u_m^{j-1} + u_m^j\left(-2 + b + a_m^j\right) + u_m^{j+1} = -u_{m-1}^{j-1} + u_{m-1}^j\left(2 + b - a_m^j\right) - u_{m-1}^{j+1} \tag{B.12}$$

If the current point is m, this means that the value of u at the range x_m can be determined by knowing the value of u at x_{m-1}, for all z-values of interest. Equation (B.12) is the case for a given height j. However, the relation is the same for any height, and can be summarized into a matrix form, equation (B.13), [Levy, 2000, p. 38], where U_{m-1} contains the field values for all heights at x-range $m-1$, and U_m is to be determined. The tridiagonal matrices contain only one number in the first and last line. This is due to the boundary conditions at the top and bottom where the field is zero and remains unchanged. The initial field should be zero at the top and bottom.

$$\begin{bmatrix} 1 & & & & \\ 1 & \alpha_m^1 & 1 & & \\ & & \ddots & & \\ & & 1 & \alpha_m^{N-1} & 1 \\ & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_m \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ -1 & \beta_m^1 & -1 & & \\ & & \ddots & & \\ & & -1 & \beta_m^{N-1} & -1 \\ & & & & -1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_{m-1} \\ \vdots \\ \vdots \end{bmatrix},$$

$$U_m = \begin{bmatrix} u_m^0 \\ u_m^1 \\ \vdots \\ u_m^{N-1} \\ u_m^N \end{bmatrix},\ U_{m-1} = \begin{bmatrix} u_{m-1}^0 \\ u_{m-1}^1 \\ \vdots \\ u_{m-1}^{N-1} \\ u_{m-1}^N \end{bmatrix} \tag{B.13}$$

Appendix C

Plots

C.1 Field Simulation Over a Wedge

Figure C.1 and C.2 show the total simulated fields over the wedge given in figure 5.25, using the SSA and FDM, respectively.

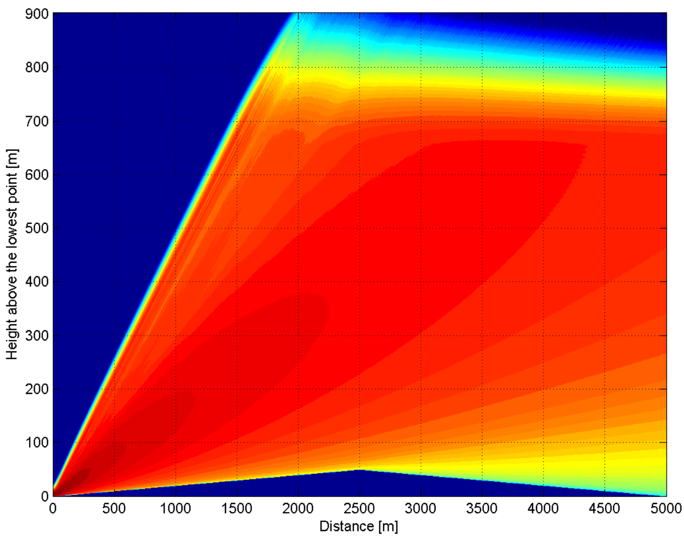


Figure C.1: Field simulation over the wedge given in figure 5.25 using the SSA. Frequency: 100 MHz.

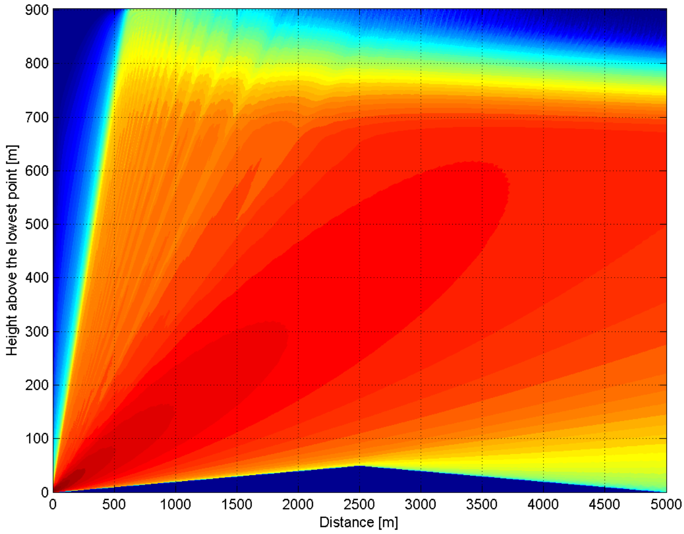


Figure C.2: Field simulation over the wedge given in figure 5.25 using the FDM. Frequency: 100 MHz.

Appendix D

Implementation and Simulation

D.1 Implementation Terminology

The terminology used in the implementation is consistent in all functions and scripts.

- Vector: 1-dimensional array: [n $\times$ 1]
- Grid: 2-dimensional array: [n $\times$ m]
- Coordinates in 2D (array): (z, x) - [lines (height), columns (distance)] in an array In the implementation x and z have changed order due to the visualization of what an array "looks like" in Matlab. The first coordinate represents the lines, height, and the second coordinate columns, distance.

D.2 Simulation Using the Implemented Functions

The functions are implemented in Matlab. In order to do simulations using the implemented functions, the procedure below has to be followed.

D.2.1 Create Initial Field

Firstly, a height vector including height points for the absorption layer needs to be created. The function $createZvectAbsorptionLayer2$ does that.

Secondly, the initial field can be created, using the function $createInitialField$.

D.2.2 Irregular Surface

For an irregular surface, a set of points describing the surface is necessary. The *interpolate* function interpolates a set of surface points, in order to have appropriate spacing between the surface points.

If the surface has surface points below zero, the surface has to be shifted upwards so that the lowest point is located at altitude zero. The function $normalizeSurface$ performs this operation.

D.2.3 Field Propagation Algorithms

The field propagation algorithms with associated implemented functions are listed below:

- SSA flat surface: $splitStepAlgorithmAbsorptionLayer$
- FDM flat surface: $FDMAbsorptionLayerNumEfficient2$
- SSA irregular surface: $SSAirregularTerrainAbsoptionLayer$
- FDM irregular surface: $FDMirregularTerrainAbsorptionLayer$

Appendix E

Implemented Code

This appendix contains the implemented code; the scripts for obtaining the results, the field propagation algorithms, and the helping functions. The zip-file attached to the thesis contains all the scripts and functions below, plus some functions and scripts that are not in use.

E.1 Scripts for the Obtained Results

E.1.1 ParabolicEquation_noGround.m

Simulation of field in free-space using the FDM. Used in section 3.2.5.

```
% ParabolicEquation_noGround.m: Script that simulate free-space only, for
%                                testing the absorption layer.
clear all

% Setting the parameters:
theta0 =0;
beta = pi/15;
A = 2;
frequency = 100*10^6;
deltaX =1;
maxX = 5000;
maxHinterestHeight =20;
numLayersArray  =10;
numPointsPerLayerArray =20;
```

```
numElts = length(numPointsPerLayerArray);
numPointsInLayer =100;
deltaZarr = 1;

% Looping over the deltaZ values in question:
for b = 1: length(deltaZarr)
    deltaZ = deltaZarr(b);

    antHeight =10;

    % Looping over the antenna heights in question:
    for a =1:length(antHeight)
        counter = 0;
        simulationType = cell(numElts,1);
        for heightIndex = 1: length(maxHinterestHeight)
            for n = 1: length(numLayersArray)

                for m = 1: length(numPointsPerLayerArray)
                    counter = counter +1;
                    numLayers = numLayersArray(n);
                    numPointsPerLayer = numPointsPerLayerArray(m);

                    zs = antHeight(a);

                    maxHinterest =260;

                    % Creating z-vector with absorption layer:
                    [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2( ...
                        maxHinterest, deltaZ,numPointsInLayer);

                    % Creatin initial field:
                    initialFieldFDM =createInitialField(zs,theta0,beta,...
                        zVectFDM, A, frequency,'gaussian1');
                    numZpoints = length(zVectFDM);


                    % Creating the x vector:
                    xVect = verticalVector([0:deltaX:maxX]);


                    numIterations = ceil(maxX/xVect(numZpoints));
                    xVectTot = xVect;
                    L = length(zVectFDM);
                    sourceIndex = ceil(((L/deltaZ) +1)*(zs/L));

                    % Simulating the field:
                    tic
                    [uValuesFDMalne,maxEigVal,antennaSourceIndex] = ...
```

```
                FDMnoGround(initialFieldFDM, ...
                zVectFDM,xVect,HindexFDM,frequency,...
                numPointsInLayer,sourceIndex);
            uValuesTot = uValuesFDMalne;

            toc

            deltaZstr = num2str(deltaZ);
            yText = 'Height distance [m]';
            %

            eField = uValuesTot(antennaSourceIndex,:);
            simulationType{counter,1} = ['FDM'];

            if counter == 1
                eFieldTot = eField;
            else

                eFieldTot = vertcat(eFieldTot,eField);
            end

            tx = zs;
            rx = zs;

            % Plotting the simulated field:
            fig = figure('visible','off');
            uValuesAux = uValuesTot;
            uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
            contourf(10.*log10(abs(uValuesAux.^2)),50)
            hold on
            contour(10.*log10(abs(uValuesAux.^2)),50)

            part1Title = ['FDM - absorption layer test'];
            part12Titile = [ ' ','\Deltaz = ',' ',deltaZstr,...
                'm, ','\Deltax = ',' ', ...
                num2str(deltaX),'m,'];
            part21Title = ['Distance from center point of ',...
                'source to beginning of absorption layer: ',...
                num2str(maxHinterest-zs),'m' ];
            part2Title=['Number of points in absorption layer:',...
                ' ', num2str(numPointsInLayer)];

            titleVal2 = {part1Title;part12Titile;part21Title;...
                part2Title};
            title(titleVal2)
            xlabel('Distance [m]');
            ylabel(yText);
            grid on
```

```
                    titleFig = [...
                        'FDM_noGround/FDM_AbsorptioLayerTest_ant_h_',...
                        '_',num2str(antHeight(a)), ...
                        '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
                        num2str(deltaX),'_max_h_',...
                        num2str(maxHinterestHeight(heightIndex)), ...
                        'num_points_layer', num2str(numPointsInLayer),...
                        '.png'];
                    saveas(fig,titleFig,'png');
                end
            end
        end
        part1Title = ['FDM - absorption layer test'];
        part12Titile = [ ' ','\Deltaz = ',' ',deltaZstr,...
            'm, ','\Deltax = ',' ', ...
            num2str(deltaX),'m,'];
        part21Title = ['Distance from center point of ',...
            'source to beginning of absorption layer: ',...
            num2str(maxHinterest-zs),'m'];
        part2Title=['Number of points in absorption layer:',...
            ' ', num2str(numPointsInLayer)];

        plotTitle = {part1Title;part12Titile;part21Title;...
            part2Title};

        titleFig = ['FDM_noGround/FDM_AbsorptioLayerTest_ant_h_','_',...
            num2str(antHeight(a)), ...
            '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
            num2str(deltaX),'_max_h_',...
            num2str(maxHinterestHeight(heightIndex)), ...
            'num_points_layer', num2str(numPointsInLayer),...
            'newAbsLAyer4','.png'];
        tx = 10000;
        rx = 10000;

        % Comparing the simulated free-space loss with analytical
        % free-space loss:
        freeSpaceLoss_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
            beta,frequency,simulationType, plotTitle,titleFig);


    end
end
```

E.1.2 ParabolicEquation_SSA_FDM.m

Field simulation over a flat surface using the SSA and FDM, Δx and Δz vary. Used in chapter 4.

```
% parabolicEquation_SSA_FDM.m: Script running simulations with varying
%                              delta x and delta x values over a flat
%                              surface using the SSA and FDM.

clear all

% Setting the parameters:
theta0 = 0;
beta =pi/18;
A = 1;
frequency = 110*10^6;
maxX = 3000;
numPtsAbsoptionLayer = 150;

deltaZarr =[0.5 1 1.3];
antHeight = [15];
deltaXvect =[0.5 1 1.3];
for a =1:length(antHeight)


    zs = antHeight(a);
    maxHinterestHeight = 10;
    numElts = 2;
    simulationType = cell(numElts,1);

    % Looping over the delta x values:
    for n = 1:length(deltaXvect)
        clear eFieldTot
        counter = 0;
        doubleCounter = 0;

        maxHinterestHeight = 10;

        for heightIndex = 1: length(maxHinterestHeight)

            % Looping over the delta z values
            for  b = 1: length(deltaZarr)
                deltaZ = deltaZarr(b);

                counter = counter +1;
                doubleCounter = doubleCounter +1;
```

```
        deltaX = deltaXvect(n);
        xVect = verticalVector([0:deltaX:maxX]);
        maxHinterest = 350 +zs;

        % Creating initial field:
        [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
            maxHinterest,deltaZ,numPtsAbsoptionLayer);
        initialFieldFDM = createInitialField(zs,theta0,beta,...
            zVectFDM,A, frequency,'gaussian1');
        numZpoints = length(zVectFDM);

        % Calculating field:
        tic
        uValuesSplitStep =splitStepAlgorithmAbsorptionLayer(...
            initialFieldFDM,zVectFDM,xVect, ...
            HindexFDM,frequency,numPtsAbsoptionLayer);
        toc
        tic
        [uValuesFDMalne,maxEigVal]=...
            FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
            zVectFDM,xVect,HindexFDM,frequency,...
            numPtsAbsoptionLayer);
        toc

        deltaZstr = num2str(deltaZ*10);
        yText = strcat('Height above surface [m]');

        % Extract the simulated fields at the height of interest:
        eFieldSSA=uValuesSplitStep(...
            ceil(HindexFDM*(zs/maxHinterest)),:);
        eFieldFDM=uValuesFDMalne(...
            ceil(HindexFDM*(zs/maxHinterest)),:);

        simulationType{doubleCounter,1} = ['SSA: \Deltax=',...
            num2str(deltaX),'m, \Deltaz=',num2str(deltaZ),'m'];
        doubleCounter = doubleCounter +1;
        simulationType{doubleCounter,1} = ['FDM:\Deltax=',...
            num2str(deltaX),'m, \Deltaz=',num2str(deltaZ),'m'];
        tx = zs;
        rx = zs;

        if counter == 1
            eFieldTot = eFieldSSA;
            eFieldTot = vertcat(eFieldTot,eFieldFDM);
        else
            eFieldTot = vertcat(eFieldTot,eFieldSSA);
            eFieldTot = vertcat(eFieldTot,eFieldFDM);
        end
```

```

            % Plot the simulated fields:
            fig2 = figure('visible','off');
            uValuesAux = uValuesSplitStep;
            uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
            contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
            hold on
            contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)

            xlabel('Distance [m]');
            ylabel(yText);
            grid on
            titleFig = ['Results_to_thesis/SSA_', ...
                'Deltaz','_',deltaZstr,'_','Deltax_','_', ...
                num2str(deltaX*10),'_distance_source_abslayer',...
                num2str(maxHinterest-zs),'freq_',...
                num2str(frequency/(10^6)),'_deltaDiff.png'];
            saveas(fig2,titleFig,'png');


            fig = figure('visible','off');
            uValuesAux = uValuesFDMalne;
            uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
            contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
            hold on
            contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
            xlabel('Distance [m]');
            ylabel(yText);
            grid on
            titleFig = ['Results_to_thesis/FDM_', ...
                'Deltaz','_',deltaZstr,'_','Deltax_','_', ...
                num2str(deltaX*10),'freq_',...
                num2str(frequency/(10^6)),'_deltaDiff.png'];
            saveas(fig,titleFig,'png');




        end

        tx =zs;
        rx = zs;
        part1Title = ['SSA and FDM - flat surface'];
        titleFig = ['Results_to_thesis/SSA_FDM_', ...
            'Deltaz','_',deltaZstr,'_','Deltax_','_', ...
            num2str(deltaX*10),'freq_',...
                num2str(frequency/(10^6)),'_deltaDiff.png'];
        % Comparison between simulated and analytical results:
```

```
                pathLossFlat_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
                    beta,frequency,simulationType, ' ',titleFig);

        end
    end
end
```

E.1.3 ParabolicEquation_SSA_FDM_deltaValueTest.m

Script showing the effect of ”slow” propagation. Used in chapter 4.

```
% parabolicEquation_SSA_FDM_deltaValueTes.m: Script simulating the ''slow''
%                                             propagation, the value of
%                                             delta x and delta z is small


% Setting the parameters:
theta0 = 0;
beta = pi/18;
A = 1;
frequency = 110*10^6;
deltaX = 1;
maxX = 3000;
numPtsAbsoptionLayer = 150;
deltaZarr =[0.3];
antHeight = [15];
deltaXvect = [0.3];

for a =1:length(antHeight)
    counter = 0;
    doubleCounter = 0;

    zs = antHeight(a);
    maxHinterestHeight = 10;
    numElts = 2;
    simulationType = cell(numElts,1);

    % Looping over the delta x values:
    for n = 1:length(deltaXvect)

        maxHinterestHeight = 10;

        for heightIndex = 1: length(maxHinterestHeight)

            % Looping over the delta z values:
```

```
        for  b = 1: length(deltaZarr)
            deltaZ = deltaZarr(b);

            counter = counter +1;
            doubleCounter = doubleCounter +1;
            deltaX = deltaXvect(n);
            xVect = verticalVector([0:deltaX:maxX]);
            maxHinterest = 350 +zs;

            % Creating initial field:
            [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
                maxHinterest,deltaZ,numPtsAbsoptionLayer);
            initialFieldFDM = createInitialField(zs,theta0,beta,...
                zVectFDM,A, frequency,'gaussian1');
            numZpoints = length(zVectFDM);

            % Calculating field:
            tic
            uValuesSplitStep =splitStepAlgorithmAbsorptionLayer(...
                initialFieldFDM,zVectFDM,xVect, ...
                HindexFDM,frequency,numPtsAbsoptionLayer);
            toc
            tic
            [uValuesFDMalne,maxEigVal]=...
                FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
                zVectFDM,xVect,HindexFDM,frequency,...
                numPtsAbsoptionLayer);
            toc

            deltaZstr = num2str(deltaZ);
            yText = strcat('Height above surface [m]');

            % Extracting field at the height of interest, the antenna
            % height:
            eFieldSSA=uValuesSplitStep(...
                ceil(HindexFDM*(zs/maxHinterest)),:);
            eFieldFDM=uValuesFDMalne(...
                ceil(HindexFDM*(zs/maxHinterest)),:);

            simulationType{doubleCounter,1} = ['SSA:'];
            doubleCounter = doubleCounter +1;
            simulationType{doubleCounter,1} = ['FDM:'];
            tx = zs;
            rx = zs;

            if counter == 1
                eFieldTot = eFieldSSA;
                eFieldTot = vertcat(eFieldTot,eFieldFDM);
```

```
        else
            eFieldTot = vertcat(eFieldTot,eFieldSSA);
            eFieldTot = vertcat(eFieldTot,eFieldFDM);
        end

        % Plotting the fields:
        fig2 = figure('visible','off');
        uValuesAux = uValuesSplitStep;
        uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
        contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        hold on
        contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        xlabel('The surface');
        ylabel(yText);
        grid on
        titleFig = ['DeltaValueTest_results/SSA', ...
            '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
            num2str(deltaX),'freq_',...
            num2str(frequency/(10^6)),'_deltaTest.png'];
        saveas(fig2,titleFig,'png');


        fig = figure('visible','off');

        uValuesAux = uValuesFDMalne;
        uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
        contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        hold on
        contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        xlabel('The surface');
        ylabel(yText);
        grid on
        titleFig = ['DeltaValueTest_results/FDM', ...
            '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
            num2str(deltaX),'freq_',...
            num2str(frequency/(10^6)),'_deltaTest_deltaTest.png'];
        saveas(fig,titleFig,'png');

    end

    tx =zs;
    rx = zs;

    plotTitle =' ';
    titleFig = ['DeltaValueTest_results/SSA_FDM', ...
        '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
        num2str(deltaX),'freq_',...
            num2str(frequency/(10^6)),'_deltaTest.png'];
```

```

            % Comparing path loss of simulated field with the analytical
            % path loss
            pathLossFlat_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
                beta,frequency,simulationType, plotTitle,titleFig);

        end
    end
end
```

E.1.4 SSA_FDM_indra_r_loss.m

Script for generation of the results for a flat surface, section 5.1.

```
% SSA_FDM_indra_r_loss.m: Script calcualting the electric field along a
% flat surface and compare the results with analytical results from Indra.
% In the computations, additional loss, (1/r) is added in order to ''make''
% a 3D model.

clear all

% Setting the parameters:
theta0 = 0;
beta =(55/(2*360))*(2*pi);
A =10;
frequency = 110*10^6;

deltaX = 1;
maxX = 3000;
compareDistance = 1000;
numPtsAbsoptionLayer = 150;
deltaZ = 1;
antHeight = 3;
deltaXvect = [1];
counter = 0;
doubleCounter = 0;
maxHinterestHeight = 10;
numElts = 2;
simulationType = cell(numElts,1);

counter = counter +1;
doubleCounter = doubleCounter +1;

xVect = verticalVector([0:deltaX:maxX]);
maxHinterest = 350 +antHeight;
```

```

% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');

% Finding the gain of the used beam [dBi]:
maxValueInitField = max(initialFieldFDM);
sumInitField = sum(initialFieldFDM);
findRes = find(abs(initialFieldFDM)>0);
numResPts = length(findRes);
isotropicSource = sumInitField/numResPts;
dbiGain = maxValueInitField/isotropicSource

numZpoints = length(zVectFDM);

% Calculating field with 1/r-loss added to the results:
tic
uValuesSplitStep =SSA_addRloss(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer,antHeight);
toc
tic
[uValuesFDMalne]=...
    FDM_addRloss(initialFieldFDM, ...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer,antHeight);
toc

deltaZstr = num2str(deltaZ);
yText = strcat('Height above surface [m]');

% Extracting the simulated values at the height of the anntenna:
eFieldSSA=uValuesSplitStep(ceil(HindexFDM*(antHeight/maxHinterest)),:);
eFieldFDM=uValuesFDMalne(ceil(HindexFDM*(antHeight/maxHinterest)),:);

simulationType{1,1} = ['SSA'];
doubleCounter = doubleCounter +1;
simulationType{2,1} = ['FDM'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);


%Plot SSA figuure:
fig2 = figure('visible','off');
```

```
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesSplitStep;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['Indra_r_loss_added/SSA_flat_rLoss.png'];
saveas(fig2,titleFig,'png');

%Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDMalne;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['Indra_r_loss_added/FDM_flat_rLoss.png'];
saveas(fig,titleFig,'png');
%
% Comparing with results from Indra (along the surface):
tx =antHeight;
rx = antHeight;
rx_xArr = xVect;
rx_zArr = ones(length(xVect),1).*antHeight;
part1Title = ['SSA and FDM - flat surface - along the surface'];
plotTitle = ' ';
titleFig = ['Indra_r_loss_added/SSA_FDM_flat_along_surface.png'];
filename = 'IndraWedge2_results/LPDA-u-kile-2_E.xls';

pathLossIndra_alongX(rx_xArr,rx_zArr,eFieldTot,frequency,...
    simulationType, plotTitle,titleFig,filename);

% Comparing with results from Indra with varying height at the end of the
% surface:
xIndex = find(xVect >= 1000,1);
```

```
eFieldSSA1 = zeros(1,length(zVectFDM));
eFieldFDM1 = zeros(1,length(zVectFDM));
for i = 1:length(zVectFDM)
    eFieldSSA1(1,i) = uValuesSplitStep(i,xIndex);
    eFieldFDM1(1,i) = uValuesFDMalne(i,xIndex);
end

plotTitle1 = ' ';

eFieldTot1 = eFieldFDM1;
eFieldTot1 = eFieldSSA1;
eFieldTot1 = vertcat(eFieldTot1,eFieldFDM1);
rx_xArr1 = ones(length(zVectFDM),1).*length(xVect);
rx_zArr1 = zVectFDM;

titleFig1 = ['Indra_r_loss_added/SSA_FDM_flat_height_varying_rLoss.png'];
filename1 = 'IndraWedge_results/LPDA-u-kile_E.xls';

compareHeight = 240;
pathLossFlat_Indra(rx_xArr1,rx_zArr1,eFieldTot1,...
    frequency,simulationType, plotTitle1,titleFig1,filename1,compareHeight)

% Calculating the field strength without any additional 1/r-loss:
tic
uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc

tic
[uValuesFDM2,maxEigVal]=FDMAbsorptionLayerNumEfficient2(initialFieldFDM,...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc

% Extracting the field strength at 1000 m in the vertical direction;
xIndex = find(xVect >= 1000,1);
eFieldSSA2 = zeros(1,length(zVectFDM));
eFieldFDM2 = zeros(1,length(zVectFDM));
for i = 1:length(zVectFDM)
    eFieldSSA2(1,i) = uValuesSSA2(i,xIndex); %...
    %length(xVect));
    eFieldFDM2(1,i) = uValuesFDM2(i,xIndex); %...
    %length(xVect));
end
eFieldTot2 = eFieldSSA2;
eFieldTot2 = vertcat(eFieldTot2,eFieldFDM2);

part1Title2 = ['SSA and FDM - flat surface - receiver height varying '...
```

```
    '- no (1/r)-loss added'];
plotTitle2 =' ';% {part1Title1};

rx_xArr2 = ones(length(zVectFDM),1).*length(xVect);
rx_zArr2 = zVectFDM;

titleFig2 =['Indra_r_loss_added/SSA_FDM_flat_height_varying_no_rLoss.png'];
filename2 = 'IndraWedge_results/LPDA-u-kile_E.xls';
% Tests the plotting the results in the same plot:
compareHeight = 240;
eFieldTot3  = eFieldSSA1;
eFieldTot3 = vertcat(eFieldTot3,eFieldFDM1);
eFieldTot3 = vertcat(eFieldTot3,eFieldSSA2);
eFieldTot3 = vertcat(eFieldTot3,eFieldFDM2);

simulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - 1/r-loss added'];
simulationType2{2,1} = ['FDM - 1/r-loss added'];
simulationType2{3,1} = ['SSA - no 1/r-loss added'];
simulationType2{4,1} = ['FDM - no 1/r-loss added'];

% Vertical comparison of relative field strengths with different max
% heights:
 pathLossFlat_Indra(rx_xArr2,rx_zArr2,eFieldTot3,...
     frequency,simulationType2, plotTitle2,titleFig2,filename2,...
     compareHeight)


 compareHeight = 50;
 titleFig3 =['Indra_r_loss_added/',...
     'SSA_FDM_flat_height_varying_no_rLoss_zoomed.png'];
  pathLossFlat_Indra(rx_xArr2,rx_zArr2,eFieldTot2,...
     frequency,simulationType, plotTitle2,titleFig3,filename2,...
     compareHeight)


 titleFig3 =['Indra_r_loss_added/',...
     'SSA_FDM_flat_height_varying_no_rLoss_zoomed2.png'];
  pathLossFlat_Indra_minComp(rx_xArr2,rx_zArr2,eFieldTot2,...
     frequency,simulationType, plotTitle2,titleFig3,filename2,...
 compareHeight)

% Due to an error when first implementing the script, the fields without
% additional loss is calculated ones more:

% Setting the parameters:
theta0 = 0;
A = 10^(0.9);
```

```
frequency = 110*10^6;
deltaX = 1;
numPtsAbsoptionLayer = 150;

deltaZarr =[1];
antHeight = 3;
deltaXvect = [1];

Xn = [0 maxX];
Zn = [0 0];


for a =1:length(antHeight)
    counter = 0;
    doubleCounter = 0;

    zs = antHeight(a);
    maxHinterestHeight = 10;
    numElts = 2;
    simulationType = cell(numElts,1);

    % Looping over the delta x values:
    for n = 1:length(deltaXvect)

        maxHinterestHeight = 10;

        for heightIndex = 1: length(maxHinterestHeight)

            % Looping over the delta z values
            for  b = 1: length(deltaZarr)
                deltaZ = deltaZarr(b);

                counter = counter +1;
                doubleCounter = doubleCounter +1;
                deltaX = deltaXvect(n);

                maxHinterest = 350 +zs;

                % Interpolate the flat surface (algorithm for irregular
                % terrain used):
                [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
                    'linear',deltaX);

                % Create initial field:
                [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
                    maxHinterest,deltaZ,numPtsAbsoptionLayer);
                initialFieldFDM = createInitialField(zs,theta0,beta,...
                    zVectFDM,A, frequency,'gaussian1');
```

```
                numZpoints = length(zVectFDM);

                % Calculating the fields:
                tic
                uValuesSplitStep3 = SSAirregularTerrainAbsoptionLayer(...
                    initialFieldFDM,zVectFDM,...
                    xVect,zSurfaceVect,HindexFDM,frequency,...
                    numPtsAbsoptionLayer,zs);
                toc
                tic
                uValuesFDM3 = FDMirregularTerrainAbsorptionLayer ...
                    (initialFieldFDM,zVectFDM,xVect,zSurfaceVect,...
                    HindexFDM,frequency,numPtsAbsoptionLayer);
                toc

                % Extracting the fields ath the given distance:
                xIndex = find(xVect >= compareDistance,1);
                eFieldSSA3 = zeros(1,length(zVectFDM));
                eFieldFDM3 = zeros(1,length(zVectFDM));
                for i = 1:length(zVectFDM)
                    eFieldSSA3(1,i) = uValuesSplitStep3(i,...
                        xIndex);
                    eFieldFDM3(1,i) = uValuesFDM3(i,...
                        xIndex);
                end

                deltaZstr = num2str(deltaZ);
                yText = strcat('Height above surface [m]');

                simulationType{doubleCounter,1} = ['SSA'];
                doubleCounter = doubleCounter +1;
                simulationType{doubleCounter,1} = ['FDM'];
                tx = zs;
                rx = zs;

                if counter == 1
                    eFieldTot3 = eFieldSSA3;
                    eFieldTot3 = vertcat(eFieldTot3,eFieldFDM3);
                else
                    eFieldTot3 = vertcat(eFieldTot3,eFieldSSA3);
                    eFieldTot3 = vertcat(eFieldTot3,eFieldFDM3);
                end

                % Plotting the fields:
                fig2 = figure('visible','on');

                part1Title = ['Split-Step Algorithm - wedge'];
                part12Titile = [ ' ','\Deltaz = ',' ',deltaZstr,...
```

```
            'm, ','\Deltax = ',' ', ...
            num2str(deltaX),'m, Source height: ', ...
            num2str(antHeight(a)),'m'];
        part21Title = ['Distance from center point of ',...
            'source to beginning of absorption layer: ',...
            num2str(maxHinterest-zs),'m' ];
        part2Title=['Number of points in absorption layer:',...
            ' ', num2str(numPtsAbsoptionLayer)];

        titleVal2 = {part1Title;part12Titile;part21Title;...
            part2Title};

        uValuesAux = uValuesSplitStep3;
        uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
        contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        hold on
        contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        %title(titleVal2)
        xlabel('Distance [m]');
        ylabel(yText);
        grid on
        titleFig =['Indra_r_loss_added/SSA_flat_noRloss','.png'];
        saveas(fig2,titleFig,'png');

        fig = figure('visible','on');

        part1Title = ['Finite-Difference Method - wedge'];
        part12Titile = [ ' ','\Deltaz = ',' ',deltaZstr,...
            'm, ','\Deltax = ',' ', ...
            num2str(deltaX),'m, Source height: ', ...
            num2str(antHeight(a)),'m'];
        part21Title = ['Distance from center point of ',...
            'source to beginning of absorption layer: ',...
            num2str(maxHinterest-zs),'m' ];
        part2Title=['Number of points in absorption layer:',...
            ' ', num2str(numPtsAbsoptionLayer)];

        titleVal2 = {part1Title;part12Titile;part21Title;...
            part2Title};

        uValuesAux = uValuesFDM3;
        uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
        contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        hold on
        contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
        %title(titleVal2)
        xlabel('Distance [m]');
        ylabel(yText);
```

```
        grid on
        titleFig =['Indra_r_loss_added/FDM_flat_noRloss.png'];

        saveas(fig,titleFig,'png');
        titleFig2=['Indra_r_loss_added/FDM_ant_h_','_',...
            num2str(antHeight(a)), ...
            '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
            num2str(deltaX),'_distance_source_abslayer',...
            num2str(maxHinterest-zs),'wedge.pdf'];
        %print (fig, '-dpdf', titleFig2);

    end

    tx =zs;
    rx = zs;
    part1Title = ['SSA and FDM - wedge '];
    part12Titile = [ ' ','\Deltaz = ',' ',deltaZstr,...
        'm, ','\Deltax = ',' ', ...
        num2str(deltaX),'m, Source height: ', ...
        num2str(antHeight(a)),'m'];
    part21Title = ['Distance from center point of ',...
        'source to beginning of absorption layer: ',...
        num2str(maxHinterest-zs),'m' ];
    part2Title=['Number of points in absorption layer:',...
        ' ', num2str(numPtsAbsoptionLayer)];

    plotTitle = {part1Title;part12Titile;part21Title;...
        part2Title};

    titleFig =['Indra_r_loss_added/SSA_FDM_ant_h_','_',...
        num2str(antHeight(a)), ...
        '_','Deltaz','_',deltaZstr,'_','Deltax_','_', ...
        num2str(deltaX),'_distance_source_abslayer',...
        num2str(maxHinterest-zs),'wedge2.png'];

    filename = 'IndraWedge_results/LPDA-u-kile_E.xls';
    rx_xArr = ones(length(zVectFDM),1).*length(xVect);
    rx_zArr = zVectFDM;

    % Vertical comparison:
    pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,...
        eFieldTot3,zs,...
        beta,frequency,simulationType,' ',titleFig,filename);

    % Comparing with results from Indra (along the surface):
    tx =antHeight;
    rx = antHeight;
    rx_xArr = xVect;
```

```
            rx_zArr = ones(length(xVect),1).*antHeight;
            part1Title = ['SSA and FDM - flat surface - along the surface- no additional l
            plotTitle = ' ';
            titleFig = ['Indra_r_loss_added/SSA_FDM_flat_along_surface_noLossAdded.png'];
            filename = 'IndraWedge2_results/LPDA-u-kile-2_E.xls';

            eFieldSSA3=uValuesSplitStep3(ceil(HindexFDM*(antHeight/maxHinterest)),:);
            eFieldFDM3=uValuesFDM3(ceil(HindexFDM*(antHeight/maxHinterest)),:);
            eFieldTot3 = eFieldSSA3;
            eFieldTot3 = vertcat(eFieldTot3,eFieldFDM3);

            pathLossIndra_alongX(rx_xArr,rx_zArr,eFieldTot3,frequency,...
                simulationType, plotTitle,titleFig,filename);


        end

    end
end
```

E.1.5 DownwardsInclinedPlane.m

Script for generation of the results for the downwards inclined plane, section 5.2.1.

```
% DownwardsInclinedPlane.m: Perform simulations on a downward inclined
%                           plane, and then compares the results with a
%                           flat plane. The beam propagating along the
%                           inclined plane has the same relative directivty
%                           as the flat plane. The comparison between the
%                           fields are done in the height direction.

clear all
% Setting the parameters for the inclined plane:
beta =(55/(2*360))*(2*pi);
A = 10^(0.9);
frequency = 110*10^6;

xDiff = 1000;
zDiff = -20;

theta0 = abs(asin(zDiff/xDiff)); % The tilt of the beam

deltaX = 1;
deltaZ = 1;
```

```
maxX = 3100;
minZ = -maxX*sin(theta0);
numPtsAbsoptionLayer = 150;
xDist = 3000;

deltaZarr =[1];
antHeight = 3;
deltaXvect = [1];

% The irregular terrain:
Xn = [0  maxX];
Zn = [0 minZ];

maxZcompare = 270;

% Interpolating the surface:
[xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
    'linear',deltaX);
% Shifts the surface:
[zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);

% Adjusting the antenna height:
antHeight = antHeight + zSurfaceNorm(1);
maxHinterest = 330 +antHeight;

simulationType = cell(4,1);

% Plot the surface:
surfacePlot = figure();
plot(xVect,zSurfaceNorm)
xlabel('Distance [m]');
ylabel('Surface height [m]');
titleFig = [...
    'DownwardsInclinedPlane_results/DownwardsInclinedPlane_surface.png'];
saveas(surfacePlot,titleFig,'png');

% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating the field over the downwards inclined plane:
tic
uValuesSplitStep = SSAirregularTerrainAbsoptionLayer(...
    initialFieldFDM,zVectFDM,...
    xVect,zSurfaceNorm,HindexFDM,frequency,...
```

```
    numPtsAbsoptionLayer,antHeight);
toc
tic
uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
    (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
deltaZstr = num2str(deltaZ);
yText = strcat('Height above the lowest point [m]');

%maxZ = 250;

% Extracting the vertical values:
eFieldSSA  = verticalVector(getVerticalValues(...
    xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesSplitStep,'down'))';


eFieldFDM = verticalVector(getVerticalValues(...
    xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesFDM,'down'))';

simulationType{1,1} = ['SSA inclined'];

simulationType{2,1} = ['FDM inclined'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);



minPlotLevel = 10^(-4);
for i = 1:1
% Plot SSA figure:
plotScale =length(zVectFDM);
fig2 = figure('visible','off');
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};
tic
uValuesAux = uValuesSplitStep(1:plotScale,:);
uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
disp('contourf is on')
contourf(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
```

```
hold on
disp('contour is on')
contour(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
eFieldSSA  = verticalVector(getVerticalValues(xVect,...
    zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesSplitStep,'down'))';

%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on

titleFig = ['DownwardsInclinedPlane_results/SSA_InclDown.png'];
disp('saving ...')
saveas(fig2,titleFig,'png');
toc

tic
% Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDM(1:plotScale,:);
uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
contourf(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
eFieldFDM = verticalVector(getVerticalValues(xVect,...
    zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesFDM,'down'))';
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['DownwardsInclinedPlane_results/FDM_InclDown.png'];
saveas(fig,titleFig,'png');
toc

end % Plot figure, not in use
```

```
% Comparing with results from Indra (along the surface),
% horizontal comparison:

tx =antHeight;
rx = antHeight;
rx_xArr = xVect;
rx_zArr = ones(length(xVect),1).*antHeight;

plotTitle = ' ';
titleFig = ['DownwardsInclinedPlane_results/SSA_FDM_InclDown.png'];
filename = 'IndraSource/wedge1/LPDA-u-kile_E.xls';


rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;

simulationType1 = cell(2,1);
simulationType1{1,1} = 'SSA inclined';
simulationType1{2,1} = 'FDM inclined';

pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eFieldTot,...
    antHeight,beta,frequency,simulationType1, plotTitle,titleFig,filename)
%zDist = 250;


% Calculating the field over a flat surface
antHeight = antHeight - zSurfaceNorm(1);
theta0 = 0;
%maxHeight = 250;
xVect = verticalVector([0:deltaX:xDist]);
% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep_flat =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
tic
[uValuesFDMalne_flat,maxEigVal]=...
    FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc
```

```

% Extracting the results for vertical comparison:
eFieldSSA = zeros(1,maxZcompare+1); %maxHeight+1);
eFieldFDM = zeros(1,maxZcompare+1); %maxHeight+1);
for i = 1:maxZcompare+1 %maxHeight+1
    eFieldSSA(1,i) = uValuesSplitStep_flat(i,...
        length(xVect));
    eFieldFDM(1,i) = uValuesFDMalne_flat(i,...
        length(xVect));
end

simulationType{3,1} = ['SSA flat'];
simulationType{4,1} = ['FDM flat'];
eFieldTot = vertcat(eFieldTot,eFieldSSA);
eFieldTot = vertcat(eFieldTot,eFieldFDM);

% Comparing with results from Indra (along the surface),
% horizontal comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;
part1Title = ' ';%['SSA and FDM - flat surface - along the surface'];
plotTitle = ' '; % {part1Title};

titleFig_comp = [...
    'DownwardsInclinedPlane_results/SSA_FDM_compare_DownIncl.png'];
filename = 'IndraWedge_results/LPDA-u-kile_E.xls';
numCases = 4;
startIndex = 4;
numElts = length(zVectFDM);

eFieldTot1 = zeros(numCases,length(eFieldTot(1,:))); %startIndex:numElts)));
for i = 1:numCases
   eFieldTot1(i,:) = eFieldTot(i,:); %startIndex:numElts);
end


compareHeight = 50;
 titleFig3 =['DownwardsInclinedPlane_results/',...
     'SSA_FDM_flat_height_varying_DownIncl_zoomed1.png'];
  pathLossFlat_Indra_minComp(rx_xArr,rx_zArr,eFieldTot1,...
     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)

 titleFig4 =['DownwardsInclinedPlane_results/',...
     'SSA_FDM_flat_height_varying_DownIncl_zoomed2.png'];
  pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
     frequency,simulationType, plotTitle,titleFig4,filename,compareHeight)
```

```
compareHeight = 240;

 pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
    frequency,simulationType, plotTitle,titleFig_comp,filename,compareHeight)

% Plot SSA figure:
yText = strcat('Height above surface [m]');
fig2 = figure('visible','off');
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesSplitStep_flat;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['DownwardsInclinedPlane_results/SSA_flat_InclDown.png'];
saveas(fig2,titleFig,'png');

% Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDMalne_flat;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['DownwardsInclinedPlane_results/FDM_flat_InclDown.png'];
saveas(fig,titleFig,'png');


% Compare along the surface at constant height of 40 m above the lowest
% point:

%eFieldAlongTot
height = 40;
startIndex = 101; %100;
```

```
eSSArunway = (uValuesSplitStep(height,startIndex:length(xVect)));
eFieldAlongTot  =  eSSArunway;
eFDMrunway = (uValuesFDM(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMrunway);

eSSAflat = (uValuesSplitStep_flat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eSSAflat);
eFDMflat = (uValuesFDMalne_flat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMflat);

imulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - downwards'];
simulationType2{2,1} = ['FDM - downwards'];
simulationType2{3,1} = ['SSA - flat'];
simulationType2{4,1} = ['FDM - flat'];

rx_xArr3 = xVect(startIndex:length(xVect));
rx_zArr3 =  ones(length(rx_xArr3),1).*height;
titleFig3 = ['DownwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal_InclDown.png'];
antHeight2 = antHeight - zSurfaceNorm(1);

titleFig4 = ['DownwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal2_InclDown.png'];

pathLossIndra_alongX(rx_xArr3,rx_zArr3,eFieldAlongTot,...
    frequency,simulationType2,' ',titleFig4,' ')


% Comparing fields along the surface at constant height above the surface
height2 = 15;
startIndex2 = 101;

fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
eSSArunway2 = zeros(length(fieldVect2),1);
eFDMrunway2 = zeros(length(fieldVect2),1);
for i = 1:length(fieldVect2)
    eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
    eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
end

eFieldAlongTot2  =  eSSArunway2';
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMrunway2');
eSSAflat2 = (uValuesSplitStep_flat(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
eFDMflat2 = (uValuesFDMalne_flat(height,startIndex2:length(xVect)));
```

```
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);

titleFig5 = ['DownwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal_cst_diff_surface_InclDown.png'];
rx_xArr4 = xVect(startIndex2:length(xVect));
rx_zArr4 =  ones(length(rx_xArr4),1).*height;


pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2,...
    frequency,simulationType2,' ',titleFig5,' ')

% Plotting the surface profile with logaritmic axes:
fig = figure();
semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
     (startIndex2:length(xVect))); %,'Parent',ax2);

legend('Surface profile','Location','SouthWest');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 200;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['DownwardsInclinedPlane_results/',...
    'Runway_profile_small_InclDown.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);
```

E.1.6 UpwardsInclinedPlane2

Script for generation of the results for the downwards inclined plane, section 5.2.2.

```
% UpwardsInclinedPlan2e.m: Perform simulations on an upwards inclined
%                          plane, and then compares the results with a
%                          flat plane. The beam propagating along the
%                          inclined plane has the same relative directivty
%                          as the flat plane. The comparison between the
%                          fields are done in the height direction.

clear all
% The inclined plane:
```

```
beta =(55/(2*360))*(2*pi);
A = 10^(0.9);
frequency = 110*10^6;

xDiff = 1000;
zDiff = 20;

theta0 = abs(asin(zDiff/xDiff));

% The delta x and delta z value:
deltaX = 1;
deltaZ = 1;
maxX = 3000;
minZ = maxX*sin(theta0);
numPtsAbsoptionLayer = 150;
xDist = 3000;
deltaZarr =[1];
antHeight = 3;
deltaXvect = [1];

% The irregular terrain:
Xn = [0  maxX];
Zn = [0 minZ];
maxZcompare = 270;

% Interpolate the surface profile:
[xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
    'linear',deltaX);

zSurfaceNorm = zSurfaceVect;
antHeight = antHeight + zSurfaceNorm(1);

maxHinterest = 330 +antHeight;

simulationType = cell(4,1);

% Plot the surface:
surfacePlot = figure();
plot(xVect,zSurfaceNorm)
xlabel('Distance [m]');
ylabel('Surface height [m]');
titleFig = ['UpwardsInclinedPlane_results/UpwardsInclinedPlane_surface.png'];
saveas(surfacePlot,titleFig,'png');


% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
```

```
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep = SSAirregularTerrainAbsoptionLayer(...
    initialFieldFDM,zVectFDM,...
    xVect,zSurfaceNorm,HindexFDM,frequency,...
    numPtsAbsoptionLayer,antHeight);
toc
tic
uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
    (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
deltaZstr = num2str(deltaZ);
yText = strcat('Height above the lowest point [m]');

%maxZ = 250;

% Extracting the vertical values:
eFieldSSA  = verticalVector(getVerticalValues(xVect,...
    zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesSplitStep,'up'))';

eFieldFDM = verticalVector(getVerticalValues(xVect,...
    zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDist,maxZcompare,uValuesFDM,'up'))';

simulationType{1,1} = ['SSA inclined'];

simulationType{2,1} = ['FDM inclined'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);

minPlotLevel = 10^(-4);
for i = 1:1
% Plot SSA figuure:
plotScale =length(zVectFDM); % ceil(2*length(zVectFDM)/3)
fig2 = figure('visible','off');
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};
tic
uValuesAux = uValuesSplitStep(1:plotScale,:);
```

```
uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
disp('contourf is on')
contourf(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
disp('contour is on')
contour(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
eFieldSSA  = verticalVector(getVerticalValues(xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ
    xVect,xDist,maxZcompare,uValuesSplitStep,'up'))';

%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on

titleFig = ['UpwardsInclinedPlane_results/SSA_field_InclUp.png'];
disp('saving ...')
saveas(fig2,titleFig,'png');
toc

tic
% Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDM(1:plotScale,:);
uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
contourf(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM(1:plotScale,1),...
    10.*log10(abs(uValuesAux.^2)),50)
hold on
eFieldFDM = verticalVector(getVerticalValues(xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,
    xVect,xDist,maxZcompare,uValuesFDM,'up'))';
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['UpwardsInclinedPlane_results/FDM_field_InclUp.png'];
saveas(fig,titleFig,'png');
```

```
toc

end

% Comparing with results from Indra (along the surface),
% horizontal comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = xVect;
rx_zArr = ones(length(xVect),1).*antHeight;

plotTitle = ' ';

titleFig = ['UpwardsInclinedPlane_results/SSA_FDM_InclUp.png'];
filename = 'IndraSource/wedge1/LPDA-u-kile_E.xls';

rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;

simulationType1 = cell(2,1);
simulationType1{1,1} = 'SSA inclined';
simulationType1{2,1} = 'FDM inclined';

pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eFieldTot,...
    antHeight,beta,frequency,simulationType1, plotTitle,titleFig,filename)
%zDist = 250;

% Calculating the field along the flat surface
antHeight = antHeight - zSurfaceNorm(1);
theta0 = 0;
%maxHeight = 250;
xVect = verticalVector([0:deltaX:xDist]);
% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep_flat =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
tic
[uValuesFDMalne_flat,maxEigVal]=...
    FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
```

```
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc

% Extracting field values for vertical comparison:
eFieldSSA = zeros(1,maxZcompare+1);
eFieldFDM = zeros(1,maxZcompare+1);
for i = 1:maxZcompare+1
    eFieldSSA(1,i) = uValuesSplitStep_flat(i,...
        length(xVect));
    eFieldFDM(1,i) = uValuesFDMalne_flat(i,...
        length(xVect));
end


simulationType{3,1} = ['SSA flat'];
simulationType{4,1} = ['FDM flat'];
eFieldTot = vertcat(eFieldTot,eFieldSSA);
eFieldTot = vertcat(eFieldTot,eFieldFDM);

% Comparing with results from Indra (along the surface),
% horizontal comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;
part1Title = ' ';
plotTitle = ' ';

titleFig_comp =['UpwardsInclinedPlane_results/SSA_FDM_compare_InclUp.png'];
filename = 'IndraWedge_results/LPDA-u-kile_E.xls';
numCases = 4;
startIndex = 4;
numElts = length(zVectFDM);

eFieldTot1 = zeros(numCases,length(eFieldTot(1,:)));
for i = 1:numCases
   eFieldTot1(i,:) = eFieldTot(i,:);
end

compareHeight = 50;
 titleFig3 =['UpwardsInclinedPlane_results/',...
     'SSA_FDM_flat_height_varying_InclUp_zoomed1.png'];
  pathLossFlat_Indra_minComp(rx_xArr,rx_zArr,eFieldTot1,...
     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)

 titleFig4 =['UpwardsInclinedPlane_results/',...
     'SSA_FDM_flat_height_varying_InclUp_zoomed2.png'];
  pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
```

```
    frequency,simulationType, plotTitle,titleFig4,filename,compareHeight)

compareHeight = 240;

 pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
   frequency,simulationType, plotTitle,titleFig_comp,filename,compareHeight)

% Plot SSA figuure:
yText = strcat('Height above surface [m]');
fig2 = figure('visible','off');
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesSplitStep_flat;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['UpwardsInclinedPlane_results/SSA_flat_InclUp.png'];
saveas(fig2,titleFig,'png');

% Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDMalne_flat;
uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['UpwardsInclinedPlane_results/FDM_flat_InclUp.png'];
saveas(fig,titleFig,'png');


% Compare along the surface at constant height of 40 m above the lowest
% point:

%eFieldAlongTot
height = 40;
```

```
startIndex = 101; %100;
eSSArunway = (uValuesSplitStep(height,startIndex:length(xVect)));
eFieldAlongTot  =  eSSArunway;
eFDMrunway = (uValuesFDM(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMrunway);

eSSAflat = (uValuesSplitStep_flat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eSSAflat);
eFDMflat = (uValuesFDMalne_flat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMflat);

imulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - upwards'];
simulationType2{2,1} = ['FDM - upwards'];
simulationType2{3,1} = ['SSA - flat'];
simulationType2{4,1} = ['FDM - flat'];

rx_xArr3 = xVect(startIndex:length(xVect));
rx_zArr3 =  ones(length(rx_xArr3),1).*height;
titleFig3 = ['UpwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal_InclUp.png'];
antHeight2 = antHeight - zSurfaceNorm(1);

titleFig4 = ['UpwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal2_InclUp.png'];

pathLossIndra_alongX(rx_xArr3,rx_zArr3,eFieldAlongTot,...
    frequency,simulationType2,' ',titleFig4,' ')


% Comparing fields along the surface at constant height above the surface
height2 = 15;
startIndex2 = 101;

fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
eSSArunway2 = zeros(length(fieldVect2),1);
eFDMrunway2 = zeros(length(fieldVect2),1);
for i = 1:length(fieldVect2)
    %a = zSurfaceNorm(i);
    eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
    eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
end

eFieldAlongTot2  =  eSSArunway2';
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMrunway2');
eSSAflat2 = (uValuesSplitStep_flat(height,startIndex2:length(xVect)));
```

```
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
eFDMflat2 = (uValuesFDMalne_flat(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);

titleFig5 = ['UpwardsInclinedPlane_results/',...
    'SSA_FDM_pathloss_horizontal_cst_diff_surface_InclUp.png'];
rx_xArr4 = xVect(startIndex2:length(xVect));
rx_zArr4 =  ones(length(rx_xArr4),1).*height;


pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2,...
    frequency,simulationType2,' ',titleFig5,' ')
%get(gcf,'CurrentAxes')
ax1 = gca;

% Plotting the surface with logarithmic axes:
fig = figure();

 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
     (startIndex2:length(xVect))); %,'Parent',ax2);
% linkaxes([ax1 ax2],'y');
% linkaxes([ax1 ax2],'x');
legend('Surface profile','Location','SouthEast');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 200;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['UpwardsInclinedPlane_results/',...
    'Runway_profile_small_InclUp.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);
```

E.1.7 WedgeComparison_Hviid

Script for generation of the results for the wedge given by the Hviid et al. [1995] article, section 5.3.

```
% WedgeComparison_Hviid.m: Simulates the field over the wedge given in the
%                          Hviid article using the SSA and FDM, and
%                          compares with the results over a flat surface.
```

```
clear all
% The inclined plane:
beta = pi/18;
A = 1;
frequency = 100*10^6;
theta0 = 0;

% The delta x and delta z value:
deltaX = 1;
deltaZ = 1;

antHeight = 3;
maxX = 5000;
halfWay = maxX/2;
maxZ = 50;
numPtsAbsoptionLayer = 250;

% The irregular terrain:
Xn = [0 halfWay maxX];
Zn = [0 maxZ 0];
maxZcompare = 500;

% Interpolating the surface:
[xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
    'linear',deltaX);

maxHinterest = 650+antHeight;

simulationType = cell(4,1);

%Plot the surface:
surfacePlot = figure();
plot(xVect,zSurfaceVect,'k')
xlabel('Distance [m]');
ylabel('Surface height [m]');
plotwidth = 560;
plotheight = 200;
set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
titleFig = ['WedgeComparison_Hviid_Results/WedgeHviid_surface.png'];
saveas(surfacePlot,titleFig,'png');
titleFig = titleFig(1:(length(titleFig)-4));
titleFig = horzcat(titleFig,'.pdf');
save2pdf(titleFig);
```

```
% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep = SSAirregularTerrainAbsoptionLayer(...
    initialFieldFDM,zVectFDM,...
    xVect,zSurfaceVect,HindexFDM,frequency,...
    numPtsAbsoptionLayer,antHeight);
toc
tic
uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
    (initialFieldFDM,zVectFDM,xVect,zSurfaceVect,...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
deltaZstr = num2str(deltaZ);
yText = strcat('Height above the lowest point [m]');


% Extracting the field at distance of interest:
 xCoord =  find(xVect >= maxX ,1);
 eFieldSSA = zeros(1,((maxZcompare +1)/deltaZ));
 eFieldFDM = zeros(1,((maxZcompare +1)/deltaZ));
 for i = 1:(maxZcompare +1)
     eFieldSSA(1,i) = uValuesSplitStep(i,xCoord);
     eFieldFDM(1,i) = uValuesFDM(i,xCoord);
 end
simulationType{1,1} = ['SSA wedge'];

simulationType{2,1} = ['FDM wedge'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);




minPlotLevel = 10^(-7);
for i = 1:1
    % Plot SSA figuure:
     fig2 = figure('visible','off');

```

```
    plotScale =length(zVectFDM);

    part1Title = ['Split-Step Algorithm - flat surface'];
    titleVal2 = {part1Title};
    tic
    uValuesAux = uValuesSplitStep(1:plotScale,:);
    uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
    minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
    maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
    disp('contourf is on')
    contourf(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on
    disp('contour is on')
    contour(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on

    %title(titleVal2)
    xlabel('Distance [m]');
    ylabel(yText);
    grid on

    titleFig = ['WedgeComparison_Hviid_Results/SSA_field_freq_',...
        num2str(frequency/(10^6)),'_HviidWedge.png'];
    disp('saving ...')
    saveas(fig2,titleFig,'png');
    titleFig = titleFig(1:(length(titleFig)-4));
    titleFig = horzcat(titleFig,'.pdf');
    save2pdf(titleFig);
    toc

    tic
    % Plot FDM figure:
    fig = figure('visible','off');
    part1Title = ['Finite-Difference Method - flat surface'];
    titleVal2 = {part1Title};

    uValuesAux = uValuesFDM(1:plotScale,:);
    uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
    minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
    maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
    contourf(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on
    contour(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    %title(titleVal2)
```

```
    hold on
    xlabel('Distance [m]');
    ylabel(yText);
    grid on
    titleFig = ['WedgeComparison_Hviid_Results/FDM_field_freq_',...
        num2str(frequency/(10^6)),'_HviidWedge.png'];
    saveas(fig,titleFig,'png');
    titleFig = titleFig(1:(length(titleFig)-4));
    titleFig = horzcat(titleFig,'.pdf');
    save2pdf(titleFig);
    toc

end

% Zoomed plots:
for i = 1:1
    % Plot SSA figuure:
     fig2 = figure('visible','off');

    plotScale =ceil(0.35*length(zVectFDM)); % ceil(2*length(zVectFDM)/3)

    part1Title = ['Split-Step Algorithm - flat surface'];
    titleVal2 = {part1Title};
    tic
    uValuesAux = uValuesSplitStep(1:plotScale,:);
    uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
    minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
    maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
    disp('contourf is on')
    contourf(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on
    disp('contour is on')
    contour(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on

    %title(titleVal2)
    xlabel('Distance [m]');
    ylabel(yText);
    grid on

    titleFig = ['WedgeComparison_Hviid_Results/SSA_field_freq_',...
        num2str(frequency/(10^6)),'_HviidWedge_zoomed.png'];
    disp('saving ...')
    saveas(fig2,titleFig,'png');
    titleFig = titleFig(1:(length(titleFig)-4));
    titleFig = horzcat(titleFig,'.pdf');
```

```
    save2pdf(titleFig);
    toc

    tic
    % Plot FDM figure:
    fig = figure('visible','off');
    part1Title = ['Finite—Difference Method — flat surface'];
    titleVal2 = {part1Title};

    uValuesAux = uValuesFDM(1:plotScale,:);
    uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
    minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
    maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
    contourf(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    hold on
    contour(xVect,zVectFDM(1:plotScale,1),...
        10.*log10(abs(uValuesAux.^2)),50)
    %title(titleVal2)
    hold on
    xlabel('Distance [m]');
    ylabel(yText);
    grid on
    titleFig = ['WedgeComparison_Hviid_Results/FDM_field_freq_',...
        num2str(frequency/(10^6)),'_HviidWedge_zoomed.png'];
    saveas(fig,titleFig,'png');
    titleFig = titleFig(1:(length(titleFig)—4));
    titleFig = horzcat(titleFig,'.pdf');
    save2pdf(titleFig);
    toc

end

% Comparing with results from Indra (along the surface):
tx =antHeight;
rx = antHeight;
rx_xArr = xVect;
rx_zArr = ones(length(xVect),1).*antHeight;
plotTitle = ' ';
titleFig = ['WedgeComparison_Hviid_Results/SSA_FDM_freq_',...
        num2str(frequency/(10^6)),'_HviidWedge.png'];
filename = 'IndraWedge2_results/LPDA—u—kile—2_E.xls';

rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;

simulationType1 = cell(2,1);
simulationType1{1,1} = 'SSA';
```

```
simulationType1{2,1} = 'FDM';
compareHeight_Hviid  = 250;
close all
initCompare = initialFieldFDM(1:maxZcompare +1);
eTest = zeros(4,length(initCompare));

for i = 1:4
    eTest(1,:) = initCompare'./eFieldTot(1,:);
end

pathLossWedge_Hviid(antHeight,rx_xArr,rx_zArr,eFieldTot,...
    frequency,simulationType1, plotTitle,titleFig,' ',compareHeight_Hviid)

% Calculating the field along the flat surface
theta0 = 0;
Xn_flat = [0 maxX];
Zn_flat = [0 0];

% Creating the field (flat surface);
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating the fields (flat surface):
tic
uValuesSplitStepFlat =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
tic
[uValuesFDMalneFlat,maxEigVal]=...
    FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc

clear eFieldSSA;
clear eFieldFDM;

% Extracting the field for vertical comparison:
eFieldSSA = zeros(1,length(eFieldTot(1,:)));
eFieldFDM = zeros(1,length(eFieldTot(1,:)));
for i = 1:length(eFieldSSA)
    eFieldSSA(1,i) = uValuesSplitStepFlat(i,...
        length(xVect));
    eFieldFDM(1,i) = uValuesFDMalneFlat(i,...
        length(xVect));
```

```

end

simulationType{3,1} = ['SSA flat'];
simulationType{4,1} = ['FDM flat'];
eFieldTot = vertcat(eFieldTot,eFieldSSA);
eFieldTot = vertcat(eFieldTot,eFieldFDM);

% Comparing with results from Indra, vertical comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = zVectFDM;
part1Title = ' ';%['SSA and FDM - flat surface - along the surface'];
plotTitle = ' '; % {part1Title};

titleFig = [...
    'WedgeComparison_Hviid_Results/SSA_FDM_compareFlat_HviidWedge.png'];
filename = 'IndraWedge_results/LPDA-u-kile_E.xls';
numCases = 4;
startIndex = 4;
numElts = length(eFieldTot(1,:));
eFieldTot1 =eFieldTot;

compareHeight = 500; % maximum height of comparison
pathLossWedge_Hviid(antHeight,rx_xArr,rx_zArr,eFieldTot1,...
    frequency,simulationType, plotTitle,titleFig,' ',compareHeight)

compareHeight = 50; % new maximum height of comparison
 titleFig3 =['WedgeComparison_Hviid_Results/',...
     'SSA_FDM_flat_height_varying_HviidWedge_zoomed1.png'];
  pathLossFlat_Indra_minComp(rx_xArr,rx_zArr,eFieldTot1,...
     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)

 titleFig4 =['WedgeComparison_Hviid_Results/',...
     'SSA_FDM_flat_height_varying_HviidWedge_zoomed2.png'];
  pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
     frequency,simulationType, plotTitle,titleFig4,filename,compareHeight)

 % Comparing along the surface, horizontal comparison:

simulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - wedge'];
simulationType2{2,1} = ['FDM - wedge'];
simulationType2{3,1} = ['SSA - flat'];
simulationType2{4,1} = ['FDM - flat'];
%eFieldAlongTot
height = 65;
```

```
startIndex = 121;
eSSAwedge = (uValuesSplitStep(height,startIndex:length(xVect)));
eFieldAlongTot  =  eSSAwedge;
eFDMwedge = (uValuesFDM(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMwedge);

eSSAflat = (uValuesSplitStepFlat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eSSAflat);
eFDMflat = (uValuesFDMalneFlat(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMflat);

rx_xArr3 = xVect(startIndex:length(xVect));
rx_zArr3 =  ones(length(rx_xArr3),1).*height;
titleFig3 = ['WedgeComparison_Hviid_Results/',...
    'SSA_FDM_pathloss_horizontal_HviidWedge.png'];

pathLossWedge_Hviid(antHeight,rx_zArr3,rx_xArr3,eFieldAlongTot,...
    frequency,simulationType2, ' ',titleFig3,' ',0)
titleFig4 = ['WedgeComparison_Hviid_Results/',...
    'SSA_FDM_pathloss_horizontal2_HviidWedge.png'];

pathLossIndra_alongX(rx_xArr3,rx_zArr3,eFieldAlongTot,...
    frequency,simulationType2,' ',titleFig4,' ')


% Comparing fields along the surface at constant height above the surface
height2 = 15;
startIndex2 = 121;
fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
eSSAwedge2 = zeros(length(fieldVect2),1);
eFDMwedge2 = zeros(length(fieldVect2),1);
for i = 1:length(fieldVect2)
    eSSAwedge2(i,1) = uValuesSplitStep(round(zSurfaceVect(i)+height), ...
        fieldVect2(i));
    eFDMwedge2(i,1) = uValuesFDM(round(zSurfaceVect(i)+height), ...
        fieldVect2(i));
end

eFieldAlongTot2  =  eSSAwedge2';
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMwedge2');
eSSAflat2 = (uValuesSplitStepFlat(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
eFDMflat2 = (uValuesFDMalneFlat(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);

titleFig5 = ['WedgeComparison_Hviid_Results/',...
    'SSA_FDM_pathloss_horizontal_cst_diff_surface_HviidWedge.png'];
```

```
rx_xArr4 = xVect(startIndex2:length(xVect));
rx_zArr4 =  ones(length(rx_xArr4),1).*height;


pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2,...
    frequency,simulationType2,' ',titleFig5,' ')
%get(gcf,'CurrentAxes')
ax1 = gca;

% Plotting the wedge surface with logarithmic scale:
fig = figure();

semilogx((xVect(startIndex2:length(xVect))),zSurfaceVect...
     (startIndex2:length(xVect)));

legend('Wedge surface','Location','NorthWest');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 200;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['WedgeComparison_Hviid_Results/',...
    'Runway_profile_small_HviidWedge.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);
```

E.1.8 Braunschweig

Script for generation of the results over the Braunschweig runway, section 5.4.1.

```
% Braunschweig.m: Script simulating the electric field over the
%                 Braunschweig airport.
clear all
close all

% Setting the parameters:
theta0 = 0;
beta = pi/18;
A = 1;
frequency = 110*10^6;

```

```
deltaX = 1;
maxX = 3000;
numPtsAbsoptionLayer = 150;

deltaZ = 1;
antHeight = 3;
deltaXvect = [1];

counter = 0;
doubleCounter = 0;


maxHinterestHeight = 10;
numElts = 2;
simulationType = cell(numElts,1);

% Get the terrain profile:
xColumn = 2;
zColumn = 3;
fileName = '\IndraSource\braunschweig\Model profile.xls';

% Importing, interpolating and shifting the surface:
[Xn,Zn]=importParametersFromFile(fileName,xColumn,zColumn);
[xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
    'curve',deltaX);
[zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);


antHeight = antHeight + zSurfaceNorm(1);

%xVect = verticalVector([0:deltaX:maxX]);
maxHinterest = 350 +antHeight;

% Plot the surface:
surfacePlot = figure();
plot(xVect,zSurfaceNorm)
xlabel('Distance [m]');
ylabel('Surface height [m]');
plotwidth = 560;
plotheight = 200;
set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
legend('Runway surface profile','Location','SouthEast');
titleFig = ['Braunschweig_results/Runway_Braunschweig.png'];
saveas(surfacePlot,titleFig,'png');
titleFig = titleFig(1:(length(titleFig)-4));
titleFig = horzcat(titleFig,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig);
```

```

% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep = SSAirregularTerrainAbsoptionLayer(...
    initialFieldFDM,zVectFDM,...
    xVect,zSurfaceNorm,HindexFDM,frequency,...
    numPtsAbsoptionLayer,antHeight);
toc
tic
uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
    (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
deltaZstr = num2str(deltaZ);
yText = strcat('Height above the lowest point [m]');

SurfEndHeight = ceil(zSurfaceNorm(length(zSurfaceNorm)));
fieldVect = [ceil(SurfEndHeight/deltaZ):1:floor(length(zVectFDM)/deltaZ)];

% Extracting field values for vertical comparison:
eFieldSSA = zeros(1,length(fieldVect));
eFieldFDM = zeros(1,length(fieldVect));
for i = 1:length(fieldVect)
    eFieldSSA(1,i) = uValuesSplitStep(fieldVect(i),...
        length(xVect));
    eFieldFDM(1,i) = uValuesFDM(fieldVect(i),...
        length(xVect));
end

simulationType{1,1} = ['SSA'];

simulationType{2,1} = ['FDM'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);


% Plot SSA figuure:
fig2 = figure('visible','off');
```

```
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesSplitStep;
uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['Braunschweig_results/SSA_flat_along_surface_Braunschweig.png'];
saveas(fig2,titleFig,'png');

% Plot FDM figure:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDM;
uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig =['Braunschweig_results/FDM_flat_along_surface_Braunschweig.png'];
saveas(fig,titleFig,'png');

% Comparing with results from Indra, vertical comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = fieldVect; % zVectFDM;
part1Title = ' ';%['SSA and FDM - flat surface - along the surface'];
plotTitle = {part1Title};
titleFig = [...
    'Braunschweig_results/SSA_FDM_flat_along_surface_Braunschweig.png'];
filename = 'IndraWedge2_results/LPDA-u-kile-2_E.xls';

pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eFieldTot,...
    antHeight,beta,frequency,simulationType, plotTitle,titleFig,' ')

% Calculating field over flat surface:
tic
```

```
uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc

tic
[uValuesFDM2,maxEigVal]=FDMAbsorptionLayerNumEfficient2(initialFieldFDM,...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc

% Extracting field values for vertical comparison:
eFieldSSA2 = zeros(1,length(fieldVect));
eFieldFDM2 = zeros(1,length(fieldVect));
for i = 1:length(fieldVect)
    eFieldSSA2(1,i) = uValuesSSA2(fieldVect(i),...
        length(xVect));
    eFieldFDM2(1,i) = uValuesFDM2(fieldVect(i),...
        length(xVect));
end


eFieldTot = vertcat(eFieldTot,eFieldSSA2);
eFieldTot = vertcat(eFieldTot,eFieldFDM2);

rx_xArr2 = ones(length(zVectFDM),1).*length(xVect);
rx_zArr2 =  fieldVect;

titleFig2 = ['Braunschweig_results/',...
    'SSA_FDM_pathloss_vertical_Braunschweig.png'];
filename2 = 'IndraWedge_results/LPDA-u-kile_E.xls';
simulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - runway'];
simulationType2{2,1} = ['FDM - runway'];
simulationType2{3,1} = ['SSA - flat'];
simulationType2{4,1} = ['FDM - flat'];
compareHeight = 200;

compareHeight_Hviid = 350;

% Vertical comparison
antHeight2 = antHeight - zSurfaceNorm(1);
pathLossWedge_Hviid(antHeight2,rx_xArr2,rx_zArr2,eFieldTot,...
    frequency,simulationType2, ' ',titleFig2,' ',compareHeight_Hviid)


% Compare along the surface at constant height of 40 m above the lowest
% point:

```

```
%eFieldAlongTot
height = 40;
startIndex = 101; %100;
eSSArunway = (uValuesSplitStep(height,startIndex:length(xVect)));
eFieldAlongTot  =  eSSArunway;
eFDMrunway = (uValuesFDM(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMrunway);

eSSAflat = (uValuesSSA2(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eSSAflat);
eFDMflat = (uValuesFDM2(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMflat);

rx_xArr3 = xVect(startIndex:length(xVect));
rx_zArr3 =  ones(length(rx_xArr3),1).*height;
titleFig3 = ['Braunschweig_results/',...
    'SSA_FDM_pathloss_horizontal_Braunschweig.png'];

pathLossWedge_Hviid(antHeight2,rx_zArr3,rx_xArr3,eFieldAlongTot,...
    frequency,simulationType2, ' ',titleFig3,' ',0)
titleFig4 = ['Braunschweig_results/',...
    'SSA_FDM_pathloss_horizontal2_Braunschweig.png'];

pathLossIndra_alongX(rx_xArr3,rx_zArr3,eFieldAlongTot,...
    frequency,simulationType2,' ',titleFig4,' ')


% Comparing fields along the surface at constant height above the surface
height2 = 15;
startIndex2 = 101;
fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
eSSArunway2 = zeros(length(fieldVect2),1);
eFDMrunway2 = zeros(length(fieldVect2),1);
for i = 1:length(fieldVect2)
    %a = zSurfaceNorm(i);
    eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
    eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
end

close all

eFieldAlongTot2   =   eSSArunway2';
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMrunway2');
eSSAflat2 = (uValuesSSA2(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
eFDMflat2 = (uValuesFDM2(height,startIndex2:length(xVect)));
```

```
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);

eFieldAlongTot2Inv = eFieldAlongTot2;

for i =0:3
    eFieldAlongTot2Inv(i+1,:) = eFieldAlongTot2(4-i,:);
end
simulationTypeInv = cell(4,1);
simulationTypeInv{4,1} = ['SSA - runway'];
simulationTypeInv{3,1} = ['FDM - runway'];
simulationTypeInv{2,1} = ['SSA - flat'];
simulationTypeInv{1,1} = ['FDM - flat'];

titleFig5 = ['Braunschweig_results/',...
    'SSA_FDM_pathloss_horizontal_cst_diff_surface_Braunschweig.png'];
rx_xArr4 = xVect(startIndex2:length(xVect));
rx_zArr4 =  ones(length(rx_xArr4),1).*height;

% Horizontal comparison:
pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2Inv,...
    frequency,simulationTypeInv,' ',titleFig5,' ')

ax1 = gca;

% Plotting the surface profile with logaritmic scale:
fig = figure();

 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
     (startIndex2:length(xVect))); %,'Parent',ax2);

legend('Runway surface profile','Location','South');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 200;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['Braunschweig_results/',...
    'Runway_profile_small_Braunschweig.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);


% Ploting the line for field observation:
fig = figure();
```

```
obsLine = zSurfaceNorm +(15/deltaX);
plot(xVect,obsLine,'--k');
hold on
plot(xVect,zSurfaceNorm,'k');
hold on
X_slope = [0 1500];
Z_slope = [4 29];
plot(X_slope,Z_slope);
legendName = cell(3,1);
legendName{1,1} = 'Observation points';
legendName{2,1} = 'Runway surface profile';
legendName{3,1} = 'Line-of-sight line';
legend(legendName,'Location','SouthEast');
%legend('Runway surface profile','Location','SouthWest');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 300;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['Braunschweig_results/',...
    'Runway_profile_lineOfSight_Braunschweig.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);
```

E.1.9 Luton2

Script for generation of the results over the Luton runway, section 5.4.2.

```
% Luton2.m: Script simulating and comparing  the electric field over the
%            Luton airport.
clear all

% Setting the parameters:
theta0 = 0;
beta = pi/18;
A = 1;
frequency = 110*10^6;

deltaX = 1;
maxX = 3000;
numPtsAbsoptionLayer = 150;
```

```
deltaZ = 1;
antHeight = 3;
deltaXvect = [1];

counter = 0;
doubleCounter = 0;
maxHinterestHeight = 10;
numElts = 2;
simulationType = cell(numElts,1);

% Get the terrain profile:
xColumn = 1;
zColumn = 3;
fileName ='\IndraSource\luton\Runway-profile_LOC26.xls'; %
%'IndraSource/luton/Runway-profile_LOC26.xls';

% Importing, interpolating, and shifting the surface height:
[Xn,Zn]=importParametersFromFile(fileName,xColumn,zColumn);

[xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
    'curve',deltaX);
[zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);

antHeight = antHeight + zSurfaceNorm(1);

maxHinterest = 350 +antHeight;

% Plot the surface:
surfacePlot = figure();
plot(xVect,zSurfaceNorm)
xlabel('Distance [m]');
ylabel('Surface height [m]');
plotwidth = 560;
plotheight = 200;
set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
legend('Runway surface profile','Location','SouthWest');
titleFig = ['Luton_results/Runway_Luton.png'];
saveas(surfacePlot,titleFig,'png');
titleFig = titleFig(1:(length(titleFig)-4));
titleFig = horzcat(titleFig,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig);

% Creating initial field:
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM = createInitialField(antHeight,theta0,beta,...
```

```
    zVectFDM,A, frequency,'gaussian1');
numZpoints = length(zVectFDM);

% Calculating field:
tic
uValuesSplitStep = SSAirregularTerrainAbsoptionLayer(...
    initialFieldFDM,zVectFDM,...
    xVect,zSurfaceNorm,HindexFDM,frequency,...
    numPtsAbsoptionLayer,antHeight);
toc
tic
uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
    (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc
deltaZstr = num2str(deltaZ);
yText = strcat('Height above the lowest point [m]');

SurfEndHeight = ceil(zSurfaceNorm(length(zSurfaceNorm)));
fieldVect =[...
    ceil(SurfEndHeight/deltaZ):1:floor(length(zVectFDM)/deltaZ)-1]+1;

% Extracting the field values for vertical comparison:
eFieldSSA = zeros(1,length(fieldVect));
eFieldFDM = zeros(1,length(fieldVect));
for i = 1:length(fieldVect)
    eFieldSSA(1,i) = uValuesSplitStep(fieldVect(i),...
        length(xVect));
    eFieldFDM(1,i) = uValuesFDM(fieldVect(i),...
        length(xVect));
end

simulationType{1,1} = ['SSA'];
simulationType{2,1} = ['FDM'];
tx = antHeight;
rx = antHeight;

eFieldTot = eFieldSSA;
eFieldTot = vertcat(eFieldTot,eFieldFDM);


% Plot SSA field:
fig2 = figure('visible','off');
part1Title = ['Split-Step Algorithm - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesSplitStep;
uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
```

```
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['Luton_results/SSA_flat_along_surface_Luton.png'];
saveas(fig2,titleFig,'png');

% Plot FDM field:
fig = figure('visible','off');
part1Title = ['Finite-Difference Method - flat surface'];
titleVal2 = {part1Title};

uValuesAux = uValuesFDM;
uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
hold on
contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
%title(titleVal2)
xlabel('Distance [m]');
ylabel(yText);
grid on
titleFig = ['Luton_results/FDM_flat_along_surface_Luton.png'];
saveas(fig,titleFig,'png');

% Comparing with results from Indra, vertical comparison:
tx =antHeight;
rx = antHeight;
rx_xArr = ones(length(zVectFDM),1).*length(xVect);
rx_zArr = fieldVect; % zVectFDM;
part1Title = ' ';%['SSA and FDM - flat surface - along the surface'];
plotTitle = {part1Title};
titleFig = ['Luton_results/SSA_FDM_flat_along_surface_Luton.png'];
filename = 'IndraWedge2_results/LPDA-u-kile-2_E.xls';

pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eFieldTot,...
    antHeight,beta,frequency,simulationType, plotTitle,titleFig,' ')

% Comparing with results from a flat surface
antHeight2 = antHeight - zSurfaceNorm(1);
[zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
    maxHinterest,deltaZ,numPtsAbsoptionLayer);
initialFieldFDM_flat = createInitialField(antHeight2,theta0,beta,...
    zVectFDM,A, frequency,'gaussian1');

% Calculating field over flat surface:
```

```
tic
uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
    initialFieldFDM_flat,zVectFDM,xVect, ...
    HindexFDM,frequency,numPtsAbsoptionLayer);
toc

tic
[uValuesFDM2,maxEigVal]=FDMAbsorptionLayerNumEfficient2(...
    initialFieldFDM_flat,...
    zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
toc

% Extracting field values for vertical comparison:
eFieldSSA2 = zeros(1,length(fieldVect));
eFieldFDM2 = zeros(1,length(fieldVect));
for i = 1:length(fieldVect)
    eFieldSSA2(1,i) = uValuesSSA2(fieldVect(i),...
        length(xVect));
    eFieldFDM2(1,i) = uValuesFDM2(fieldVect(i),...
        length(xVect));
end


eFieldTot = vertcat(eFieldTot,eFieldSSA2);
eFieldTot = vertcat(eFieldTot,eFieldFDM2);

rx_xArr2 = ones(length(zVectFDM),1).*length(xVect);
rx_zArr2 =  fieldVect;

titleFig2 = ['Luton_results/',...
    'SSA_FDM_pathloss_vertical_Luton.png'];
filename2 = 'IndraWedge_results/LPDA-u-kile_E.xls';
simulationType2 = cell(4,1);
simulationType2{1,1} = ['SSA - runway'];
simulationType2{2,1} = ['FDM - runway'];
simulationType2{3,1} = ['SSA - flat'];
simulationType2{4,1} = ['FDM - flat'];
compareHeight = 200;

compareHeight_Hviid = 350;

antHeight2 = antHeight - zSurfaceNorm(1);
% Vertical comparison:
pathLossWedge_Hviid(antHeight2,rx_xArr2,rx_zArr2,eFieldTot,...
    frequency,simulationType2, ' ',titleFig2,' ',compareHeight_Hviid)


% Compare along the surface at constant heigth of 40 m above the lowest
```

```
% point:

height = 40;
startIndex = 101;
eSSArunway = (uValuesSplitStep(height,startIndex:length(xVect)));
eFieldAlongTot  =  eSSArunway;
eFDMrunway = (uValuesFDM(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMrunway);

eSSAflat = (uValuesSSA2(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eSSAflat);
eFDMflat = (uValuesFDM2(height,startIndex:length(xVect)));
eFieldAlongTot = vertcat(eFieldAlongTot,eFDMflat);

rx_xArr3 = xVect(startIndex:length(xVect));
rx_zArr3 =  ones(length(rx_xArr3),1).*height;
titleFig3 = ['Luton_results/',...
    'SSA_FDM_pathloss_horizontal_Luton.png'];

pathLossWedge_Hviid(antHeight2,rx_zArr3,rx_xArr3,eFieldAlongTot,...
    frequency,simulationType2, ' ',titleFig3,' ',0)
titleFig4 = ['Luton_results/',...
    'SSA_FDM_pathloss_horizontal2_Luton.png'];

pathLossIndra_alongX(rx_xArr3,rx_zArr3,eFieldAlongTot,...
    frequency,simulationType2,' ',titleFig4,' ')


% Comparing fields along the surface at constant height above the surface
height2 = 10;
startIndex2 = 101;
fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
eSSArunway2 = zeros(length(fieldVect2),1);
eFDMrunway2 = zeros(length(fieldVect2),1);
for i = 1:length(fieldVect2)
    %a = zSurfaceNorm(i);
    eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
    eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
        fieldVect2(i));
end

eFieldAlongTot2  =  eSSArunway2';
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMrunway2');
eSSAflat2 = (uValuesSSA2(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
eFDMflat2 = (uValuesFDM2(height,startIndex2:length(xVect)));
eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);
```

```
eFieldAlongTot2Inv = eFieldAlongTot2;

for i =0:3
    eFieldAlongTot2Inv(i+1,:) = eFieldAlongTot2(4-i,:);
end

titleFig5 = ['Luton_results/',...
    'SSA_FDM_pathloss_horizontal_cst_diff_surface_Luton.png'];
rx_xArr4 = xVect(startIndex2:length(xVect));
rx_zArr4 =  ones(length(rx_xArr4),1).*height;

simulationTypeInv = cell(4,1);
simulationTypeInv{4,1} = ['SSA - runway'];
simulationTypeInv{3,1} = ['FDM - runway'];
simulationTypeInv{2,1} = ['SSA - flat'];
simulationTypeInv{1,1} = ['FDM - flat'];

pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2Inv,...
    frequency,simulationTypeInv,' ',titleFig5,' ')

ax1 = gca;

% Plotting the surface with logarithmic scale:
fig = figure();

semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
     (startIndex2:length(xVect)));

legend('Runway surface profile','Location','SouthWest');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 200;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['Luton_results/',...
    'Runway_profile_small_Luton.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);

% Plotting the line for field observation:
fig = figure();
obsLine = zSurfaceNorm +(15/deltaX);
plot(xVect,obsLine,'--k');
```

```
hold on
plot(xVect,zSurfaceNorm,'k');
hold on
X_slope = [0 1500];
Z_slope = [13.525 32.007];
plot(X_slope,Z_slope);
legendName = cell(3,1);
legendName{1,1} = 'Observation points';
legendName{2,1} = 'Runway surface profile';
legendName{3,1} = 'Line-of-sight line';
legend(legendName,'Location','SouthWest');
xlabel('Distance [m]');
ylabel('Height [m]');
plotwidth = 560;
plotheight = 300;
set(fig, 'Position', [500 100 plotwidth plotheight]);
grid on
titleFig6 = ['Luton_results/',...
    'Runway_profile_lineOfSight_Luton.png'];
saveas(fig,titleFig,'png');
titleFig6 = titleFig6(1:(length(titleFig6)-4));
titleFig6 = horzcat(titleFig6,'.pdf');
%print (fig, '-dpdf', titleFig);
save2pdf(titleFig6);
```

E.2 Implemented Matlab Functions

E.2.1 Field Simulation Algorithms

E.2.1.1 FDMnoGround

Calculates field in free-space, using the FDM.

```

% FDMnoGrond.m : Calculating the electric field in free-space
%                using the finite-difference method.
%
% Assumption: Propagation in vacuum, no variations in the refractive index,
%             n(x,z) = 1 up to the max height of consideration, for all x
%             and z.

% zFieldInit: Vector containing the initial electric field along the z-axis
% zVect: Vector containing the z-coordinates of interest
```

```
% xVect: Vector containing the x-coordinates of interest
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation
% antennaIndex: The index of the center source point of the field in the
%               initital field.
%
% return: uValues: The u-values (electric field) for the entire
%                  computational domain.
%         maxEigVal: The maximum eigenvalue of the ''system'' matrix that
%                    is raisd to the n'th power.
%         antennaSourceIndex: The height index of the center point of the
%                             source for the output field.

function [uValues,maxEigVal,antennaSourceIndex] = FDMnoGround(zFieldInit, ...
    zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer,...
    antennaIndex)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
Hindex = maxHeigthInterestZIndex;
xVect= verticalVector(xVect);
zVect = verticalVector(zVect);

deltaZ = zVect(2,1)-zVect(1,1);
deltaZvect = zeros(numZpoints,1);
deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);

deltaX = xVect(2,1)-xVect(1,1);
deltaXvect = zeros(numXpoints,1);
deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);

% Flipping the Gaussian beam around the source center:
noGroundVect = zFieldInit(antennaIndex:numZpoints,1);
auxVect = flipud(noGroundVect(2:length(noGroundVect),1));
antennaSourceIndex = length(auxVect) +1;



% Creating absorption layer:
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsInAbsLayer);
```

```
indexRefraction = verticalVector(indexRefraction);

auxRefraction = indexRefraction(antennaIndex:numZpoints,1);
auxRefraction2 = flipud(auxRefraction(2:length(auxRefraction),1));
indexRefraction = vertcat(auxRefraction2,auxRefraction);

zFieldInit = vertcat(auxVect,noGroundVect);
numZpoints = length(indexRefraction);



% Finding the ''grid'' of a- and b-values:

% Vector a- and b-grid:

aGrid = zeros(numZpoints,1);
bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);


% Calculate the u-values for successive x-values:
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
onesVect = ones(numZpoints-1,1);

% Creates VmMat:
diagVmat = 2 + bGrid(1,1)- aGrid(:,1);
diagVmat(1,1) = 1;
diagVmat(numZpoints,1) = 1;
VmMat = (diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1)));
VmMat(1,2) = 0;
VmMat(numZpoints,numZpoints-1) = 0;

% Creates Amat:
diagAmat = -2 + bGrid(1,1)+ aGrid(:,1);
diagAmat(1,1) = 1;
diagAmat(numZpoints,1) = 1;
AmMat = (diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1));
AmMat(1,2) = 0;
AmMat(numZpoints,numZpoints-1) = 0;

% Create transition matrix:
transitionMat = (VmMat/AmMat);

% Diagonalization of the transition matrix:
%disp('Eigenvalues start');
%tic
[eigVect,eigValues] = eig(transitionMat);
```

```
%toc

maxEigVal=max(max(abs(eigValues)))

% Inversing eigVect using pseudo inverse:
eigVectInv = sparse(pinv(eigVect));
eigVect = sparse(eigVect);
eigValues = sparse(eigValues);

initEigInv = sparse(eigVectInv*zFieldInit);


auxEigValues = eigValues;
parfor x = 2:numXpoints
    auxEigValues = eigValues.^x;
    uGrid(:,x) = eigVect*auxEigValues*initEigInv;

end

uValues = uGrid;
end
```

E.2.1.2 splitStepAlgorithmAbsorptionLayer

Field simulation over a flat surface using the SSA.

```
% splitStepAlgorithmAbsorptionLayer.m: Calculates the electric field over a
%                                      flat surface using the split-step
%                                      algorithm. The algorithm inclueds an
%                                      absorption layer at the top.
%
% Remarks: length(xVect) >= length(zVect)

% zFieldInit: Vector containing the initial electric field along the z-axis
% zVect: Vector containing the z-coordinates of interest
% xVect: Vector containing the x-coordinates of interest
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation
% numPtsAbsoptionLayer: The number of height points in the absorption
%                       layer.

function uValues = splitStepAlgorithmAbsorptionLayer(zFieldInit,zVect, ...
```

```
    xVect,maxHeigthInterestZIndex,frequency,numPtsAbsoptionLayer)

c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
Hindex = maxHeigthInterestZIndex;
xVect = verticalVector(xVect);
zVect = verticalVector(zVect);


deltaZvect = zeros(numZpoints,1);
deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);

deltaXvect = zeros(numXpoints,1);
deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);

uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
indicesZminusOne = verticalVector(linspace(1,(numZpoints-1),...
    (numZpoints-1)));
Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
    .*deltaXvect(1:1,1)./(2.*k.*(numZpoints.^2)));


Pprime = verticalVector(Pprime);

% Creating the absorption layer:
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPtsAbsoptionLayer);

% Calculating the field:
for x = 2:numXpoints

    uGrid(1:(numZpoints-1),x) = inverseDiscreteSineTrans( ...
        Pprime.*discreteSineTrans(uGrid(1:(numZpoints-1),x-1)));
    uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 -1) ...
        .*deltaXvect(1,1)./2).*uGrid(1:numZpoints,x);
end
uValues = uGrid;

end
```

E.2.1.3 FDMAbsorptionLayerNumEfficient2

Field simulation over a flat surface using the FDM.

```

% FDMAbsorptionLayerNumEfficient2.m : Calculating the electric field over a
%                                     flat surface using the
%                                     finite-difference method.

% zFieldInit: Vector containing the initial electric field along the z-axis
% zVect: Vector containing the z-coordinates of interest
% xVect: Vector containing the x-coordinates of interest
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation
% numPointsInAbsLayer: The thickness of the absorption layer in the number
%                      of points.

% return: uValues: The simulated field.
%         maxEigVal: The maximum eigenvalue of the transistion matrix,
%                    gives the stability of the system.

function [uValues,maxEigVal] = FDMAbsorptionLayerNumEfficient2(zFieldInit, ...
    zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
Hindex = maxHeigthInterestZIndex;
xVect= verticalVector(xVect);
zVect = verticalVector(zVect);

deltaZ = zVect(2,1)-zVect(1,1);
deltaZvect = zeros(numZpoints,1);
deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);

deltaX = xVect(2,1)-xVect(1,1);
deltaXvect = zeros(numXpoints,1);
deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);

% Creating absorption layer:
```

```
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsInAbsLayer);

% Finding the ''grid'' of a- and b-values:
% Vector a- and b-grid:

aGrid = zeros(numZpoints,1);
bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);


% Creating grid for the uValues with the initial field:
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
onesVect = ones(numZpoints-1,1);

% Create Vmat:
diagVmat = 2 + bGrid(1,1)- aGrid(:,1);
diagVmat(1,1) = 1;
diagVmat(numZpoints,1) = 1;
VmMat = (diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1)));
VmMat(1,2) = 0;
VmMat(numZpoints,numZpoints-1) = 0;

% Create Amat:
diagAmat = -2 + bGrid(1,1)+ aGrid(:,1);
diagAmat(1,1) = 1;
diagAmat(numZpoints,1) = 1;
AmMat = (diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1));
AmMat(1,2) = 0;
AmMat(numZpoints,numZpoints-1) = 0;

% Creating transition marix:
transitionMat = (VmMat/AmMat);

% Diagonalization of the transition matrix:
[eigVect,eigValues] = eig(transitionMat);
maxEigVal=max(max(abs(eigValues)))

% Taking the pseudo-inverse of eigVect:
eigVectInv = sparse(pinv(eigVect));
eigVect = sparse(eigVect);
eigValues = sparse(eigValues);

initEigInv = sparse(eigVectInv*zFieldInit);

auxEigValues = eigValues;
% Calculating the field:
```

```
parfor x = 2:numXpoints
    auxEigValues = eigValues.^x;
    uGrid(:,x) = eigVect*auxEigValues*initEigInv;
end

uValues = uGrid;
end
```

E.2.1.4 SSA_addRloss

Field simulation over a flat surface using the SSA. Adds $\frac{1}{r}$-loss to the results.

```
% SSA_addRloss.m: Calculates the electric field (u-values) along a flat
%                 surface using the split-step algorithm. Additional loss
%                 is added, (1/r), to try to create a 3D model out of the
%                 2D model.
% Remarks: length(xVect) >= length(zVect)

% zFieldInit: Vector containing the initial electric field along the z-axis
% zVect: Vector containing the z-coordinates of interest
% xVect: Vector containing the x-coordinates of interest
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation
% numPtsAbsoptionLayer: The number of height points in the absorption
%                       layer.
% antHeight: Antenna height [m]

% return: uValues: calcultaed u-values with 1/r-loss added.


function uValues = SSA_addRloss(zFieldInit,zVect, ...
    xVect,maxHeigthInterestZIndex,frequency,numPtsAbsoptionLayer,...
    antHeight)

c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
Hindex = maxHeigthInterestZIndex;

xVect= verticalVector(xVect);
```

```
zVect = verticalVector(zVect);
deltaX = xVect(2,1)−xVect(1,1);

% Creating the a grid for the u−values
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
indicesZminusOne = verticalVector(linspace(1,(numZpoints−1),(numZpoints−1)));
Pprime = exp(−1j.*(pi.^2).*(indicesZminusOne.^2)...
    .*deltaX./(2.*k.*(numZpoints.^2)));
Pprime = verticalVector(Pprime);

% Creating absorption layer:
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPtsAbsoptionLayer);

% Calculating the field:
for x = 2:numXpoints
    uGrid(1:(numZpoints−1),x) = inverseDiscreteSineTrans( ...
        Pprime.*discreteSineTrans(uGrid(1:(numZpoints−1),x−1)));
    uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 −1) ...
        .*deltaX./2).*uGrid(1:numZpoints,x);

end

parfor x = 2:numXpoints
   distVect = sqrt(x.^2 + (zVect−antHeight).^2);
   % Adding additional loss (1/r):
    uGrid(:,x) = uGrid(:,x).*(1./distVect);
end
uValues = uGrid;

end
```

E.2.1.5 FDM_addRloss

Field simulation over a flat surface using the FDM. Adds $\frac{1}{r}$-loss to the results.

```

% FDM_addRloss.m : Calculates the electric field along a surface using the
%                  finite−difference method. Adds additional loss: (1/r),
%                  to compensate for the 2D model (''making'' it 3D). Works
%                  for a flat surface.
% zFieldInit: Vector containing the initial electric field along the z−axis
% zVect: Vector containing the z−coordinates of interest
% xVect: Vector containing the x−coordinates of interest
```

```
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation [Hz]
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% antHeight: Antenna height [m]
% return: uValues: Calculated u-values (with 1/r-loss added)

function [uValues] = FDM_addRloss(zFieldInit, ...
    zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer,...
    antHeight)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
Hindex = maxHeigthInterestZIndex;
xVect= verticalVector(xVect);
zVect = verticalVector(zVect);

deltaZ = zVect(2,1)-zVect(1,1);
deltaX = xVect(2,1)-xVect(1,1);

% Creating the absortion layer:
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsInAbsLayer);

% Vector a- and b-grid:
aGrid = zeros(numZpoints,1);
bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);


% Preparations for calculation the u-values for successive x-values:
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
onesVect = ones(numZpoints-1,1);
diagVmat = 2 + bGrid(1,1)- aGrid(:,1);
diagVmat(1,1) = 1;
diagVmat(numZpoints,1) = 1;
VmMat = (diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1)));
VmMat(1,2) = 0;
VmMat(numZpoints,numZpoints-1) = 0;
diagAmat = -2 + bGrid(1,1)+ aGrid(:,1);
```

```
diagAmat(1,1) = 1;
diagAmat(numZpoints,1) = 1;
AmMat = (diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1));
AmMat(1,2) = 0;
AmMat(numZpoints,numZpoints-1) = 0;

%disp('Create transition matrix')
transitionMat = (VmMat/AmMat);

% Diagonalization of the transition matrix:
[eigVect,eigValues] = eig(transitionMat);
eigVectInv = sparse(pinv(eigVect));
eigVect = sparse(eigVect);
eigValues = sparse(eigValues);
initEigInv = sparse(eigVectInv*zFieldInit);


auxEigValues = eigValues;
% Calculating the field:
parfor x = 2:numXpoints
   distVect = sqrt(x.^2 + (zVect-antHeight).^2);
    auxEigValues = eigValues.^x;
    uGrid(:,x) = eigVect*auxEigValues*initEigInv;


end
parfor x = 2:numXpoints
   distVect = sqrt(x.^2 + (zVect-antHeight).^2);
   % Adding additional loss (1/r):
    uGrid(:,x) = uGrid(:,x).*(1./distVect);
end

uValues = uGrid;

end
```

E.2.1.6 SSAirregularTerrainAbsoptionLayer

Field simulation over irregular terrain using the SSA.

```
% SSAirregularTerrainAbsoptionLayer.m: Uses the split-step-algoritm to
%                                       calculate the wave propagation over
%                                       irregular terrain with an
%                                       absorption layer.
% zFieldInit: Vector containing the initial electric field along the z-axis
```

```
% zVect: Vector containing the z-coordinates of interest
% xVect: Vector containing the x-coordinates of interest
% zSurfaceVect: Vector containing the z-coordinates of the surface
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                          interest, above this index there will be
%                          absorption in order to avoid reflection from the
%                          sky.
% frequency: The frequency of operation
% numPointsLayer: The number of points in the absorption layer
% zs: Transmitter antenna height

% return: uValues: The calculated u-values.

function [uValues] = SSAirregularTerrainAbsoptionLayer(zFieldInit,zVect,...
    xVect,zSurfaceVect,maxHeigthInterestZIndex,frequency,numPointsLayer,zs)

c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
zVectInit = zVect;
numZpoints = length(zVect);
numZpointsInit = numZpoints;
numXpoints = length(xVect);
zVect = verticalVector(zVect);
xVect = verticalVector(xVect);
zSurfaceVect = verticalVector(zSurfaceVect);
Hindex = maxHeigthInterestZIndex;


deltaZ = zVect(2,1) - zVect(1,1);
deltaX = xVect(2,1) - xVect(1,1);

% Finding the maximum height of the surface:
[maxZ, indexMaxZ] = max(zSurfaceVect);

% The number of points to add:
addVect =verticalVector([zVect(numZpoints,1)+deltaZ:deltaZ:...
    (zVect(numZpoints,1)+ maxZ+deltaZ)]);
zVect = vertcat(zVect,addVect);
numZpoints = length(zVect);
% Mapping the surface-vector to correspond with the heights in zVect
% The values of the new surface vector contains the indices corresponding
% to the height in the zVect
zSurfaceIndices = zeros(numXpoints,1);
for x = 1:numXpoints
    zSurfaceIndices(x,1) = find(zVect >=zSurfaceVect(x,1),1,'first');
end

```

```
% Creating absorption layer:
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsLayer);

uGrid = zeros(numZpoints,numXpoints);
uGrid(1:numZpointsInit,1)= zFieldInit;
indicesZminusOne = verticalVector(linspace(1,(numZpointsInit-1), ...
    (numZpointsInit-1)));
Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
    .*deltaX./(2.*k.*(numZpointsInit.^2)));
Pprime = verticalVector(Pprime);

% % Creating the vGrid:
% vGrid = zeros(numZpointsInit,numXpoints);
%
% % Creating the vector containing the slopes of the terrain
% alphaVect =  (zSurfaceVect(2:numXpoints,1)- ...
%     zSurfaceVect(1:(numXpoints-1),1))./deltaX;
% alphaVect = vertcat(0,alphaVect);
%
% zeta = zeros(numZpoints,1);
%
% vGrid(:,1) = uGrid(zSurfaceIndices(1):...
%     (zSurfaceIndices(1)+numZpointsInit-1),1);


% The staircase model:
zVect = zVectInit;
numZpoints = length(zVect);
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsLayer);
indicesZminusOne = verticalVector(...
    linspace(1,(numZpoints-1),(numZpoints-1)));
Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
    .*deltaX./(2.*k.*(numZpoints.^2)));
Pprime = verticalVector(Pprime);

% Calculating the field:
for x = 2:numXpoints
     numZpoints = length(zVect);
    uGrid(1:(numZpoints-1),x) = inverseDiscreteSineTrans( ...
        Pprime.*discreteSineTrans(uGrid(1:(numZpoints-1),x-1)));
     uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 -1) ...
        .*deltaX./2).*uGrid(1:numZpoints,x);

    % Adjusting to the surface:
```

```
    if zSurfaceIndices(x,1) > 1
        uGrid(1:(zSurfaceIndices(x,1)-1),x) = 0;
    end

end
uValues = uGrid;
end
```

E.2.1.7 FDMirregularTerrainAbsorptionLayer

Field simulation over irregular terrain using the FDM.

```
% FDMirregularTerrainAbsorptionLayer.m : Calculates the electric field
%                             using the finite difference method for a given
%                             terrain profile. The function
%                             has an absorbing layer at the top, preventing
%                             the simulations to have reflections from the
%                             sky.

% zFieldInit: Vector containing the initial electric field along the z-axis
% zVect: Vector containing the z-coordinates of interest
% xVect: Vector containing the x-coordinates of interest
% zSurfaceVect: Vector containing the z-coordinates of the surface
% maxHeigthInterestZIndex: the z-index that contains the highest index of
%                           interest, above this index there will be
%                           absorption in order to avoid reflection from the
%                           sky.
% frequency: The frequency of operation
% numPointsInAborptionLayer: Number of points in the absorption layer.

% return: uValues: The calculated u-values.

function [uValues] = FDMirregularTerrainAbsorptionLayer(zFieldInit, ...
    zVect,xVect,zSurfaceVect,maxHeigthInterestZIndex,frequency, ...
    numPointsInAborptionLayer)

c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numZpoints = length(zVect);
numXpoints = length(xVect);
zSurfaceVect = verticalVector(zSurfaceVect);
xVect = verticalVector(xVect);
zVect= verticalVector(zVect);
Hindex = maxHeigthInterestZIndex;
```

```

% Mapping the surface—vector to correspond with the heights in zVect
% The values of the new surface vector contains the indices corresponding
% to the height in the zVect
zSurfaceIndices = zeros(numXpoints,1);
for x = 1:numXpoints
    zSurfaceIndices(x,1) = find(zVect >=zSurfaceVect(x,1),1,'first');
end

% Hanning window for absorption layer:
% indiceValues = (verticalVector([0:1:numZpoints—Hindex]))./(numZpoints—Hindex);
% absorptionLayer = (1 +cos(pi.*indiceValues))./2;
% indexRefraction = ones(numZpoints,1);
%
% indiceValues = verticalVector([1:1:(numZpoints—Hindex+1)]);
% indiceValues = (indiceValues./(numPointsPerLayer*numLayers));
indexRefraction = createAbsorptionLayer(zFieldInit,...
    Hindex,numPointsInAborptionLayer);
%indexRefraction(Hindex:numZpoints,1) = indexRefraction(Hindex:numZpoints,1) ...
%     + 1j.*absorptionLayer;

% Finding the ''grid'' of a— and b—values:
deltaZ = zVect(2,1)—zVect(1,1);
deltaX = xVect(2,1)—xVect(1,1);
bGrid = 1j.*ones(numZpoints,1).*4.*k.*(deltaZ.^2)./deltaX;
aGrid = ones(numZpoints,1).*k.^2.*(indexRefraction.^2 —1).*deltaZ.^2;


% Calculate the u—values for successive x—values:
uGrid = zeros(numZpoints,numXpoints);
uGrid(:,1)= zFieldInit;
onesVect = ones(numZpoints—1,1);
VmMat = zeros(numZpoints,numZpoints);
VmRes = zeros(numZpoints,1);
AmMat =  zeros(numZpoints,numZpoints);

 diagVmat = 2 + bGrid(:,1)— aGrid(:,1);
    diagVmat(1,1) = 1;
    diagVmat(numZpoints,1) = 1;
    VmMat = diag(diagVmat)+ (—1.*diag(onesVect,—1))+(—1.*diag(onesVect,1));
    VmMat(1,2) = 0;
    VmMat(numZpoints,numZpoints—1) = 0;

     diagAmat = —2 + bGrid(:,1)+ aGrid(:,1);
    diagAmat(1,1) = 1;
    diagAmat(numZpoints,1) = 1;
     AmMat = diag(diagAmat)+diag(onesVect,—1) + diag(onesVect,1);
     AmMat(1,2)= 0;
```

```
     AmMat(numZpoints,numZpoints−1) = 0;

for x = 2:numXpoints
    % Filling in the Um and Vm matrices for field propagation estimation:

    VmRes = VmMat*uGrid(:,x−1);



    %AmMat*uGrid(:,x) = VmRes
    uGrid(:,x) = linsolve(AmMat,VmRes);

    if zSurfaceIndices(x,1) > 1
        uGrid(1:(zSurfaceIndices(x,1)−1),x) = 0;
        %uGrid(1:(zSurfaceIndices(x,1)),x) = 0;
    end
    %x

end

uValues = uGrid;
%uValues(abs(uValues)<10^−3) = 10^−3;
%uValues(abs(uValues)>10^−1) = 10^−1;
end
```

E.2.2 Comparison Functions

E.2.2.1 freeSpaceLoss_beamParam

Compares the simulated free-space loss with the analytical free-space loss, using the correct beam shape.

```
% freeSpaceLoss.m: Calculates the free−space loss. Adjusts the beam pattern
%                   according to the parameters (works only for rx = tx).
% tx: Transimtter antenna height
% rx: Receiver antenna height
% distance: The distance between the antennas, ignoring height difference
% eField: Calculated electric field along the surface at a given height,
%          zs, may contain multiple heights of consideration
% zs: The height eField is taken from
% beta: Beam−width [rad]
% frequency: The frequency of operation
% simulationType: Cell−array containing strings with the name of the
%                  simulation types used.
```

```
% plotTitle: The title of the plot; Vector of strings.
% saveFig: String of path and filename for saving the plot result. To not
%          save the result, saveFig should be an empty string (''). File
%          extension: .png


function freeSpaceLoss_beamParam(A,tx,rx,xVect,eField,zs,...
    beta,frequency,simulationType, plotTitle,saveFig)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numXpoints = length(xVect);
xVect = verticalVector(xVect);
numXpoints = length(xVect);
tx_x = xVect(1,1);
rx_x = xVect(numXpoints,1);
tx_z = tx;
rx_z = rx;
% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);

plotNames = cell(1+length(simulationType),1);
plotNames{1,1} = 'Free-space Loss - analytical model';
for i = 1: length(simulationType);
    plotNames{i+1,1} = simulationType{i};
end
% Calculating the received fields:
a = tx_z/rx_z;

xVectAux = xVect(2:numXpoints,1);
thetaDir = atan((rx_z-tx_z)./(abs(tx_x-xVectAux)));
Adirect = A.*exp(-2.*log10(2).*((thetaDir/beta).^2));
Atotal = Adirect;
Pl = (lambda./(4.*pi.*(abs(tx_x-xVectAux)))).*((Atotal./Adirect).^2);

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
```

```
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end


% Calculating the plane earth loss (path loss for plane earth):

Pl_dB = 20.*log10(verticalVector(Pl(9:numXpoints-1,1)));
semilogx(xVect(9:numXpoints-1,1),Pl_dB,'Color',cmap(1,:));
xlabel('Distance [m]');
ylabel('Path Loss [-dB]');

for i = 1: numCases
    hold on

    % The simulated field:
    pl1 = verticalVector(20.*log10(abs(eField(i,:))));
    pl2 = - 10.*log10(xVect);

    pathLoss = pl1 + pl2;
    semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),...
        'Color',cmap(i+1,:));

end
legend(plotNames);
title(plotTitle);
grid on

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
end

end
```

E.2.2.2 pathLossFlat_beamParam

Compares the path loss over a flat surface from the simulated fields with the analytical path loss.

```
% pathLossFlat.m: Calculates the path loss along a flat surface, used for
%                 comparing with plane earth loss model, assuming zero
%                 reflection loss. Adjusts the beam pattern according to
%                 the parameters (works only for rx = tx).
% tx: Transimtter antenna height
% rx: Receiver antenna height
% eField: Calculated electric field along the surface at a given height,
%         zs, may contain multiple heights of consideration
% zs: The height eField is taken from
% beta: Beam-width [rad]
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: The title of the plot; Vector of strings.
% saveFig: String of path and filename for saving the plot result. To not
%          save the result, saveFig should be an empty string (''). File
%          extension: .png


function pathLossFlat_beamParam(A,tx,rx,xVect,eField,zs,...
    beta,frequency,simulationType, plotTitle,saveFig)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
numXpoints = length(xVect);
xVect = verticalVector(xVect);
numXpoints = length(xVect);
tx_x = xVect(1,1);
rx_x = xVect(numXpoints,1);
tx_z = tx;
rx_z = rx;
% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);

```

```
plotNames = cell(1+length(simulationType),1);
plotNames{1,1} = 'Plane Earth Loss';
for i = 1: length(simulationType);
    plotNames{i+1,1} = simulationType{i};
end
% Calculating the received fields:
a = tx_z/rx_z;

xVectAux = xVect(2:numXpoints,1);
thetaDir = atan((rx_z-tx_z)./(abs(tx_x-xVectAux)));
Adirect = A.*exp(-2.*log10(2).*((thetaDir/beta).^2));

% The reflected field:

% Find theta:
d_tx = a.*abs(tx_x-xVectAux)./(1+a);
thetaRefl = (pi/2) - atan(d_tx./tx_z);
Areflect = A.*exp(-2.*log10(2).*((thetaRefl./beta).^2));

Atotal = Adirect + Areflect.*exp(1j.*k.*rx_z.*tx_z./(abs(tx_x-xVectAux)));

Pl = (lambda./(4.*pi.*(abs(tx_x-xVectAux)))).*(abs(Atotal./Adirect));

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end


% Calculating the plane earth loss (path loss for plane earth):
Pl_dB = 20.*log10(verticalVector(Pl(9:numXpoints-1,1)));
semilogx(xVect(9:numXpoints-1,1),Pl_dB,'Color',cmap(1,:));
xlabel('Distance [m]');
ylabel('Path Loss [-dB]');
for i = 1: 2:(numCases-1)
    % Calculating plane earth loss from the simulated results:
    hold on

    pl1 = verticalVector(20.*log10(abs(eField(i,:))));
    pl2 = - 10.*log10(xVect);

```

```
    pathLoss = pl1 + pl2;
    semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),...
        'Color',cmap(i+1,:));

    pl1 = verticalVector(20.*log10(abs(eField(i+1,:))));
    pl2 = - 10.*log10(xVect);

    pathLoss = pl1 + pl2;
    semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),'--',...
        'Color',cmap(i+1,:));

end

legend(plotNames,'Location','SouthWest');
title(plotTitle);
grid on

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        %print (fig, '-dpdf', titleFig);
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        save2pdf(titleFig);
end

end
```

E.2.2.3 pathLossIndra_alongX

Horizontal comparison between the relative field strengths from the simulated results and the results from a given file.

```
% pathLossIndra_alongX.m: Compare the relative field strengths from the
%                         simulated results with the results from a
%                         given file. Horizontal comparison.

% rx_xArr: Receiver antenna (obseravation points) coordinates along the
%          x-axis.
% rx_zArr: Receiver antenna (observation points) coordinates along the
%          z-axis.
% eField: Calculated electric field at the coordinates to rx_xArr and
%         rx_zArr.
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: Title of the plot.
% saveFig: file name of the plot, leave blank (' ') if the plot should not
%          be saved.
% compareFile: File name and relative path if necessary of the file to
%              compare results with. Leave blank (' ') if there is no file
%              to compare with.


function pathLossIndra_alongX(rx_xArr,rx_zArr,eField,...
    frequency,simulationType, plotTitle,saveFig,compareFile)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
columnNumber = 2;
if compareFile ~= ' '

    [eFieldValues,zVectIndra]=importFieldResultsFromFile(...
        compareFile,columnNumber);
end

rx_zArr = verticalVector(rx_zArr);
numRxPoints = length(rx_zArr);

thisFontSize = 10;


% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);
```

```
if compareFile ~= ' '
    plotNames = cell(1+length(simulationType),1);
    plotNames{1,1} = 'E-field Indra';
    for i = 1: length(simulationType);
        plotNames{i+1,1} = simulationType{i};
    end
else % No file to compare with
    plotNames =simulationType;
end

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        %figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end
lineWidth  = 1;
if compareFile ~= ' '
 Pl =eFieldValues; % In dB already
 Pl_dB =Pl;
 semilogx(zVectIndra,Pl_dB,'Color',cmap(1,:),'LineWidth',lineWidth);
 minCompareValue = min(Pl_dB);
end
ax1 = gca;
set(ax1,'XColor','k','YColor','k')
%legend(plotNames{1,1})
set(ax1, 'Box', 'off')
set(ax1, 'Color', 'none')
 for i = 1:numCases
     if compareFile ~= ' '
         hold on
     elseif i > 1
         hold on
     end
    pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
    pl2 = 0; % - 10.*log10(xVect);
    minValueField = min(pl1);

    pathLoss = pl1 + pl2;


   semilogx(rx_xArr,pathLoss,'Color',cmap(5-i+1,:),'LineWidth',lineWidth);
```

```
 end
 grid on
set(gca,'FontSize',thisFontSize)
legend(plotNames,'Location','NorthEast')
ylabel('Relative field strength [dB]');
xlabel('Distance [m]');

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        save2pdf(titleFig);
end

end
```

E.2.2.4 pathLossFlat_Indra

Vertical comparison between the relative field strengths from the simulated results and the results from a given file.

```
% pathLossFlat_Indra.m: Vertical comparison between the relative field
%                       strengths of the simulated fields and the results
%                       from a given file.

% rx_xArr: Receiver antenna (obseravation points) coordinates along the
%          x-axis.
% rx_zArr: Receiver antenna (observation points) coordinates along the
%          z-axis.
% eField: Calculated electric field at the coordinates to rx_xArr and
```

```
%          rx_zArr.
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: Title of the plot.
% saveFig: file name of the plot, leave blank (' ') if the plot should not
%           be saved.
% compareFile: File name and relative path if necessary of the file to
%              compare results with.


function pathLossFlat_Indra(rx_xArr,rx_zArr,eField,...
    frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
columnNumber = 3;
[eFieldValues,zVectIndra]=importFieldResultsFromFile(compareFile,...
    columnNumber);
eFieldValues = verticalVector(eFieldValues);
zVectIndra = verticalVector(zVectIndra);

indexMaxIndra = find(zVectIndra >= compareHeight,1);
zVectIndra = zVectIndra(1:indexMaxIndra,1);
eFieldValues = eFieldValues(1:indexMaxIndra,1);

rx_xArr = verticalVector(rx_xArr);
rx_zArr = verticalVector(rx_zArr);

indexMaxZ = find(rx_zArr >= compareHeight,1); %240 ,1);
rx_xArr = rx_xArr(1:indexMaxZ,1);
rx_zArr = rx_zArr(1:indexMaxZ,1);
eField = eField(:,1:indexMaxZ,1);

rx_zArr = verticalVector(rx_zArr);
numRxPoints = length(rx_zArr);
thisFontSize = 10;



% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
```

```
cmap = hsv(numCases+1);

plotNames = cell(1+length(simulationType),1);
plotNames{1,1} = 'E-field Indra';
for i = 1: length(simulationType);
    plotNames{i+1,1} = simulationType{i};
end

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end

% Plotting the E-field calculated at indra:
lineWidth  = 1;
Pl =eFieldValues; % In dB already
Pl_dB =Pl;
hl1 = line(zVectIndra(7:length(zVectIndra),1),...
    Pl_dB(7:length(zVectIndra),1),'Color',cmap(1,:),'LineWidth',lineWidth);
ax1 = gca;
set(ax1,'XColor','k','YColor','k')
%legend(plotNames{1,1})
set(ax1, 'Box', 'off')
set(ax1, 'Color', 'none')
maxImportedEfield = max(eFieldValues);
set(gca,'FontSize',thisFontSize)
xlabel('Height [m]');
ylabel('Relative field strength [dB]');

ax2 = axes('Position',get(ax1,'Position'),...
           'XAxisLocation','top',...
           'YAxisLocation','right',...
           'Color','none',...
           'XColor','k','YColor','k');

for i = 1: numCases
    hold on

    % The simulated fields:
    pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
    maxThisField = max(pl1);
```

```
    maxDiff = maxImportedEfield - maxThisField;
    pl2 = 0;
    pathLoss = pl1 + pl2;

    pathLoss = pathLoss + maxDiff;

    hl2 = line(rx_zArr(4:length(rx_zArr),1),pathLoss(4:length(rx_zArr)),...
        'Color',cmap(5-i+1,:),'Parent',ax2,'LineWidth',lineWidth);
end

set(ax2, 'Box', 'off')
set(gca,'FontSize',thisFontSize)
legend(ax1,plotNames{1,1},'Location','South')
legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
title(plotTitle);

grid on
linkaxes([ax2 ax1],'y');

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        %print (fig, '-dpdf', titleFig);
        save2pdf(titleFig);
end

end
```

E.2.2.5 pathLossFlat_Indra_minComp

Vertical comparison up to a given maximum height, between the relative field strengths from the simulated results and the results from a given file. The fields are shifted to have the same minimum value.

```
% pathLossFlat_Indra_minComp.m: Vertical comparison between the relative
%                               field strengths of the simulated fields and
%                               the results from a given file. The
%                               maximum height of comparison is specified in
%                               compareHeight variable. The functions shifts
%                               the fields to have the same minimum value.

% rx_xArr: Receiver antenna (obseravation points) coordinates along the
%          x-axis.
% rx_zArr: Receiver antenna (observation points) coordinates along the
%          z-axis.
% distance: The distance between the antennas, ignoring height difference
% eField: Calculated electric field at the coordinates to rx_xArr and
%         rx_zArr.
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: Title of the plot.
% saveFig: file name of the plot, leave blank (' ') if the plot should not
%          be saved.
% compareFile: File name and relative path if necessary of the file to
%              compare results with.
% compareHeight: The maximum height of comparison.

function pathLossFlat_Indra_minComp(rx_xArr,rx_zArr,eField,...
    frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
columnNumber = 3;
[eFieldValues,zVectIndra]=importFieldResultsFromFile(compareFile,...
    columnNumber);
eFieldValues = verticalVector(eFieldValues);
zVectIndra = verticalVector(zVectIndra);

indexMaxIndra = find(zVectIndra >= compareHeight,1);
zVectIndra = zVectIndra(1:indexMaxIndra,1);
eFieldValues = eFieldValues(1:indexMaxIndra,1);

rx_xArr = verticalVector(rx_xArr);
rx_zArr = verticalVector(rx_zArr);

indexMaxZ = find(rx_zArr >= compareHeight,1);
rx_xArr = rx_xArr(1:indexMaxZ,1);
rx_zArr = rx_zArr(1:indexMaxZ,1);
```

```
eField = eField(:,1:indexMaxZ,1);

rx_zArr = verticalVector(rx_zArr);
numRxPoints = length(rx_zArr);

thisFontSize = 10;


% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);

plotNames = cell(1+length(simulationType),1);
plotNames{1,1} = 'E-field Indra';
for i = 1: length(simulationType);
    plotNames{i+1,1} = simulationType{i};
end

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');

end

% Plotting the E-field calculated at indra:
lineWidth  = 1;
startIndex_importField  =14; %7; %13;
Pl =eFieldValues; % In dB already
Pl_dB =Pl;
hl1 = line(zVectIndra(startIndex_importField:length(zVectIndra),1),...
    Pl_dB(startIndex_importField:length(zVectIndra),1),'Color',...
    cmap(1,:),'LineWidth',lineWidth);
ax1 = gca;
set(ax1,'XColor','k','YColor','k')
%legend(plotNames{1,1})
```

```
set(ax1, 'Box', 'off')
set(ax1, 'Color', 'none')
maxImportedEfield =min(Pl_dB(startIndex_importField:length(zVectIndra),1));
set(gca,'FontSize',thisFontSize)
xlabel('Height [m]');
ylabel('Relative field strength [dB]');


ax2 = axes('Position',get(ax1,'Position'),...
            'XAxisLocation','top',...
            'YAxisLocation','right',...
            'Color','none',...
            'XColor','k','YColor','k');
startIndex = 4;
for i = 1: numCases
    hold on

    % The simulated fields:
    pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
    maxThisField = min(pl1(startIndex:length(rx_zArr)));
    maxDiff = maxImportedEfield - maxThisField;
    pl2 = 0;
    pathLoss = pl1 + pl2;
    pathLoss = pathLoss + maxDiff;

    hl2 = line(rx_zArr(startIndex:length(rx_zArr),1),...
        pathLoss(startIndex:length(rx_zArr)),...
        'Color',cmap(5-i+1,:),'Parent',ax2,'LineWidth',lineWidth);
end
set(ax2, 'Box', 'off')
set(gca,'FontSize',thisFontSize)
legend(ax1,plotNames{1,1},'Location','South')
legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
title(plotTitle);

grid on
linkaxes([ax2 ax1],'y');
linkaxes([ax2 ax1],'x');
switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
```

```
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        %print (fig, '-dpdf', titleFig);
        save2pdf(titleFig);%,handle,dpi)
end

end
```

E.2.2.6 pathLossWedge_Indra

Vertical comparison between the relative field strengths from the simulated results and the results from a given file. Shifts the fields to be aligned at maximum height of comparison.

```
% pathLossWedge_Indra.m: Vertical comparison between the relative field
%                        strengths of the simulated fields and the results
%                        from a given file. Shiftes the fields to be
%                        alligned with the maximim value of the results
%                        imported from the file.
% Xn: Vector of x-values
% Zn: Vector of corresponding z-values
% deltaX: spacing between the sampling points along the x-axis, if set to 0
%         the spacing is according to Nyquist's theorem.
% tx: Transimtter antenna height
% rx_xArr: Receiver antenna (obseravation points) coordinates along the
%          x-axis.
% rx_zArr: Receiver antenna (observation points) coordinates along the
%          z-axis.
% distance: The distance between the antennas, ignoring height difference
% eField: Calculated electric field at the coordinates to rx_xArr and
%         rx_zArr.
% zs: The height eField is taken from
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: Title of the plot.
% saveFig: file name of the plot, leave blank (' ') if the plot should not
%          be saved.
% compareFile: File name and relative path if necessary of the file to
%              compare results with. Leave empty (' ') if there are no
```

```
%               results to compare with.


function pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eField,...
    zs,beta,frequency,simulationType, plotTitle,saveFig,compareFile)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
columnNumber = 3;
thisFontSize = 10;

tx_x = Xn(1,1);
tx_z = tx;

indexMaxZ = find(rx_zArr >= 250 ,1);
rx_xArr = rx_xArr(1:indexMaxZ);
rx_zArr = rx_zArr(1:indexMaxZ);
eField = eField(:,1:indexMaxZ);


rx_zArr = verticalVector(rx_zArr);
numRxPoints = length(rx_zArr);


% Finding the dimensions of the incoming e-field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end

switch compareFile
```

```
    case ' '
        % Do not try to import anyting
        plotNames = cell(length(simulationType),1);
        for i = 1: length(simulationType);
            plotNames{i,1} = simulationType{i};
        end

        xlabel('Height [m]');
        ylabel('Relative field strength [dB]');
        for i = 1: numCases
            hold on
            pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
            pl2 = 0;
            pathLoss = pl1 + pl2;
            hl2 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i+1,:));
        end
        legend(plotNames,'Location','SouthEast')
        title(plotTitle);
        grid on


    otherwise
        [eFieldValues,zVectIndra]=importFieldResultsFromFile(...
            compareFile,columnNumber);

        plotNames = cell(1+length(simulationType),1);
        plotNames{1,1} = 'E-field Indra';
        for i = 1: length(simulationType);
            plotNames{i+1,1} = simulationType{i};
        end

        % Plotting the E-field calculated at indra:
        lineWidth  = 1;
        Pl =eFieldValues; % In dB already
        Pl_dB =Pl(4:length(Pl));
        maxValueCompareField = max(Pl_dB);
        hl1 = line(zVectIndra(4:length(Pl)),Pl_dB,'Color',cmap(1,:), ...
            'LineWidth',lineWidth);
        ax1 = gca;
        set(ax1,'XColor','k','YColor','k')
        set(ax1, 'Box', 'off')
        set(ax1, 'Color', 'none')
        set(gca,'FontSize',thisFontSize)
        xlabel('Height [m]');
        ylabel('Relative field strength [dB]');
        ax2 = axes('Position',get(ax1,'Position'),...
            'XAxisLocation','top',...
            'YAxisLocation','right',...
```

```
            'Color','none',...
            'XColor','k','YColor','k');

        for i = 1: numCases
            hold on
            pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
            maxValueEfield = max(pl1);
            maxValDiff = maxValueCompareField - maxValueEfield;
            pl2 = 0;
            pathLoss = pl1 + pl2;
            pathLoss = pathLoss + maxValDiff;
            hl2 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i+1,:),...
                'Parent',ax2,'LineWidth',lineWidth);
        end
        set(ax2, 'Box', 'off')
        set(gca,'FontSize',thisFontSize)
        legend(ax1,plotNames{1,1},'Location','East'); %'South')
        legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
        title(plotTitle);
        grid on
        linkaxes([ax2 ax1],'y');

end

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        %print (fig, '-dpdf', titleFig);
        save2pdf(titleFig);
end

end
```

E.2.2.7 pathLossWedge_Hviid

Plots the path loss of the given e-fields for vertical comparison if there are no results from file to compare with. If there are result from file to compare with, the relative field strengths are plotted.

```
% pathLossWedge_Hviid.m: If there are no results from files to compare
%                         with, the path losses of the incoming fields are
%                         calculated and plotted for vertical comparison.
%                         If there are results from other files to compare
%                         with, the relative field strenghts are plotted for
%                         vertical comparison.

% antHeight: The height of the transmitter antenna.
% rx_xArr: Receiver antenna (obseravation points) coordinates along the
%          x-axis.
% rx_zArr: Receiver antenna (observation points) coordinates along the
%          z-axis.
% eField: Calculated electric field at the coordinates to rx_xArr and
%         rx_zArr.
% frequency: The frequency of operation
% simulationType: Cell-array containing strings with the name of the
%                 simulation types used.
% plotTitle: Title of the plot.
% saveFig: file name of the plot, leave blank (' ') if the plot should not
%          be saved.
% compareFile: File name and relative path if necessary of the file to
%              compare results with. Leave empty (' ') if there are no
%              results to compare with.
% compareHeight: The maximum height of interest for comparison in the
%                vertical direction.


function pathLossWedge_Hviid(antHeight, rx_xArr,rx_zArr,eField,...
    frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;
columnNumber = 3;
thisFontSize = 10;
rx_xArr = verticalVector(rx_xArr);

if (compareHeight > 0)
indexMaxZ = find(rx_zArr >= compareHeight ,1);
rx_xArr = rx_xArr(1:indexMaxZ);
rx_zArr = rx_zArr(1:indexMaxZ);
```

```
eField = eField(:,1:indexMaxZ);
end

rx_zArr = verticalVector(rx_zArr);
numRxPoints = length(rx_zArr);


% Finding the dimensions of the incoming e—field array:
[lines,columns] = size(eField);
if(lines>columns)
    eField = eField';
    numCases = columns;
else
    numCases = lines;
end
cmap = hsv(numCases+1);

% Save the figure?
saveFigVal = isempty(saveFig);

switch saveFigVal
    case 1
        % Not save figure
        figure()
    case 0
        % Save figure
        fig = figure('visible','on');
end

switch compareFile


    case ' '
        % Do not try to import anyting
        plotNames = cell(length(simulationType),1);
        for i = 1: length(simulationType);
            plotNames{i,1} = simulationType{i};
        end
        %figure (1)
        hFig = fig; % figure(1);
        set(gcf,'PaperPositionMode','auto')
        xwidth =450; % 360;
        ywidth = 480;
        set(hFig, 'Position',[800 200 xwidth ywidth])
        xTick_arr = [0:20:compareHeight];
        set(gca,'XTick',xTick_arr);
        set(gca,'YDir','reverse');

```

```
        startIndex = 3;
        numPointsZ = length(rx_zArr);
        rx_zArr = rx_zArr(startIndex:numPointsZ,1);
        rx_xArr = rx_xArr(startIndex:numPointsZ,1);
        xlabel('Receiver height [m]');
        ylabel('Path loss [dB]');
        distVect = (((antHeight-rx_zArr).^2) + (rx_xArr.^2)).^0.5;
        eField=eField(:,startIndex:numPointsZ);
        for i = 1: numCases
            hold on
            % The simulated fields:
            pl1 = -smooth(verticalVector(20.*log10(abs(eField(i,:)))));
            pl2 = +10.*log10(distVect)+ 20*log10(4*pi)-30*log(lambda);
            pathLoss = (pl1 + pl2);
            % Moving the pathloss  up or down:
            hl2 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i,:));
        end
        legend(plotNames,'Location','SouthEast')
        title(plotTitle);
        grid on


    otherwise
        [eFieldValues,zVectIndra]=importFieldResultsFromFile(...
            compareFile,columnNumber);

        plotNames = cell(1+length(simulationType),1);
        plotNames{1,1} = 'E-field Indra';
        for i = 1: length(simulationType);
            plotNames{i+1,1} = simulationType{i};
        end

        % Plotting the E-field calculated at indra:
        lineWidth  = 1;
        Pl =eFieldValues; % In dB already
        Pl_dB =Pl(4:length(Pl));
        maxValueCompareField = max(Pl_dB);
        hl1 = line(zVectIndra(4:length(Pl)),Pl_dB,'Color',cmap(1,:), ...
            'LineWidth',lineWidth);
        ax1 = gca;
        set(ax1,'XColor','k','YColor','k')
        set(ax1, 'Box', 'off')
        set(ax1, 'Color', 'none')
        set(gca,'FontSize',thisFontSize)
        xlabel('Height [m]');
        ylabel('Relative field strength [dB]');
        ax2 = axes('Position',get(ax1,'Position'),...
            'XAxisLocation','top',...
```

```
            'YAxisLocation','right',...
            'Color','none',...
            'XColor','k','YColor','k');

        for i = 1: numCases
            hold on
            % The simulated fields:
            pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
            maxValueEfield = max(pl1);
            maxValDiff = maxValueCompareField - maxValueEfield;
            pl2 = 0;
            pathLoss = pl1 + pl2;
            pathLoss = pathLoss + maxValDiff;
            hl2 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i+1,:),...
                'Parent',ax2,'LineWidth',lineWidth);
        end
        set(ax2, 'Box', 'off')
        set(gca,'FontSize',thisFontSize)
        legend(ax1,plotNames{1,1},'Location','East'); %'South')
        legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
        title(plotTitle);
        grid on
        linkaxes([ax2 ax1],'y');

end

switch saveFigVal
    case 0
        % Save figure
        titleFig = verticalVector(saveFig)';
        regexprep(titleFig,' ','_');
        regexprep(titleFig,'\','_');
        regexprep(titleFig,':','_');
        regexprep(titleFig,'=','_');

        if (isempty(strfind(titleFig,'.png')) == 1)
            % File extension needs to be added.
            titleFig = horzcat(titleFig,'.png');
        end

        saveas(fig,titleFig,'png');
        titleFig = titleFig(1:(length(titleFig)-4));
        titleFig = horzcat(titleFig,'.pdf');
        %print (fig, '-dpdf', titleFig);
        save2pdf(titleFig);
end

end
```

E.2.3 Helping Functions

E.2.3.1 interpolate

Interpolates the given surface coordinates so that they get the correct spacing.

```
% interpolate : Interpolates the given points according to the frequency so
% that the sampling distance between the points in the x-direction is less
% than wavelength/2.

% Xn: Vector of x-values
% Zn: Vector of corresponding z-values
% frequency: The frequency of operation [Hz]
% mode: 'linear' or 'curve' interpolation
% deltaX: spacing between the sampling points along the x-axis, if set to 0
%         the spacing is according to Nyquist's theorem.

% Returns: X- and Z-vectors containg the interpolated values.

% (tested)

function[Xni,Zni] = interpolate(Xn,Zn, frequency,mode, deltaX)
c = 3*10^8; % Speed of light
lambda = c/frequency; % Wavelength

switch deltaX
    case 0
        maxSampleDist = lambda/2;
        sampleDist = maxSampleDist - 0.1*maxSampleDist;
    otherwise
        sampleDist = deltaX;
end

% Creating array containing x-values with appropriate sampling distance:
Xni = [Xn(1):deltaX:Xn(length(Xn))];

% Interpolating the given values:
switch mode
    case 'linear'
        Zni = interp1(Xn,Zn,Xni);
    case 'curve'
        Zni = spline(Xn,Zn,Xni);
end
end
```

E.2.3.2 normalizeSurface

Shifts the altitudes of a surface so that the lowest point is situated at altitude zero.

```
% normalizeSurface.m: Shifts the surface heights so that lowest point is
%                      situated at height 0.
% zSurfaceVect: The initital heights of the surface
% return:
%         zSurfaceNorm: The normalized heights of the surface.
%         truncationValue: The height level the surface is shifted with


function [zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect)

minVal = min(zSurfaceVect);

truncationValue = -minVal;
zSurfaceNorm = truncationValue + zSurfaceVect;
end
```

E.2.3.3 createZvectAbsorptionLayer2

Creates the z-vector with additional height for absorption layer

```
% createZvectAbsorptionLayer2.m: Calculates the z-vector based on the
%                                 maximum heigth of interest. Includes
%                                 additional height for absorption layer.
% Usage: To be used in combination with the split-step algorithm and
%        finite-difference method where an absorption is included.

% maxHeigthInterest: The maximum heigth of interest
% deltaZ: The spacing between the elements in the z-vector
% numPointsInAbsorptionLayer: The number of points in the absorption layer

% return:
%         zVect: Vector containing the z-values
%         Hindex: The index of the value closest to the maximum height of
%                 interest in the zVect.


function [zVect, Hindex]=createZvectAbsorptionLayer2(maxHeigthInterest, ...
    deltaZ,numPointsInAbsorptionLayer)
L = ceil((maxHeigthInterest/deltaZ))+ (numPointsInAbsorptionLayer);
```

```
zVect = verticalVector([0:deltaZ:L]);

% Finding the closest index to the height of interest:
Hindex = ceil(((L/deltaZ) +1)*(maxHeigthInterest/L));
end
```

E.2.3.4 createInitialField

Creates the initial field.

```
% initialField.m: Creates the initial field, using the specified beam
%                 shape.
% zs: source height in meters
% theta0: elevation angle in radians
% beta: Half-power beamwidth (Gaussian beam)
% zVect: Vector containing the heights for field estimation in z-direction
% A: parameter for calculating free-space loss
% frequency: the frequency of operation
% source: 'gaussian1', 'gaussian2' or 'circular'
% return: initialField: The initial field for range x=0


function initialField = createInitialField(zs,theta0,beta,zVect,A,...
    frequency, source)
c = 3*10^8;
lambda = c/frequency;
k = 2*pi/lambda;


numZpoints = length(zVect);
initialField = zeros(numZpoints,1);

switch source
    case 'gaussian1'
        initialField = A.*(k.*beta./(2.*sqrt(2.*log10(2)))) ...
            .*exp(-1j.*k.*theta0.*zVect)...
            .*exp(-((beta.^2)./(8.*log10(2))).*(k.^2).*((zVect-zs).^2));

    case 'gaussian2'
        theta1 = beta;
        theta2 = theta0;
        initialField = sqrt(k).*tan(theta1).*exp(-((k.^2)./2).*((zVect-zs).^2).* ...
            (tan(theta1).^2)).*exp(1j.*k.*(zVect-zs).*sin(theta2));
    case 'circular'
        auxTheta = asin(abs((zVect-zs))./A);
```

```
%         for x = 1:length(auxTheta)
%             if (auxTheta(x) > pi/2)
%                 auxTheta(x) = 0;
%             end
%         end
        auxTheta(abs(auxTheta) > (pi/4)) =pi/2;
        %auxTheta(auxTheta < (-pi/2)) =0; %pi;
        initialField = A*cos(auxTheta);

end
initialField(1) = 0;
end
```

E.2.3.5 verticalVector

Makes a vector vertical if it was not initially.

```
% verticalVector.m
% Checks wether a vector is horizontal or vertical.
% If the vector is horizontal, it is transposed into being vertical.
% The returned vector is vertical

% Vin: Input vector (horizontal or vertical)
% Vout: Output vector, vertical

function [Vout] = verticalVector(Vin)

if( length(Vin(1,:)) > length(Vin(:,1)))
    % Vin is a horizontal vector
    Vout = Vin';
else
    Vout = Vin;
end
end
```

E.2.3.6 getVerticalValues

Extracts the vertical values from the results on an inclined plane.

```
% getVerticalValues.m: Calculates the values in the "vertical" direction,
%                      the direction perpendicular to the downward inclined
%                      plane.
```

```
% xDiff: The distance difference in the x-direction
% zDiff: The height difference in the z-direction
% deltaX: The step size along the x-axis
% deltaZ: The step size along the z-axix
% xVect: The vector containing the x-values
% xDistCompare: The distance along the x-axis (if the surface was flat) where
%              the field comparison "takes place".
% maxZ: The maximum height for field comparison.
% uValues: Calculated grid of u-values.
% directionIncl: The direction of inclination: 'up' or 'down'

% return: zValues: The field values in the ''vertical'' direction



function zValues = getVerticalValues(Xn,Zn,xDiff,zDiff,deltaX,deltaZ,...
    xVect,xDistCompare,maxZ,uValues, directionIncl)

zValues = verticalVector([0:deltaZ:maxZ]); % counts from 0 to maxZ
xStartIndex =  find(xVect >= xDistCompare ,1);

switch directionIncl

    case 'down'
        thetaAux = atan(zDiff/xDiff);
        xDist2 = xDistCompare*cos(thetaAux);
        xStartIndex =  find(xVect >= xDist2 ,1);

        theta = atan(xDiff/abs(zDiff));
        phi = (2*pi) - pi- theta;
        % The height difference at the xDistCompare along the plane to the flat
        % surface:
        %zDiffAdd = round(abs(zDiff) - abs((xDistCompare*sin(thetaAux))));
        zDiffAdd = round(abs(Zn(1)) - abs((xDistCompare*sin(thetaAux))));

    case 'up'
        theta = atan(zDiff/xDiff);
        phi = pi - (pi/2)- theta;
        xDist2 = xDistCompare*cos(theta);
        xStartIndex =  find(xVect >= xDist2 ,1);

        % The height at which is the plane is situated at, at xDistCompare:
        % (the coordinate)
        zDiffAdd = round(xDistCompare*sin(theta)/deltaZ);
        zDiffAdd_noRounded = xDistCompare*sin(theta)/deltaZ;
        %Finding the endpoint of the vector
        vectorLength = maxZ;
```

```
        totLength = vectorLength + zDiffAdd_noRounded;
        xDiffLength = totLength.*cos(phi);
        xDiffCoord = round(xStartIndex - (xDiffLength./deltaX));
        zHeight =  totLength.*sin(phi);
        normVect = [(zHeight-zDiffAdd_noRounded) -xDiffLength];
        % NB: Convention for this thesis: [z x]


end

xCoordVect = zeros(length(zValues),1);
zCoordVect = zeros(length(zValues),1);
for i = 1:length(zValues)
    xCoord = xStartIndex - round(zValues(i).*deltaX.*cos(phi));
    zCoord = zDiffAdd + round(zValues(i).*deltaZ.*sin(phi));
    if(zCoord == 0)
        zCoord = 1;
    end

    zValues(i,1) = uValues(zCoord,xCoord);


    xCoordVect(i,1) = xCoord;
    zCoordVect(i,1) = zCoord;
end

% % Verifying that the length of normVect is vectorLength:
% norm(normVect)
% % Verifying the dot product between the normal and the plane (should be 90
% % deg):
% surfaceVect = [(Zn(length(Zn))-Zn(1)) -(Xn(length(Xn))-Xn(1)) ];
% dotRes = dot(normVect,surfaceVect)/(norm(normVect)*norm(surfaceVect))
%
% figure()
% plot(Xn,Zn);
% hold on
% plot([xStartIndex xDiffCoord],[zDiffAdd zHeight]);

% % Finding the length of the segment between the plane and flat surface, if
% % the normal vector was extended.
% xPosNorm = xDistCompare/cos(theta);



%figure()
plot(Xn,Zn,'k')
hold on
plot(xCoordVect,zCoordVect,'k')
```

```
grid on

% Calculating the dot product:
surfaceVect = [-(Xn(length(Xn))-Xn(1)) (Zn(length(Zn))-Zn(1))];
ninetyDegVect = [-(xCoordVect(length(xCoordVect))-xCoordVect(1)) ...
    (zCoordVect(length(zCoordVect))-zCoordVect(1))];

% The result of the dot product should be zero (or very close to)
dotResult = dot(surfaceVect,ninetyDegVect)/(norm(surfaceVect).*norm(ninetyDegVect))




end
```

E.2.3.7 createAbsorptionLayer

Creates absorption layer.

```
% createAbsorptionLayer.m: Creates an absorption layer for the given
%                            parameters.
% intitialField: The initial field.
% maxHeigthInterestZIndex: The index of the maximum height of interest.
% numPointsInLayer: Number of points in the absorption layer

% return: indexOfRefraction: Vector containing the index of refraction in
%                              the entire z-direction

function [indexOfRefraction] = createAbsorptionLayer(intitialField,...
    maxHeigthInterestZIndex,numPointsInLayer)
numZpoints = length(intitialField);
Hindex = maxHeigthInterestZIndex;

indiceValues = verticalVector([1:1:(numZpoints-Hindex+1)]);
indiceValues = (indiceValues./(numPointsInLayer));
gamma0 = 0.15*0.098;
absorptionLayer = gamma0.*(indiceValues);

indexOfRefraction = ones(numZpoints,1);
indexOfRefraction(Hindex:numZpoints,1) = ...
    indexOfRefraction(Hindex:numZpoints,1) + 1j.*absorptionLayer;

end
```

E.2.3.8 discreteSineTrans

Performs the discrete sine transform.

```
% discreteSineTrans.m: Performs a discrete sine transform of given set of
%                      values, xIn
% xIn: Vector containing the input values

% return: dst: Vector containing the calculated values.

function dst = discreteSineTrans(xIn)
numPoints = length(xIn);
dst = zeros(numPoints,1);
indices = verticalVector(linspace(1,numPoints,numPoints));

for k = 1:numPoints
    dst(k) = sum(xIn.*sin(pi.*k.*indices./(numPoints+1)));
end
end
```

E.2.3.9 inverseDiscreteSineTrans

Performs the inverse discrete sine transform.

```
% inverseDiscreteSineTrans.m: Performs the inverse sine transformation on a
%                             given set of values, yIn
% yIn: Vector containing the input values

% return: idst: Vector containg the calculated values

function idst = inverseDiscreteSineTrans(yIn)
numPoints = length(yIn);
idst = zeros(numPoints,1);
indices = verticalVector(linspace(1,numPoints,numPoints));

for k = 1:numPoints
    idst(k) =(2./(numPoints+1)).*sum(yIn.*sin(pi.*k.*indices./(numPoints+1)));
end
end
```

E.2.3.10 importFieldResultsFromFile

Imports results from file.

```
% importFieldResultsFromFile.m:Imports field results from a specified file,
%                              with the e—field values in column 4, the
%                              z—values in a column to be specified. The
%                              function is adjusted to the format of the
%                              results from Indra.

% fileName: String containing the file name
% column: Column number containing desired range column.

function [eFieldValues,zVect]=importFieldResultsFromFile(fileName,column)

fileValues = xlsread(fileName);

eFieldValues = fileValues(:,4);
zVect = fileValues(:,column);

end
```

E.2.3.11 importParametersFromFile

Imports surface coordinates from file.

```
% importParametersFromFile.m: Imports the surface profile
%                             coordinates from file.

% fileName: String containing the file name
% xColumnNb: The column number containing the x—coordinates.
% zColumnNB: The column number containing the z—coordinates.

% return: Xn: Vector containing the x—coordinates of the surface
%         Zn: Vector containing the z—coordinates of the surface


function [Xn,Zn]=importParametersFromFile(fileName,xColumnNb,zColumnNB)
fileValues = xlsread(fileName);
Xn = fileValues(:,xColumnNb);
Zn = fileValues(:,zColumnNB);
end
```

E.2.3.12 save2pdf

Saves figure to .pdf. Not implemented by the author of the thesis. Courtesy of Gabe Hoffmann for the implementation.

```
%SAVE2PDF Saves a figure as a properly cropped pdf
%
%   save2pdf(pdfFileName,handle,dpi)
%
%   - pdfFileName: Destination to write the pdf to.
%   - handle:  (optional) Handle of the figure to write to a pdf.  If
%              omitted, the current figure is used.  Note that handles
%              are typically the figure number.
%   - dpi: (optional) Integer value of dots per inch (DPI).  Sets
%          resolution of output pdf.  Note that 150 dpi is the Matlab
%          default and this function's default, but 600 dpi is typical for
%          production-quality.
%
%   Saves figure as a pdf with margins cropped to match the figure size.

%   (c) Gabe Hoffmann, gabe.hoffmann@gmail.com
%   Written 8/30/2007
%   Revised 9/22/2007
%   Revised 1/14/2007

function save2pdf(pdfFileName,handle,dpi)

% Verify correct number of arguments
error(nargchk(0,3,nargin));

% If no handle is provided, use the current figure as default
if nargin<1
    [fileName,pathName] = uiputfile('*.pdf','Save to PDF file:');
    if fileName == 0; return; end
    pdfFileName = [pathName,fileName];
end
if nargin<2
    handle = gcf;
end
if nargin<3
    dpi = 150;
end

% Backup previous settings
prePaperType = get(handle,'PaperType');
prePaperUnits = get(handle,'PaperUnits');
```

```
preUnits = get(handle,'Units');
prePaperPosition = get(handle,'PaperPosition');
prePaperSize = get(handle,'PaperSize');

% Make changing paper type possible
set(handle,'PaperType','<custom>');

% Set units to all be the same
set(handle,'PaperUnits','inches');
set(handle,'Units','inches');

% Set the page size and position to match the figure's dimensions
paperPosition = get(handle,'PaperPosition');
position = get(handle,'Position');
set(handle,'PaperPosition',[0,0,position(3:4)]);
set(handle,'PaperSize',position(3:4));

% Save the pdf (this is the same method used by "saveas")
print(handle,'-dpdf',pdfFileName,sprintf('-r%d',dpi))

% Restore the previous settings
set(handle,'PaperType',prePaperType);
set(handle,'PaperUnits',prePaperUnits);
set(handle,'Units',preUnits);
set(handle,'PaperPosition',prePaperPosition);
set(handle,'PaperSize',prePaperSize);
```