



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 frisktlinje 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 parabolisk ligning. Prinsippet for integralligningsmetoden er å beregne feltstyrken i et gitt punkt basert på den direkte bølgen fra senderen og den induerte overflatestrømmen. I litteraturen som ble brukt manglet det noen vesentlige detaljer, og integralligningsmetoden kunne derfor ikke bli ferdigimplementert. Prinsippet for metoden med parabolisk ligning er å løse "the standard parabolic equation", en ligning utledet fra den skalare bølgeligningen. Metoden er implementert 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 irregulariteter 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 elektromagnetiske feltet vesentlig. Derfor bør rullebaneprofilen tas hensyn til ved elektromagnetiske beregninger. Hvor bra integralligningsmetoden fungerer forblir en oppgave til videre arbeid. Metoden med parabolisk ligning er derimot egnet for elektromagnetiske feltberegninger i frekvensområdet til localizeren.

Contents

Contents	v
List of Figures	ix
Nomenclature	xvi
1 Introduction	1
2 Analytical Models and Source Modeling	5
2.1 Plane Earth Loss and Source Modeling	5
2.2 Numerical Source Modeling	10
2.2.1 Isotropic Source	10
2.2.2 Gaussian Source	10
2.2.3 Choice of Source for Numerical Simulations	11
3 Numerical Methods	13
3.1 The Integral Equation Model	13
3.1.1 Numerical Implementation	16
3.1.1.1 Assumptions	16
3.1.1.2 Implementation Based on Hviid et al. [1995]	17
3.1.1.3 Implementation Based on Brennan and Cullen [1998]	18
3.1.2 Remaining Issues	24
3.2 The Parabolic Equation Method	24
3.2.1 Use of Fourier Transform for Solving the Scalar Wave Equation	30
3.2.2 Split-Step Algorithm - Flat Surface	30
3.2.3 Finite-Difference Method - Flat Surface	31
3.2.4 Mathematical Aspects and Simplifications	34
3.2.5 Absorption Layer	35
3.2.6 Non-Flat Surface	41

3.3	Verification of the Simulation Results	42
3.3.1	Relative Field Strength	42
3.3.2	Path Loss	43
3.3.2.1	Flat Surface	43
3.3.2.2	Non-Flat Surface	43
4	Choice of Parameters	45
5	Results	49
5.1	Simulations over a Flat Surface	51
5.1.1	Horizontal Comparison - Comparison along the Surface . . .	55
5.1.2	Vertical Comparison - Comparison in the Height Direction .	57
5.1.3	Flat Surface Summary	60
5.2	Inclined Plane	61
5.2.1	Downwards Inclined Plane	62
5.2.2	Upwards Inclined Plane	68
5.2.3	Inclined Surface Summary	74
5.3	Simulations over a Wedge	74
5.3.1	Wedge Summary	82
5.4	Simulation over Runways	83
5.4.1	The Braunschweig Airport Runway	83
5.4.2	The Luton Airport Runway	88
5.4.3	Runway Simulation Summary	93
5.5	Results Summary	93
6	Discussion and Further Work	95
6.1	Field Strength Near the Surface for Inclined Plane	95
6.2	Modeled Surface Resolution	96
6.3	SSA and FDM Differences - Flat and Non-Flat Surface	96
6.4	Localizer Signals and Wide-Angle Propagation	96
6.5	Runtime	97
6.6	Commercial Software	97
6.7	3D Loss in 2D	97
6.8	3D - Parabolic Equation	98
6.9	Integral Equation Model	98
7	Conclusion	101
	References	103

Appendix A Mathematical Tools	105
A.1 Fourier Transform	105
A.2 Fourier Sine Transform	105
A.3 Discrete Fourier Sine Transform	106
A.4 Approximations of Differentials	106
Appendix B Derivations	107
B.1 Derivation of the Standard Parabolic Equation	107
B.2 Derivation of the Numerical Standard Parabolic Equation	108
Appendix C Plots	111
C.1 Field Simulation Over a Wedge	111
Appendix D Implementation and Simulation	113
D.1 Implementation Terminology	113
D.2 Simulation Using the Implemented Functions	113
D.2.1 Create Initial Field	114
D.2.2 Irregular Surface	114
D.2.3 Field Propagation Algorithms	114
Appendix E Implemented Code	115
E.1 Scripts for the Obtained Results	115
E.1.1 ParabolicEquation_noGround.m	115
E.1.2 ParabolicEquation_SSA_FDM.m	119
E.1.3 ParabolicEquation_SSA_FDM_deltaValueTest.m	122
E.1.4 SSA_FDM_indra_r_loss.m	125
E.1.5 DownwardsInclinedPlane.m	134
E.1.6 UpwardsInclinedPlane2	142
E.1.7 WedgeComparison_Hviid	150
E.1.8 Braunschweig	159
E.1.9 Luton2	166
E.2 Implemented Matlab Functions	173
E.2.1 Field Simulation Algorithms	173
E.2.1.1 FDMnoGround	173
E.2.1.2 splitStepAlgorithmAbsorptionLayer	176
E.2.1.3 FDMAbsorptionLayerNumEfficient2	178
E.2.1.4 SSA_addRloss	180
E.2.1.5 FDM_addRloss	181
E.2.1.6 SSAirregularTerrainAbsoptionLayer	183

E.2.1.7	FDMirregularTerrainAbsorptionLayer	186
E.2.2	Comparison Functions	188
E.2.2.1	freeSpaceLoss_beamParam	188
E.2.2.2	pathLossFlat_beamParam	191
E.2.2.3	pathLossIndra_alongX	193
E.2.2.4	pathLossFlat_Indra	196
E.2.2.5	pathLossFlat_Indra_minComp	199
E.2.2.6	pathLossWedge_Indra	203
E.2.2.7	pathLossWedge_Hviid	207
E.2.3	Helping Functions	211
E.2.3.1	interpolate	211
E.2.3.2	normalizeSurface	212
E.2.3.3	createZvectAbsorptionLayer2	212
E.2.3.4	createInitialField	213
E.2.3.5	verticalVector	214
E.2.3.6	getVerticalValues	214
E.2.3.7	createAbsorptionLayer	217
E.2.3.8	discreteSineTrans	218
E.2.3.9	inverseDiscreteSineTrans	218
E.2.3.10	importFieldResultsFromFile	219
E.2.3.11	importParametersFromFile	219
E.2.3.12	save2pdf	220

List of Figures

1.1	Illustration of the principle for navigation using the signals from the localizer, [Holm, 2002, p. 2-4].	2
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].	2
2.1	Notation for analytical model for plane earth loss.	7
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]	9
2.3	Gaussian beam, half-power beamwidth: 40°	11
2.4	Polar plot of the Gaussian beam: half-power beamwidth at $\pm 20^\circ$. This is at the edge of the domain of validity for the numerical algorithm, it will be treated later.	12
3.1	Illustration of the principle of the Integral Equation Method.	17
3.2	Geometry of the scattering problem for the Integral Equation Method.	18
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.	20
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.	21
3.5	Geometrical aid for equation (3.12).	23
3.6	The setup for the Parabolic Equation Method. h : Transmitter antenna height.	25
3.7	The error due to approximation of square-root operator as a function of the angle from the paraxial direction.	29

3.8	The geometry for the Finite-Difference Method.	32
3.9	Field propagated over a flat surface.	37
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.	38
3.11	Field propagating in free-space with absorption layer at the top and bottom.	39
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.	40
3.13	Irregular terrain modeled using the staircase model.	41
3.14	Principle of the algorithm for calculation of field in ascending terrain. The dots illustrate the field calculation points.	41
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.	42
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.	46
4.2	Path loss comparison at the antenna height, 15 m, along the surface.	47
4.3	Path loss comparison at the antenna height, 15 m, along the surface.	47
4.4	Path loss comparison at the antenna height, 15 m, along the surface.	48
5.1	Color bar for the field plots.	50
5.2	Field simulation using the SSA.	51
5.3	Field simulation using the FDM.	52
5.4	Field simulation using the SSA, $\frac{1}{r}$ -loss added.	53
5.5	Field simulation using the FDM, $\frac{1}{r}$ -loss added.	53
5.6	Illustration of horizontal and vertical comparison.	54
5.7	Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. No additional loss.	56
5.8	Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. $\frac{1}{r}$ -loss added.	56

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.	57
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.	58
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.	59
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.	61
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.	62
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.	63
5.15	Vertical comparison of the relative field strength at the distance of 1000 m along a downwards inclined plane.	64
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.	65
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.	66
5.18	Horizontal comparison between the field over the runway and over a flat surface. Figure 5.18a is for surface reference.	67
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.	68
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.	69
5.21	Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane.	70

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.	71
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.	72
5.24	Horizontal comparison between the field over the upwards inclined plane and over a flat surface. Figure 5.24a is for surface reference.	73
5.25	The setup for the wedge.	74
5.26	Field simulation over the wedge given in figure 5.25 using the SSA. Frequency: 100 MHz.	75
5.27	Field simulation over the wedge given in figure 5.25 using the FDM. Frequency: 100 MHz.	76
5.28	Vertical comparison of path loss at the distance of 5000 m, for field propagated over the wedge in figure 5.25.	78
5.29	Vertical comparison of path loss of the field propagated along a flat surface and behind the wedge, at 5000 m.	80
5.30	Horizontal comparison between the field over the wedge and over a flat surface. Figure 5.30a is for the wedge surface reference.	81
5.31	Surface profile of the runway at the Braunschweig airport. Please note the scaling difference between the x - and z -axis.	83
5.32	Field over the Braunschweig airport, using SSA.	84
5.33	Field over the Braunschweig airport, using FDM.	84
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.	85
5.35	Horizontal comparison between the field over the runway and over a flat surface. Figure 5.35b is for runway surface reference.	87
5.36	The surface profile of the runway at the Luton airport. Please note the scaling difference between the x - and z -axis.	88
5.37	Field over the Luton airport, using SSA.	89
5.38	Field over the Luton airport, using FDM.	89
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.	90
5.40	Horizontal comparison between the field over the runway and over a flat surface. Figure 5.35b is for runway surface reference.	92
6.1	Computational domain for simulation using the Parabolic Equation method in 3D.	98

C.1	Field simulation over the wedge given in figure 5.25 using the SSA.	
	Frequency: 100 MHz.	111
C.2	Field simulation over the wedge given in figure 5.25 using the FDM.	
	Frequency: 100 MHz.	112

Nomenclature

Roman Symbols

E	Electric field
H	Magnetic field
r'	Vector from origin to the source point
r	Vector from origin to the observation point
$E^s(r)$	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

LIST OF FIGURES

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].

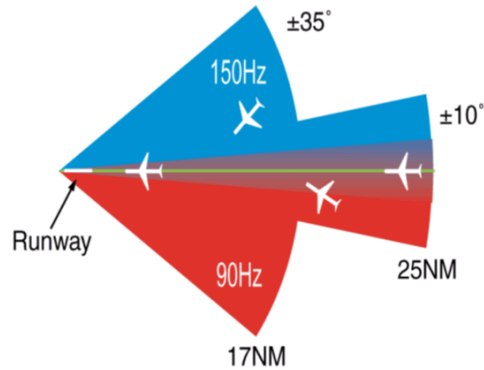


Figure 1.1: Illustration of the principle for navigation using the signals from the localizer, [Holm, 2002, p. 2-4].

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

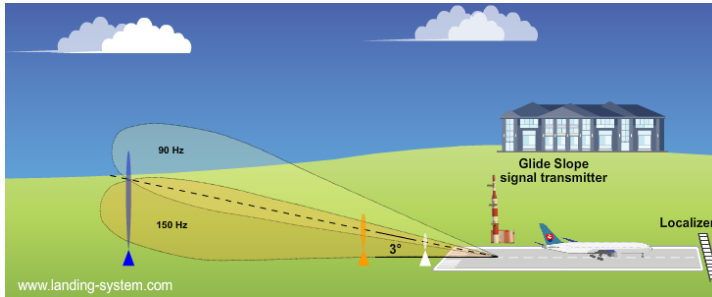


Figure 1.2: Illustration of the ILS with glide path signals and marker beam. The dotted line is the desired path towards the runway, [Landing-Systems].

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].

$$A_{\text{total}} = A_{\text{direct}} + A_{\text{reflected}} \quad (2.1)$$

For calculations of the plane earth loss¹, 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

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

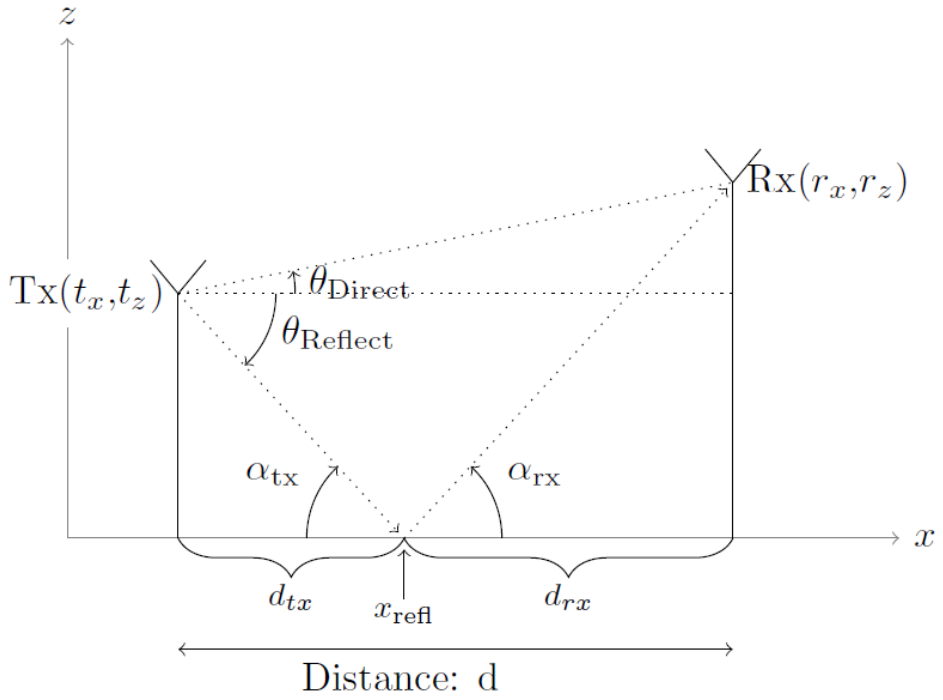


Figure 2.1: Notation for analytical model for plane earth loss.

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.

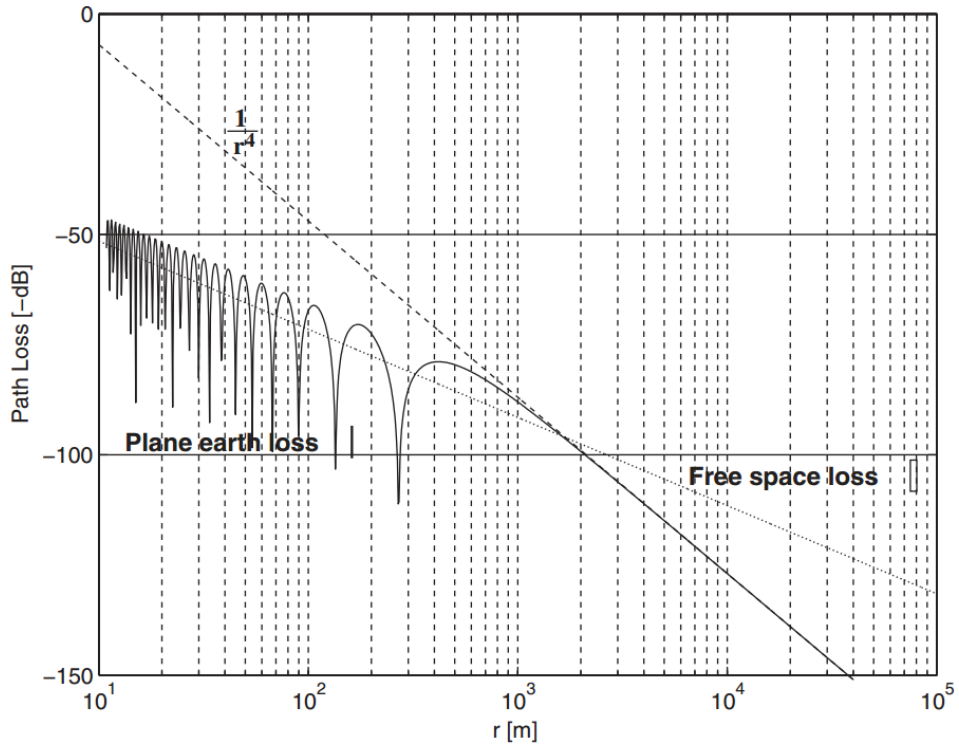


Figure 2.2: The solid line shows the plane earth loss: $f = 900\text{Hz}$, transmitter height: 30m , receiver height: 1.5m . r [m]: the distance along the surface between transmitter and receiver. [Saunders and Aragón-Zavala, 2007, p. 100]

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 direction¹. 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

¹Paraxial 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)

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

z : The height direction

$k = \frac{2\pi}{\lambda}$: Wavenumber, λ is the wavelength

(2.5)

θ_0 : The tilt of the beam [rad]

z_s : Antenna height

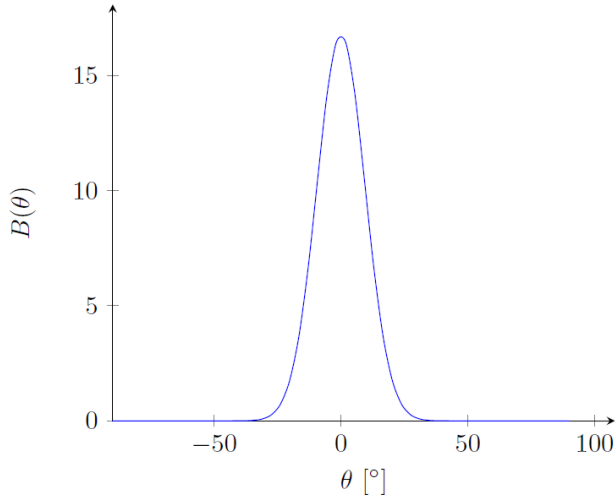


Figure 2.3: Gaussian beam, half-power beamwidth: 40°.

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,

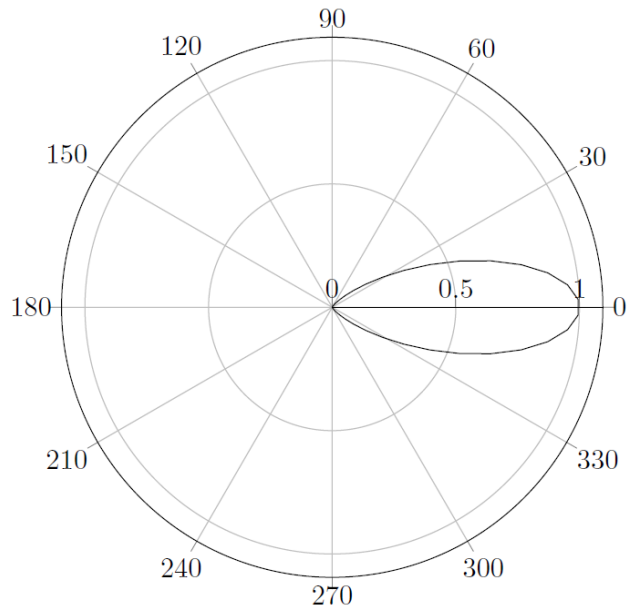


Figure 2.4: Polar plot of the Gaussian beam: half-power beamwidth at $\pm 20^\circ$. This is at the edge of the domain of validity for the numerical algorithm, it will be treated later.

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, $\mathbf{E}^s(\mathbf{r})$, 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, $\mathbf{E}^s(\mathbf{r})$, the field emitted from the induced surface currents, is found using the Maxwell's equations. A common way to calculate $\mathbf{E}^s(\mathbf{r})$, is to use an intermediate step via the auxiliary vector $\mathbf{A}(\mathbf{r})$, given by equation (3.1), [Gibson, 2007, p. 12]. $\mathbf{E}^s(\mathbf{r})$ 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.

$$\mathbf{A}(\mathbf{r}) = \mu \iint_S \mathbf{J}(\mathbf{r}') G(\mathbf{r}, \mathbf{r}') d\mathbf{r}'$$

$G(\mathbf{r}, \mathbf{r}')$: Electromagnetic Green's function, equation (3.3)
 $\mathbf{J}(\mathbf{r}')$: Induced surface current
 \mathbf{r}' : Vector from origin to the source point
 \mathbf{r} : Vector from origin to the observation point
 μ : Permeability of the medium

$$\mathbf{E}^s(\mathbf{r}) = -j\omega\mathbf{A} - \frac{j}{\omega\mu\epsilon} \nabla(\nabla \cdot \mathbf{A})$$

$\omega = 2\pi f$: Angular frequency
 ϵ : Permittivity of the medium

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

$$G(\mathbf{r}, \mathbf{r}') \simeq \begin{cases} 1 - j\frac{2}{\pi} \log \frac{\gamma kr}{2}, & r \rightarrow 0 \\ -\frac{j}{4} H_0^{(2)}(kr) = -\frac{j}{4} \sqrt{\frac{2j}{\pi kr}} \exp(-jkr) \exp(jk\mathbf{r}' \cdot \mathbf{r}), & kr \rightarrow \infty \end{cases}$$

γ : A constant

$k = \frac{2\pi}{\lambda}$: Wavenumber

$H_0^{(2)}(\cdot)$: Hankels function of second kind

$r = |\mathbf{r}' - \mathbf{r}|$: The distance between the source and the observation point

Equation (3.2) includes derivatives that depend on the distance \mathbf{r} . For large distances, where $kr \gg 1$, 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 $kr \gg 1$, 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).

$$\mathbf{E}^s(\mathbf{r}) = -j\omega\mathbf{A}$$

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 incidence¹, 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

¹Grazing 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]

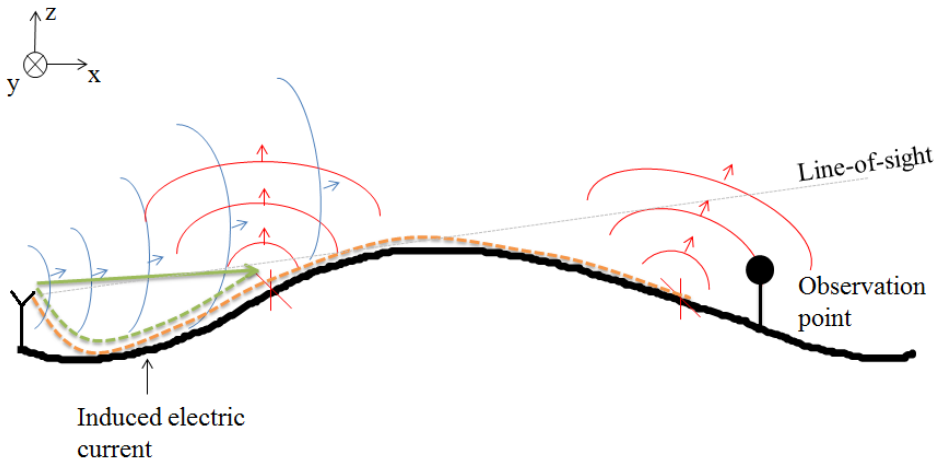


Figure 3.1: Illustration of the principle of the Integral Equation Method.

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

3. Numerical Methods

point n. R_1 , R_2 , r_1 and r_2 in formula (3.5) are according to figure 3.2.

$$M_n^s = TM_{i,n}^s + \frac{T}{4\pi} \sum_{m=0}^{n-1} M_m^s f(n, m) \Delta x_m,$$

$$f(n, m) = (\vec{n} \cdot \vec{r}_2) \frac{jk}{R_2} \sqrt{\lambda \frac{R_1 R_2}{R_1 + R_2}} e^{-j(kR_2 + \pi/4)} \frac{\Delta l_m}{\Delta x_m}$$

$k = \frac{2\pi}{\lambda}$: 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)

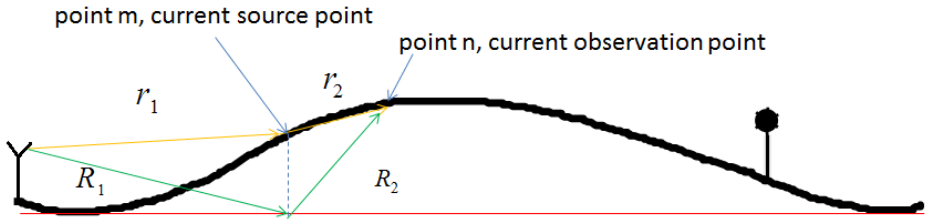


Figure 3.2: Geometry of the scattering problem for the Integral Equation Method.

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).

$$\begin{aligned} \mathbf{E}^s(\mathbf{r}) &= -j\omega\mu \int_S G(\mathbf{r}, \mathbf{r}') \mathbf{J}(\mathbf{r}') d\mathbf{r}' \\ &= \begin{cases} -j\frac{\mu}{4} \int_S \mathbf{J}(\mathbf{r}') \left[1 - j\frac{2}{\pi} \log \frac{\gamma k r}{2} \right] d\mathbf{r}', & \text{near-field} \\ -j\frac{\mu}{4} \int_S \mathbf{J}(\mathbf{r}') \sqrt{\frac{2j}{\pi k r}} e^{-jk r} e^{jk \mathbf{r}' \cdot \mathbf{r}} d\mathbf{r}', & \text{far-field} \end{cases} \end{aligned} \quad (3.6)$$

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

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

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). \mathbf{J}_n is a vector containing the induced

surface currents, and V_m a vector containing the incident electric fields.

$$\begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} V_m = \begin{bmatrix} \ddots & & & & \\ - & \ddots & & & 0 \\ - & Z_{mn} & \ddots & & \\ - & - & - & \ddots & \\ \vdots & - & - & - & \ddots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} J_n \quad (3.8)$$

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, ρ_1, \dots, ρ_D , see figure 3.3.

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

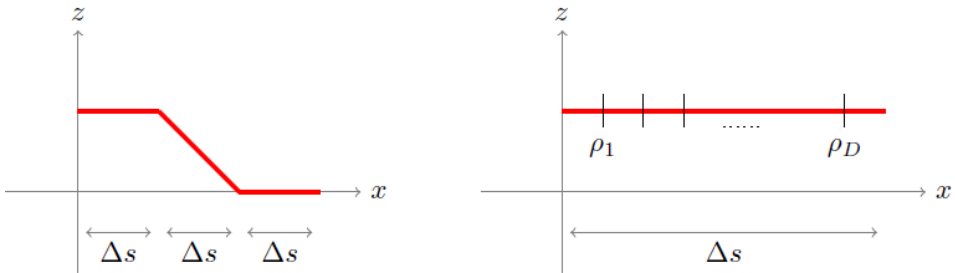


Figure 3.3: The surface is divided into D sections of width Δs . Each of the D sections are described by D uniformly spaced points within each section.

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 .

$$\begin{aligned} \text{Near-field region: } R &< 2\frac{D^2}{\lambda} \\ \text{Far-field region: } R &> 2\frac{D^2}{\lambda} \end{aligned} \tag{3.9}$$

R : Distance from radiation source

D : Largest dimension of radiation source

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

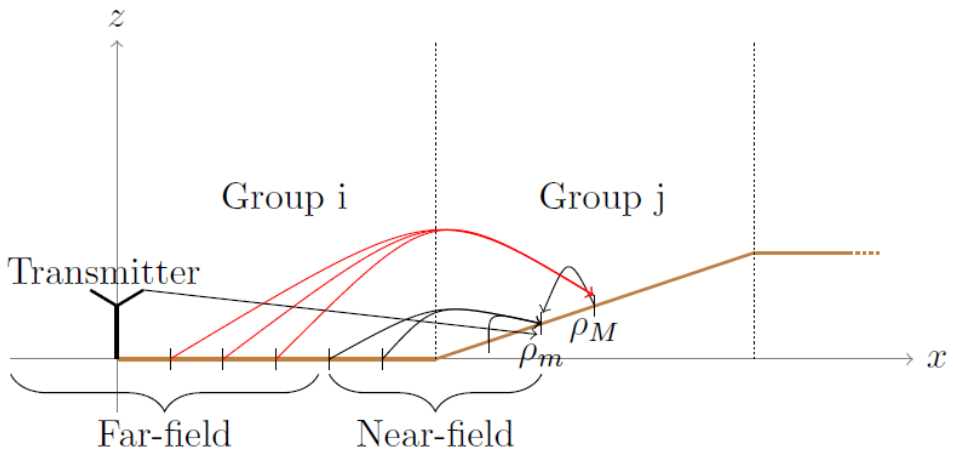


Figure 3.4: The induced surface current at a given point ρ_m depends on the direct field from the transmitter, and the near- and far-field from previous scattering points.

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

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

J : Vector of basisfunction coefficients.

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

F_{Mm}^i : 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).

$$Z = \left(\begin{array}{cccc|cccc|c}
 \hline
 Z_{11} & 0 & 0 & 0 & & & & & 0 \\
 \vdots & \ddots & 0 & 0 & & & & & \\
 \vdots & & \ddots & 0 & & & & & \\
 Z_{D1} & \cdots & \cdots & Z_{Dk} & & & & & \\
 \hline
 Z_{MFF_j} & Z_{MFF_j} & Z_{mx_l NF} & Z_{mx_k NF} & Z_{mn} & 0 & 0 & 0 & \\
 Z_{MFF_j} & Z_{MFF_j} & Z_{Mx_l NF} & Z_{Mx_k NF} & Z_{Mn} & \text{Group } j & & 0 & \\
 Z_{MFF_j} & Z_{MFF_j} & \cdots & \cdots & \cdots & & \cdots & 0 & \\
 Z_{MFF_j} & Z_{MFF_j} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\
 \hline
 \end{array} \right)$$

Green: Far-field of group j

Red: Near-Field of Z_m

(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).

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

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

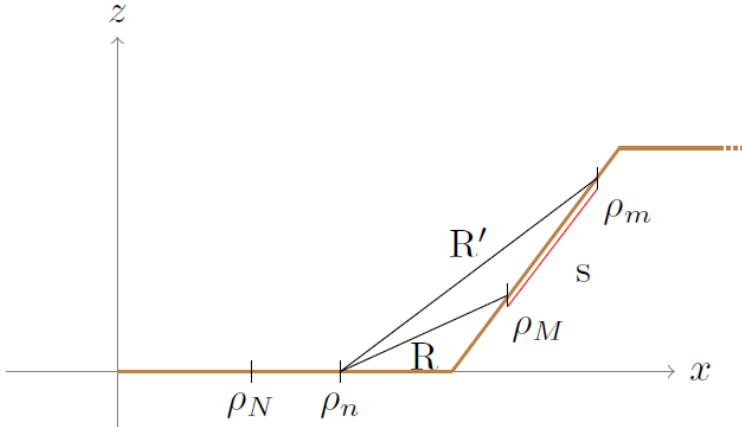


Figure 3.5: Geometrical aid for equation (3.12).

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 look like the illustration in equation (3.13).

$$Z = \begin{pmatrix}
 \begin{array}{ccc|ccc}
 Z_{11} & 0 & 0 & & & \\
 \vdots & \text{Group i} & 0 & & & \\
 Z_{D1} & \cdots & Z_{Dk} & & & \\
 \hline
 & & & \begin{array}{ccc}
 \ddots & 0 & 0 \\
 \cdots & \text{Group j} & 0 \\
 \cdots & \cdots & \cdots
 \end{array} & & & \\
 & & & & & & \\
 & & & & & & \begin{array}{ccc}
 Z_{mn} & 0 & 0 \\
 Z_{Mn} & \text{Group k} & 0 \\
 \cdots & \cdots & \cdots
 \end{array} \\
 \hline
 Z_{MFF_k} & Z_{MFF_k} & Z_{mx_k NF} & & & \\
 Z_{MFF_k} & Z_{MFF_k} & Z_{Mx_k NF} & & & \\
 Z_{MFF_k} & Z_{MFF_k} & \cdots & & &
 \end{array}
 \end{pmatrix}
 \quad (3.13)$$

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

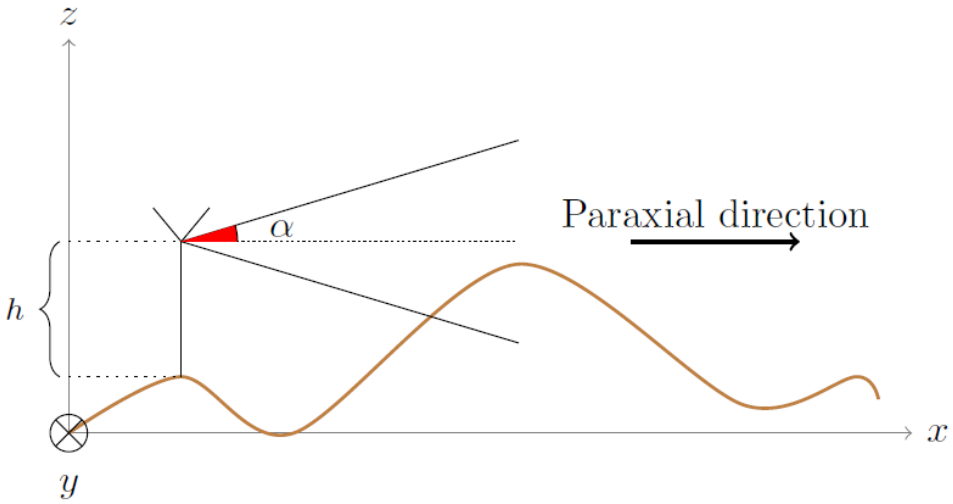


Figure 3.6: The setup for the Parabolic Equation Method.
 h : Transmitter antenna height.

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.

$$\begin{aligned}
 \nabla \times \mathbf{H} &= \mathbf{j}_c + \epsilon \frac{\partial \mathbf{E}}{\partial t} \\
 \nabla \times \mathbf{E} &= -\mu \frac{\partial \mathbf{H}}{\partial t} \\
 \nabla \cdot \mathbf{E} &= \frac{\rho}{\epsilon} \\
 \nabla \cdot \mathbf{H} &= 0 \\
 \mathbf{E} &= \mathbf{E}(x, y, z; t): \text{ Electric field} \\
 \mathbf{H} &= \mathbf{H}(x, y, z; t): \text{ Magnetic field}
 \end{aligned} \tag{3.14}$$

\mathbf{j}_c : Current density at the given point

ϵ : Absolute permittivity

μ : Absolute permeability

ρ : Charge density at the given point

Kouzaev [2011]

3. Numerical Methods

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

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

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

$$\begin{aligned}\nabla \times \mathbf{E} &= -\mu \frac{\partial \mathbf{H}}{\partial t} \\ &= -j\omega\mu\mathbf{H}\end{aligned}\quad (3.16)$$

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

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

Using vector operator identity gives $\nabla \times \nabla \times \mathbf{E} = \nabla\nabla \cdot \mathbf{E} - \nabla^2\mathbf{E}$. Since there is no source at the given point, $\rho = 0$. Therefore, $\nabla \cdot \mathbf{E} = 0$ and $\nabla\nabla \cdot \mathbf{E} = 0$. 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).

$$\begin{aligned}\nabla^2\mathbf{E} &= -\omega^2\mu\epsilon\mathbf{E} \\ \Leftrightarrow e^{(j\omega t)}\nabla^2\mathbf{E}_0 &= -\omega^2\mu\epsilon\mathbf{E}_0e^{(j\omega t)} \\ \Rightarrow \nabla^2\mathbf{E}_0 &= -\omega^2\mu\epsilon\mathbf{E}_0 \\ \Rightarrow \frac{\partial^2\mathbf{E}_0}{\partial x^2} + \frac{\partial^2\mathbf{E}_0}{\partial y^2} + \frac{\partial^2\mathbf{E}_0}{\partial z^2} + \omega^2\mu\epsilon\mathbf{E}_0 &= 0 \\ \text{Accounting for 2D propagation} \Rightarrow \frac{\partial^2\mathbf{E}_0}{\partial x^2} + \frac{\partial^2\mathbf{E}_0}{\partial z^2} + \omega^2\mu\epsilon\mathbf{E}_0 &= 0\end{aligned}\quad (3.18)$$

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), \mathbf{E}_0 , is a

scalar.

For propagation in vacuum: $\epsilon_r = \mu_r = 1$

The speed-of-light in vacuum: $c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}$ [Lee, 2013, p. 390]

The speed-of-light in a given medium: $v = \frac{c}{\sqrt{\mu_r \epsilon_r}}$ [Lee, 2013, p. 389]

The index of refraction: $n = \frac{c}{v} = \sqrt{\mu_r \epsilon_r}$ (3.19)

$$\Rightarrow \omega^2 \epsilon \mu = \frac{\omega^2}{c^2} n^2 = \left(\frac{2\pi}{\lambda} \right)^2 n^2 = k^2 n^2$$

$$\Rightarrow \frac{\partial^2 \mathbf{E}_0}{\partial x^2} + \frac{\partial^2 \mathbf{E}_0}{\partial z^2} + k^2 n^2 \mathbf{E}_0 = 0$$

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 $e^{j\omega t}$. However, the Parabolic Equation Method deals with how the field propagates in space, not in time.

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial z^2} + k^2 n^2 \psi = 0$$

x : Direction of propagation

z : Height with respect to the coordinate system (3.20)

k : Wave number in vacuum, $\frac{2\pi}{\lambda}$

n : Refractive index, function of x and z , slowly varying

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, $u(x, z) = e^{jkx}\psi(x, z)$ 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.

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

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

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 $\alpha = 20^\circ$, $\sin^2(\alpha) \simeq 0.12$. From $\alpha = 20^\circ$ and up, the error increases relatively fast.

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

α : angle according to figure 3.6

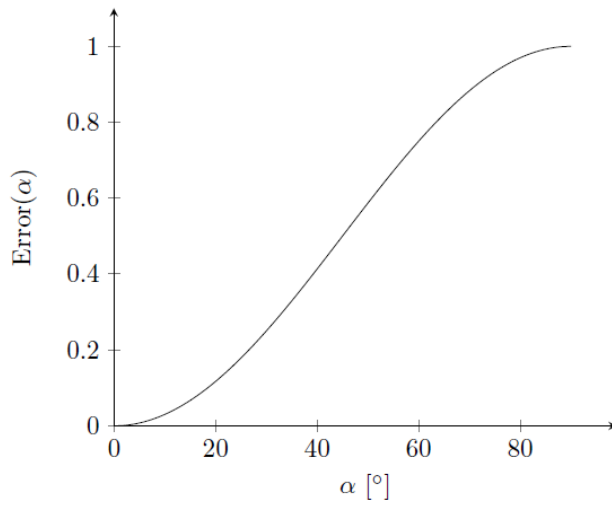


Figure 3.7: The error due to approximation of square-root operator as a function of the angle from the paraxial direction.

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).

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

$u(x, z)$ 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].

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

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].

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

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

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.

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

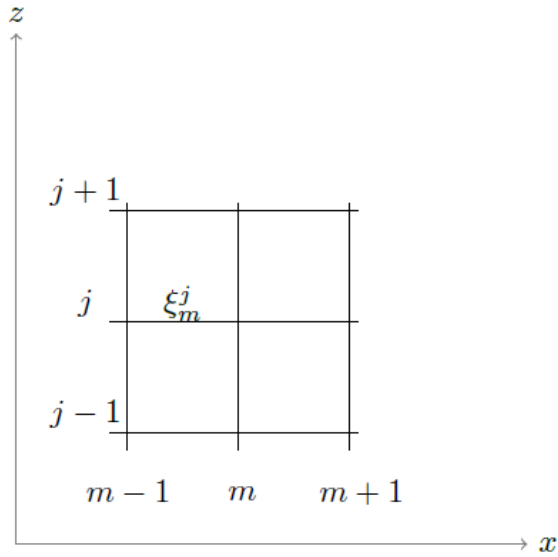


Figure 3.8: The geometry for the Finite-Difference Method.

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 $\beta_m^j = 2 + b - a_j^m$. 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.

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

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

(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 .

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

(3.33)

The simulated results will not be valid unless equation (3.33) converges as $m \rightarrow \infty$. 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 stable¹ 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.

¹Neutral 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 α s is denoted A , and the tri-diagonal matrix containing the β s is denoted B . This results in equation (3.34), provided that A can be inverted.

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

A matrix M can be inverted if all the eigenvalues, $\lambda_m \neq 0$, if A is non-singular. 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 diagonalized, 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 .

$$C = SAS^{-1} \tag{3.35}$$

$$\begin{aligned} C^m &= (SAS^{-1}) \dots (SAS^{-1}) \\ &= SA^m S^{-1} \end{aligned} \tag{3.36}$$

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)

$$\left[\begin{array}{cccc} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_{N-1} \\ & & & & \lambda_N \end{array} \right]^m = \left[\begin{array}{cccc} \lambda_1^m & & & \\ & \lambda_2^m & & \\ & & \ddots & \\ & & & \lambda_{N-1}^m \\ & & & & \lambda_N^m \end{array} \right] \tag{3.37}$$

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.

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

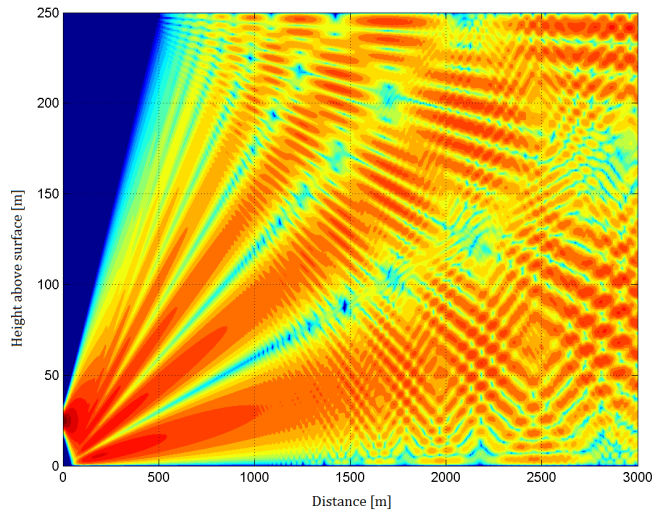
The value of σ_0 is chosen to $\sigma_0 = 0.0015$. σ_0 is limited by an upper and lower boundary. The upper limit is set by the criteria that the waves should not be damped too fast, because it leads to unwanted reflections. The lower limit is set

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.

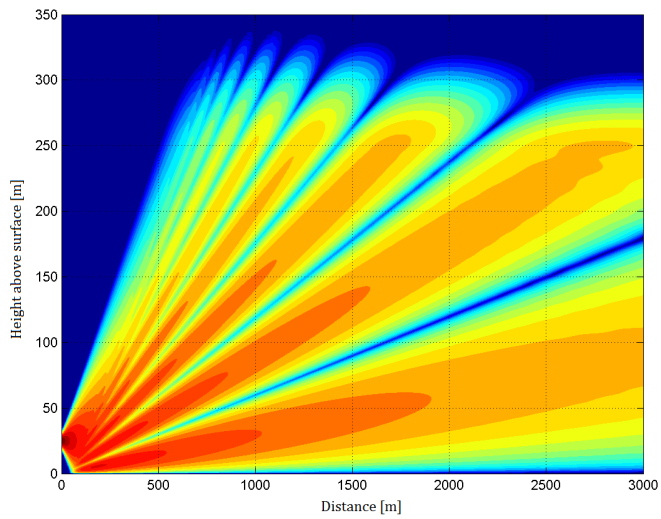
The height of the computational domain and the number of points in the absorption layer can be varied according to the problem. Figure 3.9 shows two examples of the same propagated field, one without and one with absorption layer, figure 3.9a and 3.9b, respectively. The absorption layer clearly damps 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

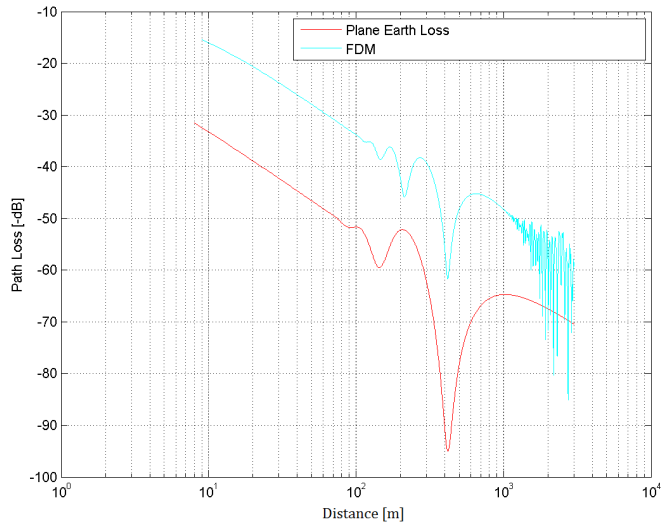


(a) No absorption layer at the top.

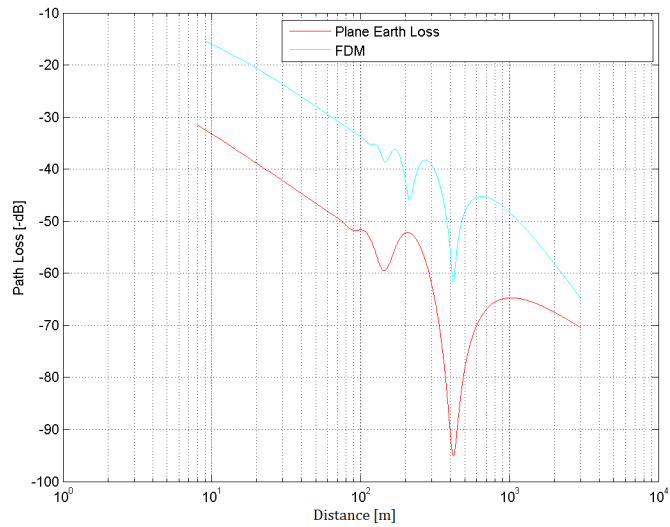


(b) Absorption layer at the top.

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.

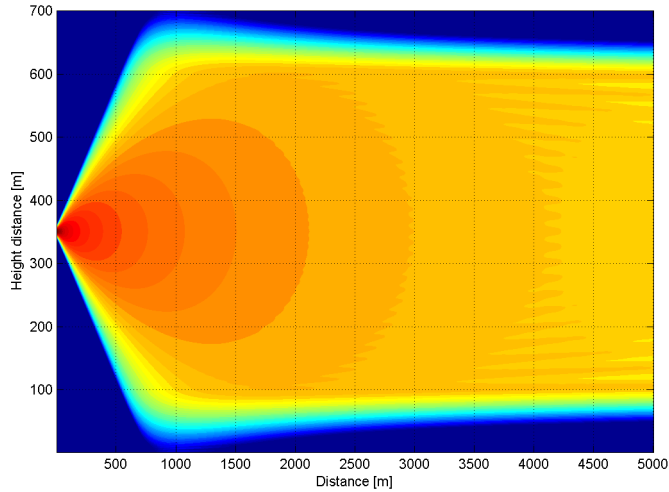


Figure 3.11: Field propagating in free-space with absorption layer at the top and bottom.

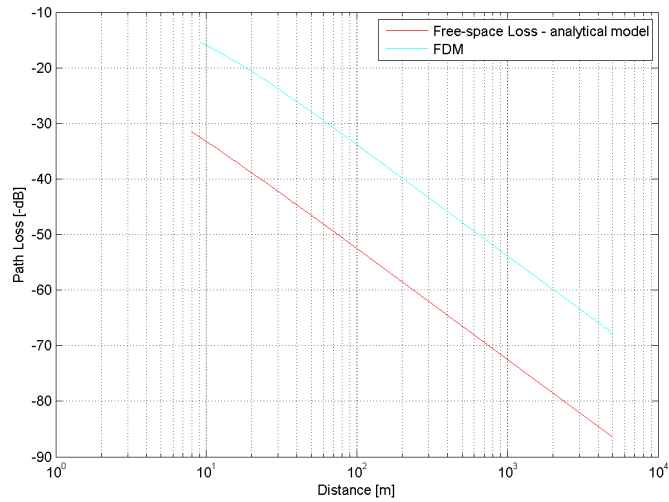


Figure 3.12: Comparison between analytical free-space loss and simulated free-space loss using absorption layer at the top and bottom, same field as in figure 3.11. The comparison is taken at the height of the center of the source. "FDM" is the simulated field. In free-space, path loss is the same as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

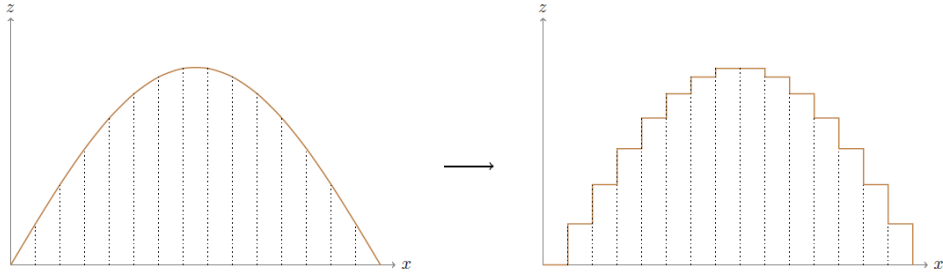


Figure 3.13: Irregular terrain modeled using the staircase model.

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

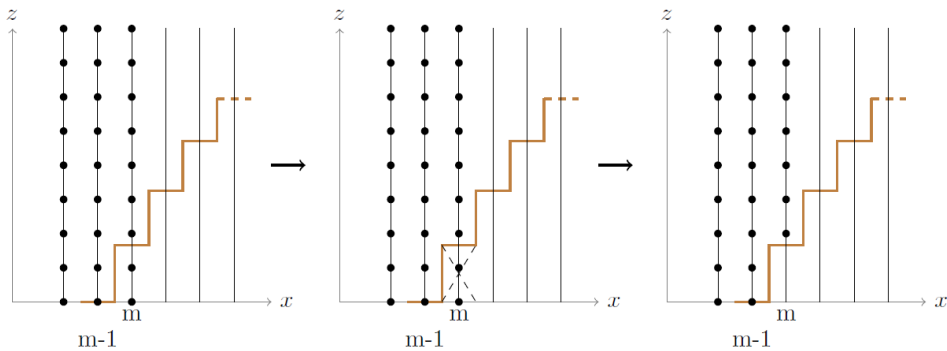


Figure 3.14: Principle of the algorithm for calculation of field in ascending terrain. The dots illustrate the field calculation points.

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.

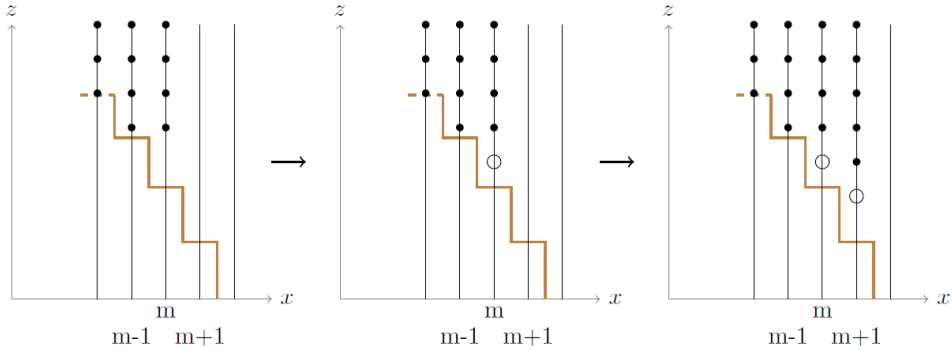


Figure 3.15: Principle of the algorithm for calculation of the electromagnetic field in descending terrain. The dots illustrate the field calculation points, the dots without fill represent the padded dots where the value is zero.

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.

$$L = \frac{P_{TI}}{P_{RI}} \tag{3.39}$$

P_{TI} : Effective isotropic transmitter power

P_{RI} : Effective isotropic receiver power

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).

$$\frac{P_r}{P_t} = \left(\frac{\lambda}{4\pi d} \right)^2 \left| 1 + \exp \left(jk \frac{2h_m h_b}{d} \right) \right|^2 \tag{3.40}$$

d : Distance between the transmitter and receiver along the surface

h_b : Height of the transmitter

h_m : Height of the receiver

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.

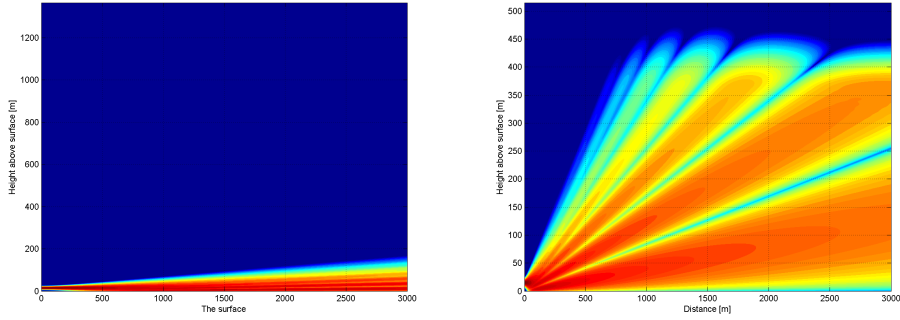
Chapter 4

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

4. Choice of Parameters



(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.

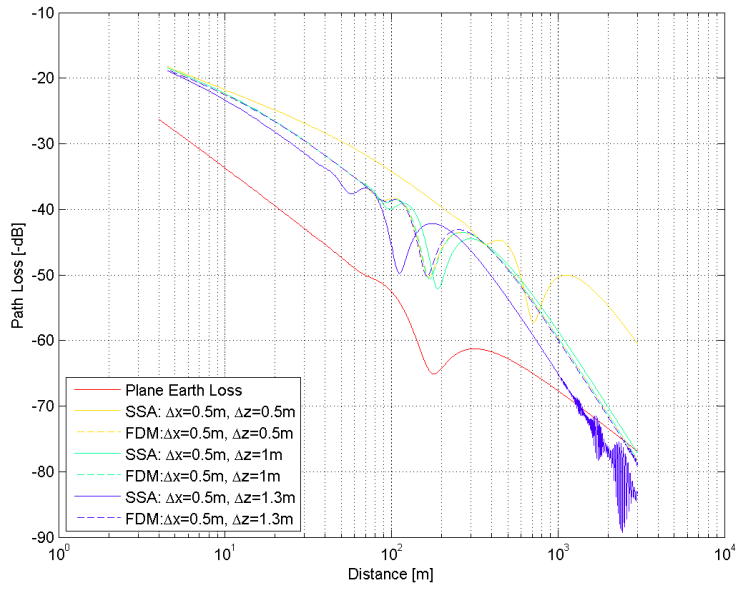


Figure 4.2: Path loss comparison at the antenna height, 15 m, along the surface.

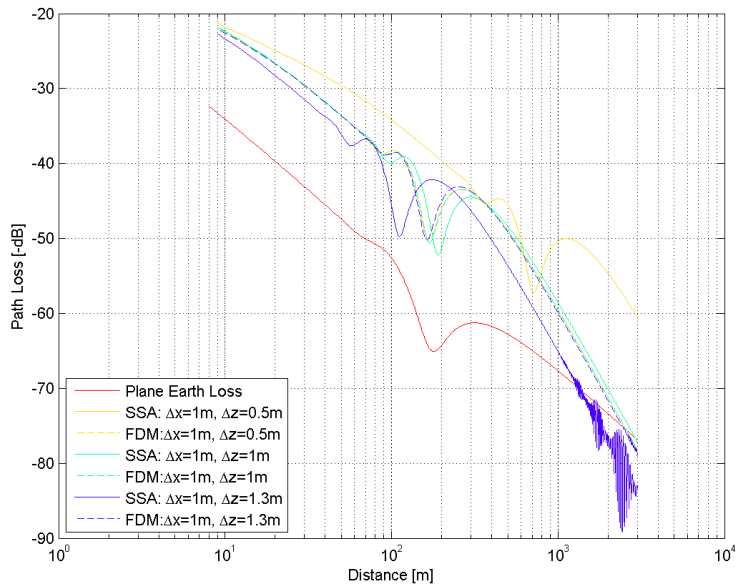


Figure 4.3: Path loss comparison at the antenna height, 15 m, along the surface.

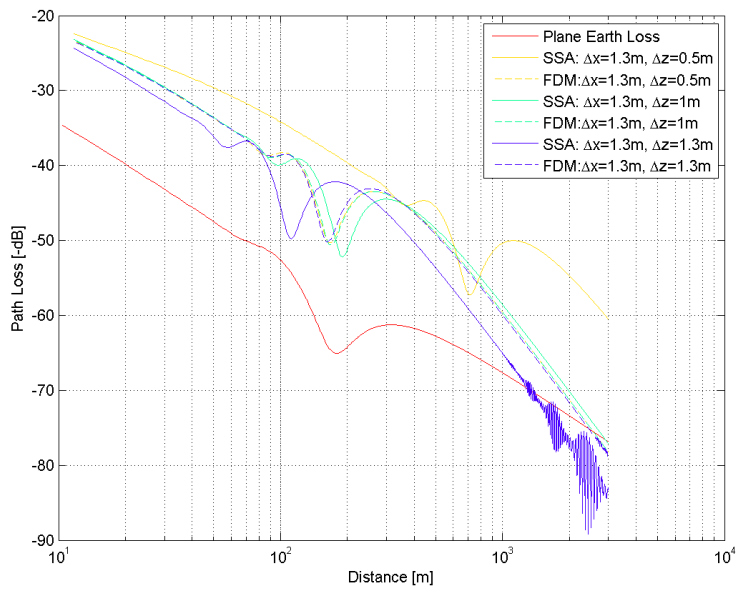


Figure 4.4: Path loss comparison at the antenna height, 15 m, along the surface.

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 \pm -sign, or the total half-power beamwidth, without the \pm -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 $\pm 20^\circ$ 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 $\alpha > 20^\circ$ the error will theoretically increase rapidly, see figure 3.7. It is therefore desirable to have a beamwidth of maximum $\pm 20^\circ$.

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-file¹ attached to the thesis.

¹The 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° , $\pm 27.5^\circ$, 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

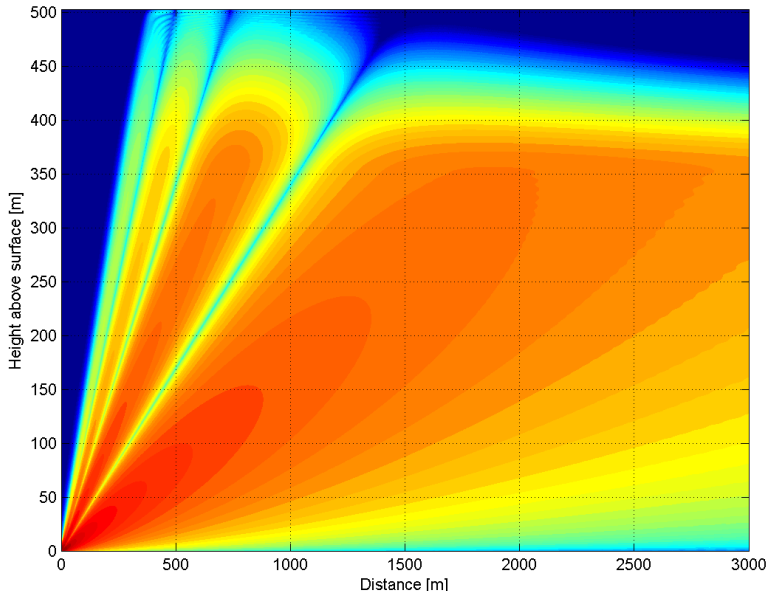


Figure 5.2: Field simulation using the SSA.

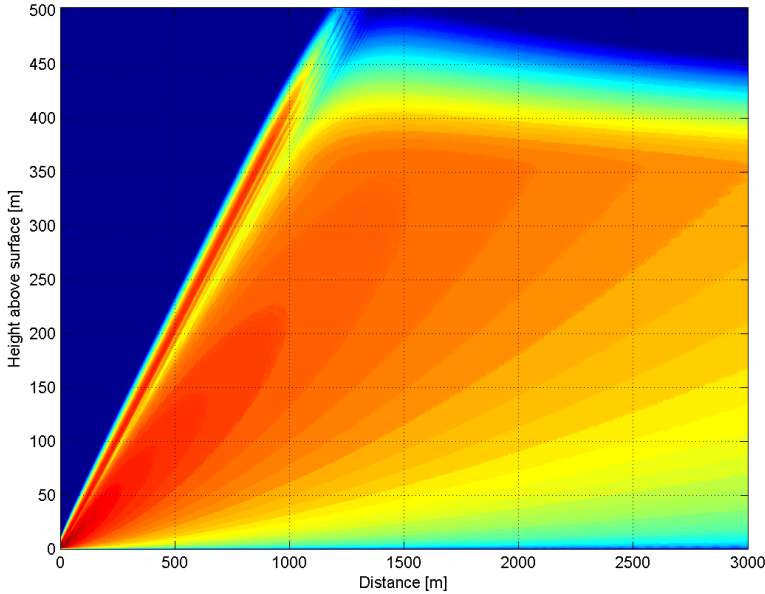


Figure 5.3: Field simulation using the FDM.

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.

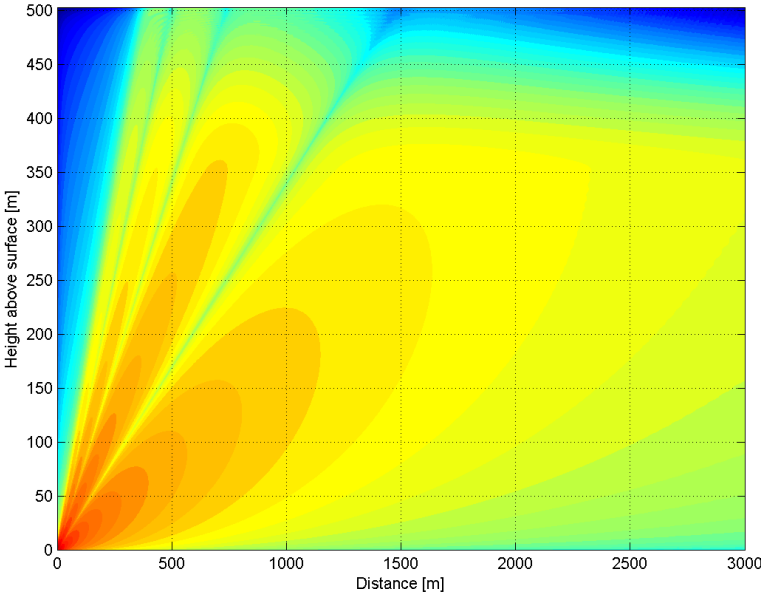


Figure 5.4: Field simulation using the SSA, $\frac{1}{r}$ -loss added.

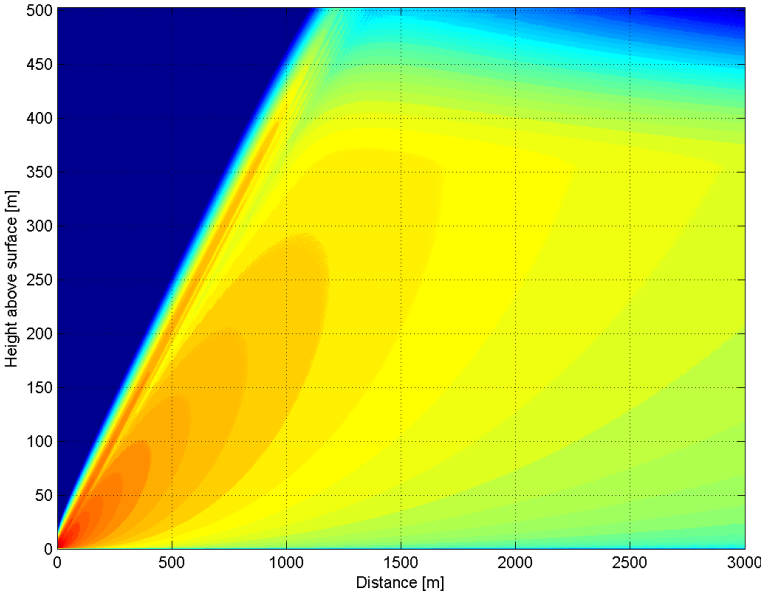


Figure 5.5: Field simulation using the FDM, $\frac{1}{r}$ -loss added.

5. Results

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.

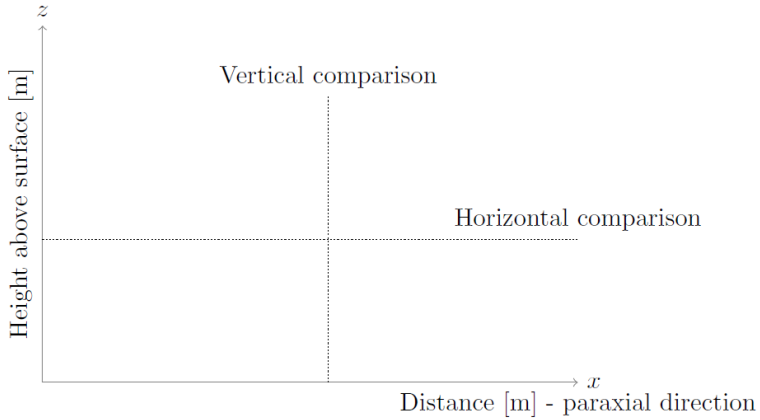


Figure 5.6: Illustration of horizontal and vertical comparison.

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.

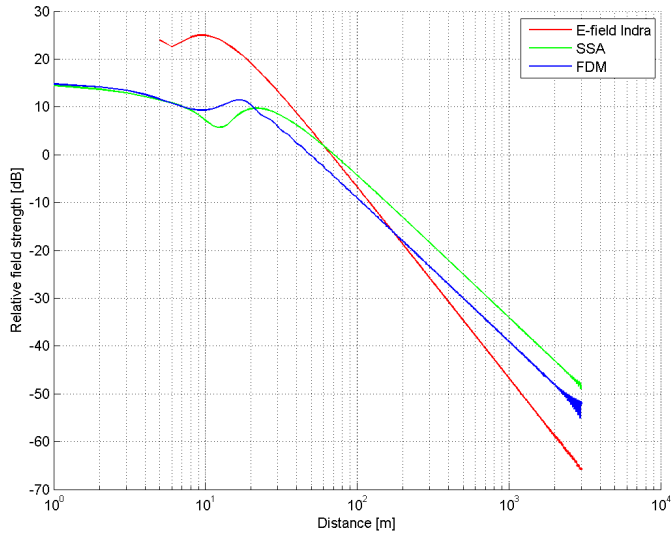


Figure 5.7: Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. No additional loss.

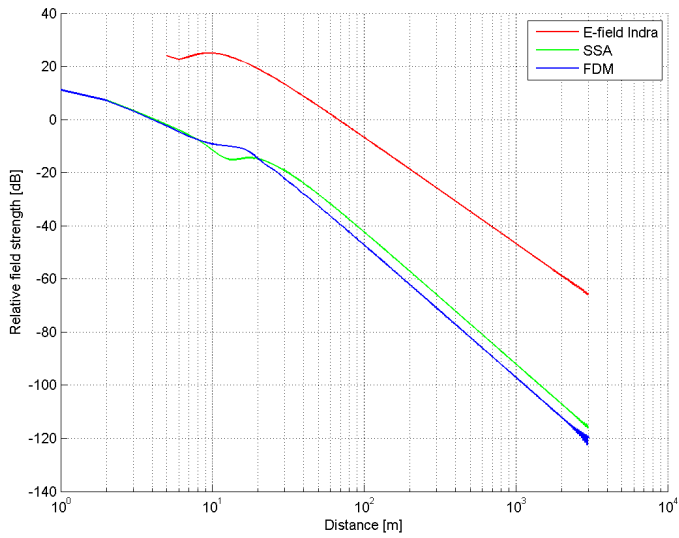


Figure 5.8: Horizontal comparison, relative field strengths at the height of 3 m, along a flat surface. $\frac{1}{r}$ -loss added.

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.

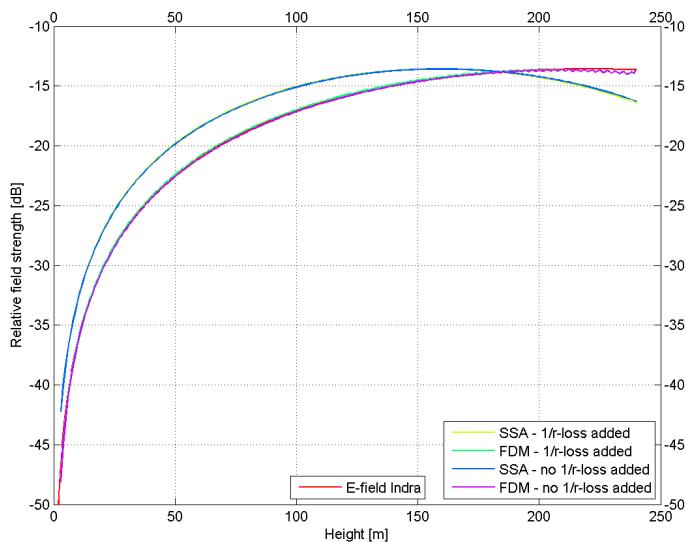


Figure 5.9: Relative field strength at the distance of 1000 m along a flat surface. Both SSA algorithms overlap each other, and both FDM algorithms overlap each other.

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

5. Results

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.

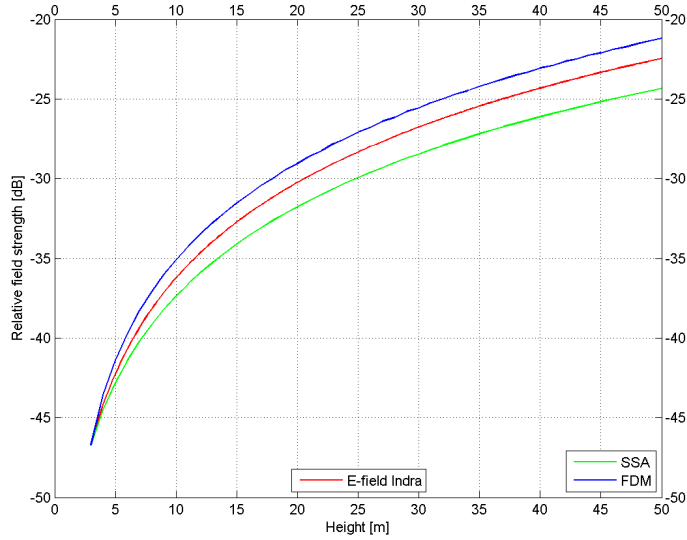


Figure 5.10: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. No $\frac{1}{r}$ -loss added. Aligned at the lowest height.

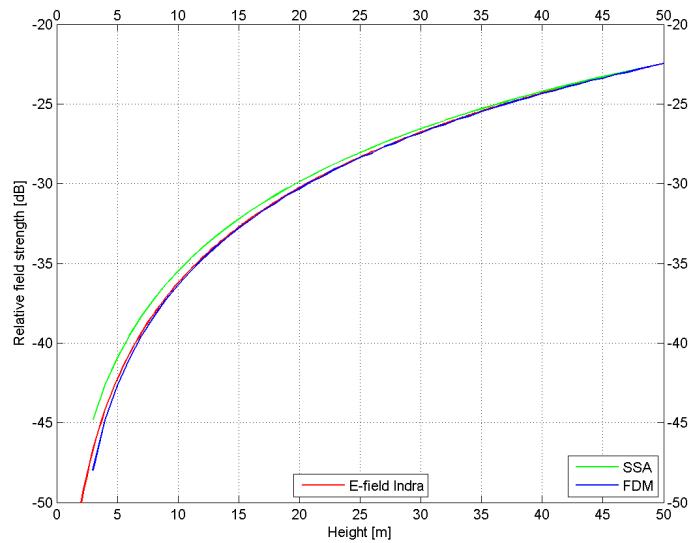


Figure 5.11: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. No $\frac{1}{r}$ -loss added. Aligned at the maximum value.

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 ever 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.

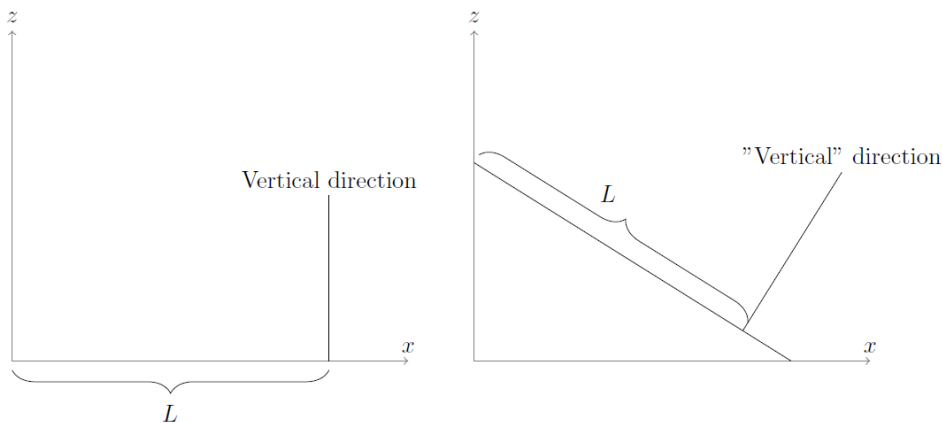


Figure 5.12: Flat surface and downwards inclined plane, field comparison in the vertical direction. The distance between transmitter and the line-of-comparison along the surface is L . The transmitter is assumed to be along the z -axis at the same height relative to the ground.

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.

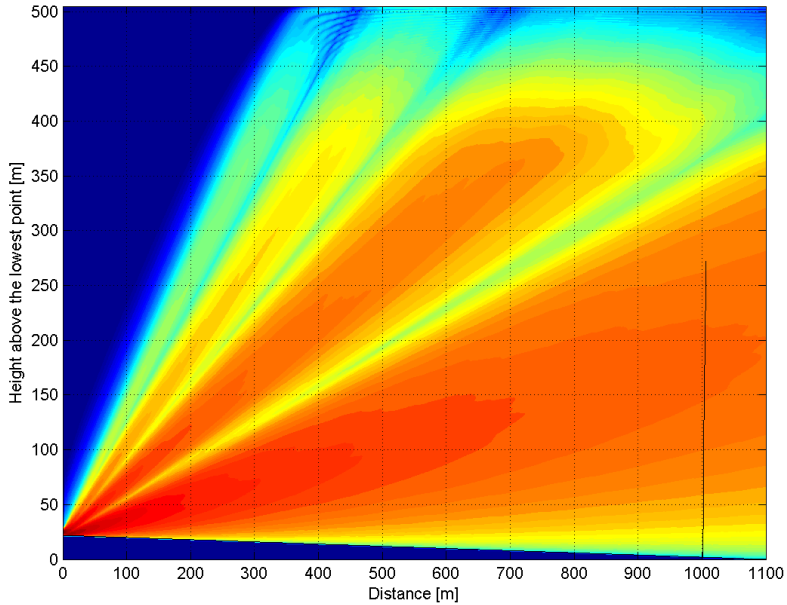


Figure 5.13: Field simulation using the SSA on a downwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

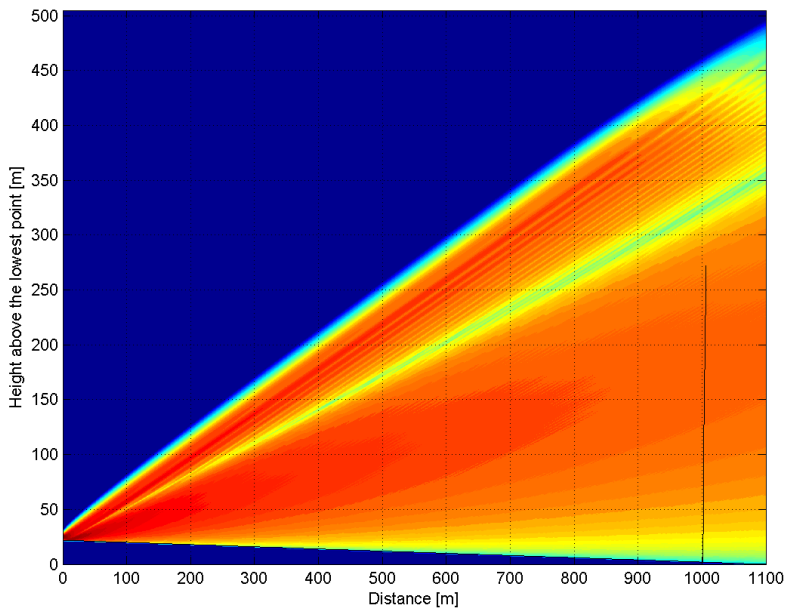


Figure 5.14: Field simulation using the FDM on a downwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

5. Results

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.

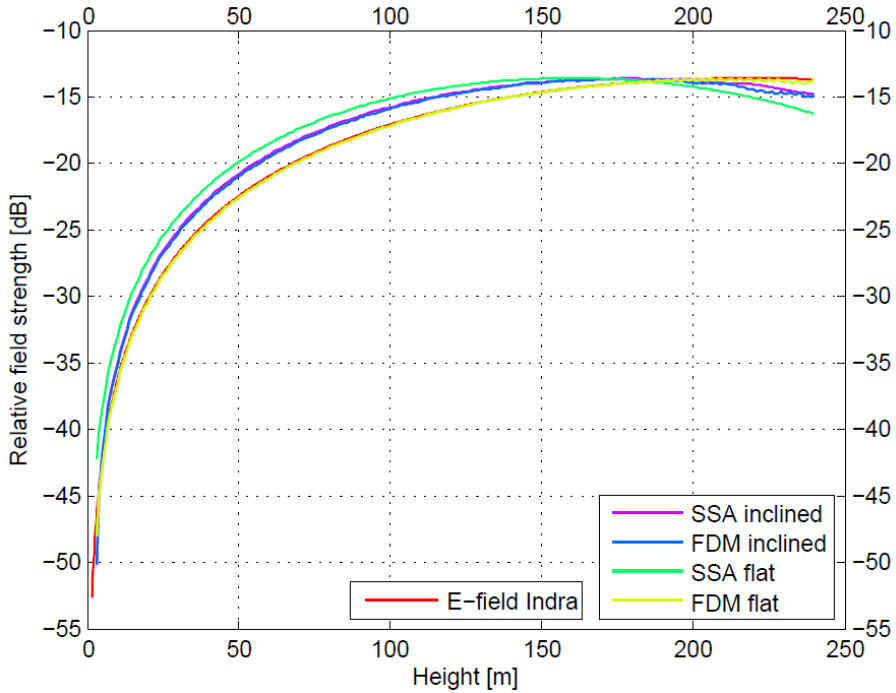


Figure 5.15: Vertical comparison of the relative field strength at the distance of 1000 m along a downwards inclined plane.

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.

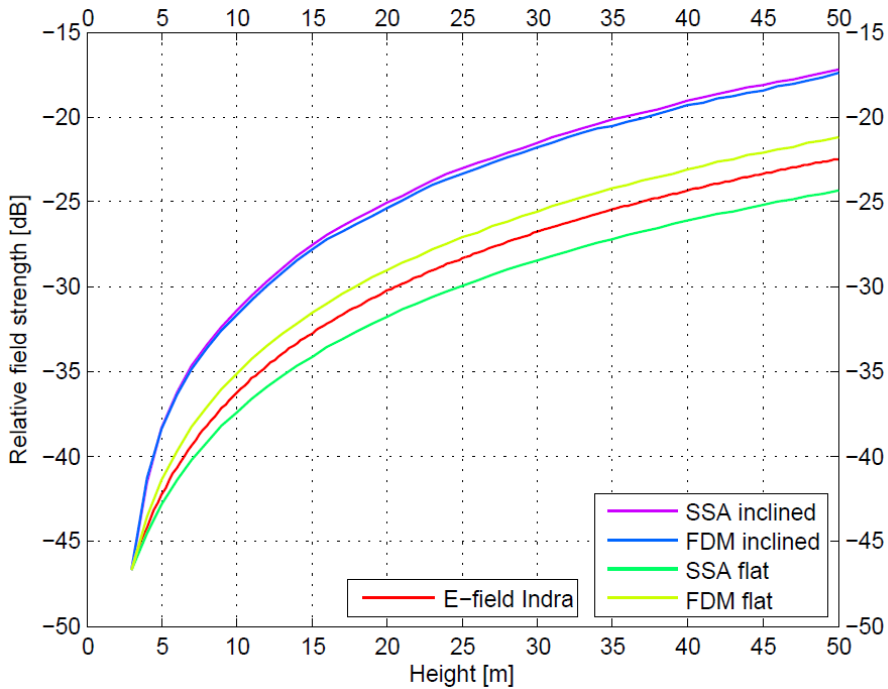


Figure 5.16: Vertical comparison of the relative field strength at the distance of 1000 m along a downwards inclined plane. The relative field strengths are aligned to the minimum value of the analytic field.

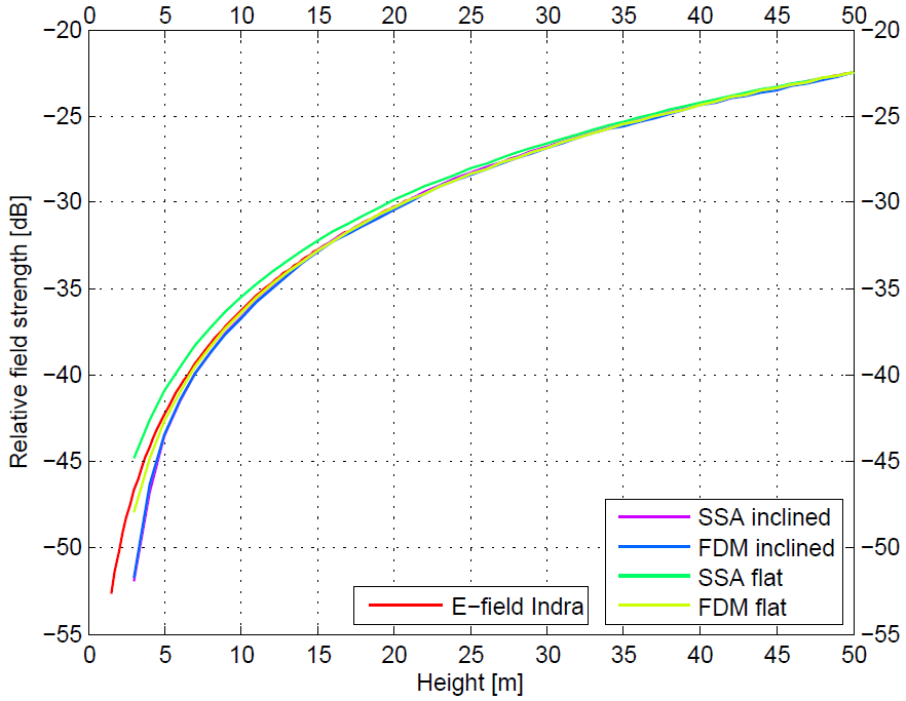
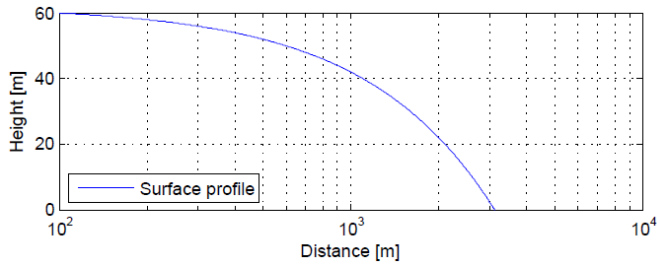
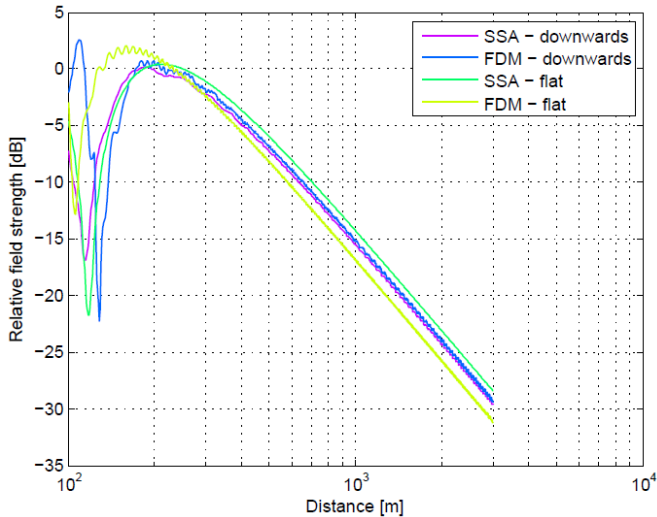


Figure 5.17: Relative field strength at the distance of 1000 m along a flat surface, the receiver height is varying. The relative field strengths are aligned to the maximum value of the analytic field.



(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.

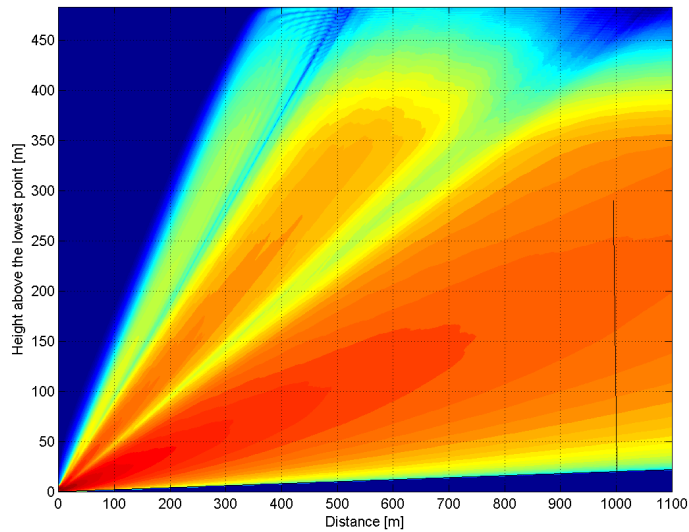


Figure 5.19: Field simulation using the SSA on an upwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

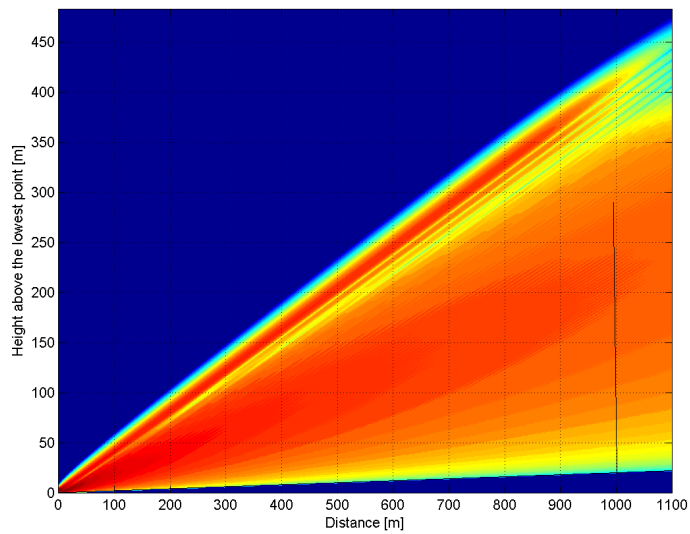


Figure 5.20: Field simulation using the FDM on an upwards inclined surface. The black line at 1000 m along the inclined plane is the "vertical" direction to the plane at this point.

5. Results

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.

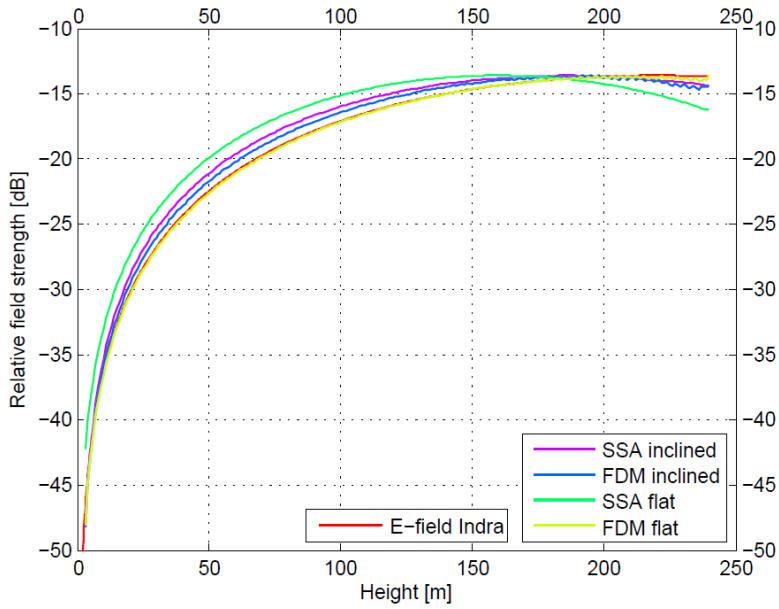


Figure 5.21: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane.

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.

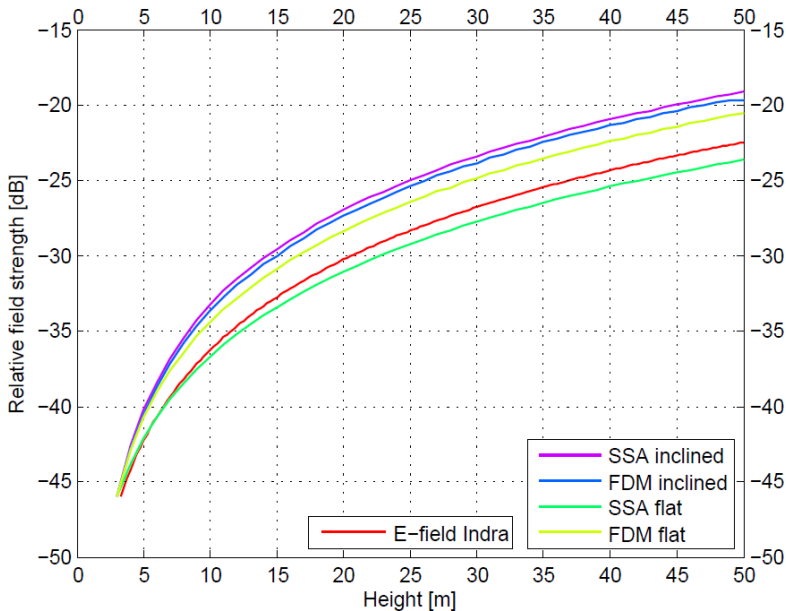


Figure 5.22: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane. The relative field strengths are aligned to the minimum value of the analytical field.

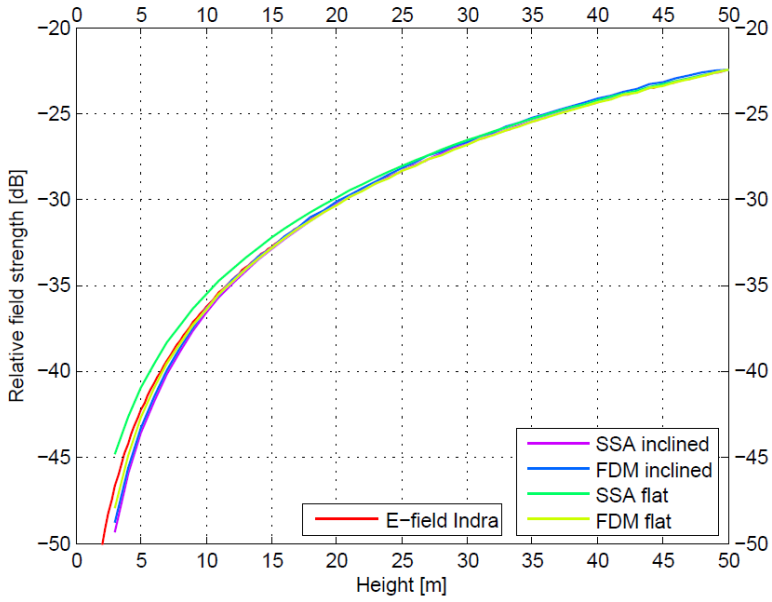
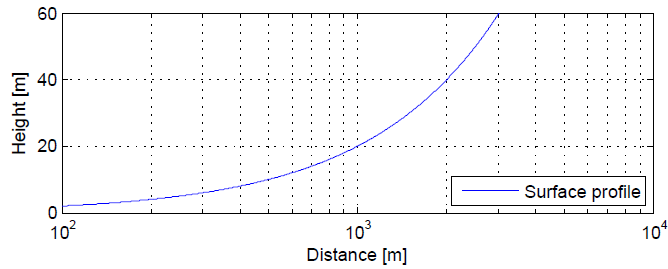
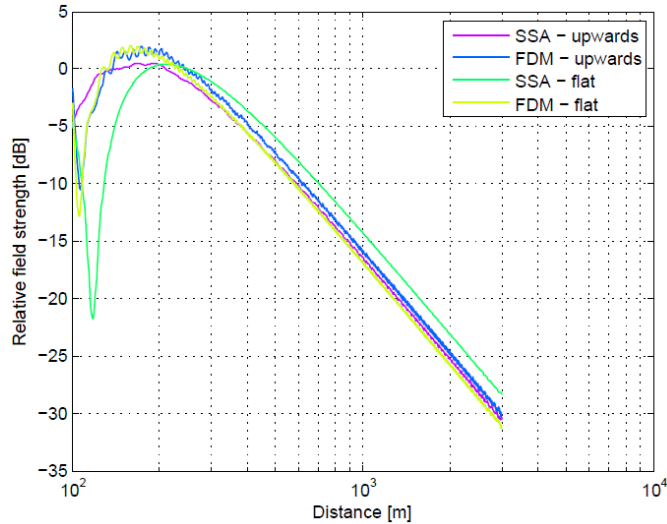


Figure 5.23: Vertical comparison of the relative field strengths at the distance of 1000 m along an upwards inclined plane. The relative field strengths are aligned to the maximum value of the analytic field.



(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 in 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

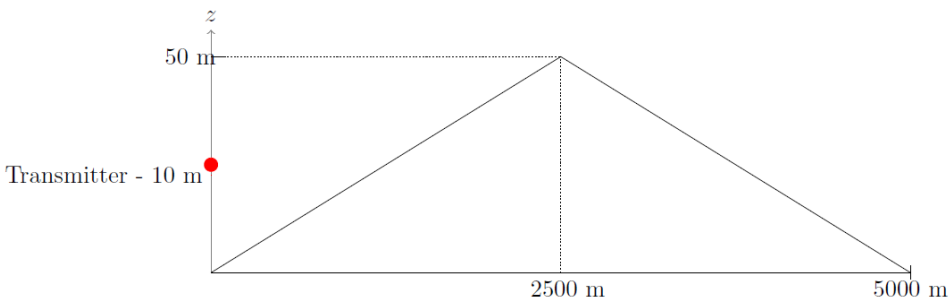


Figure 5.25: The setup for the wedge.

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.

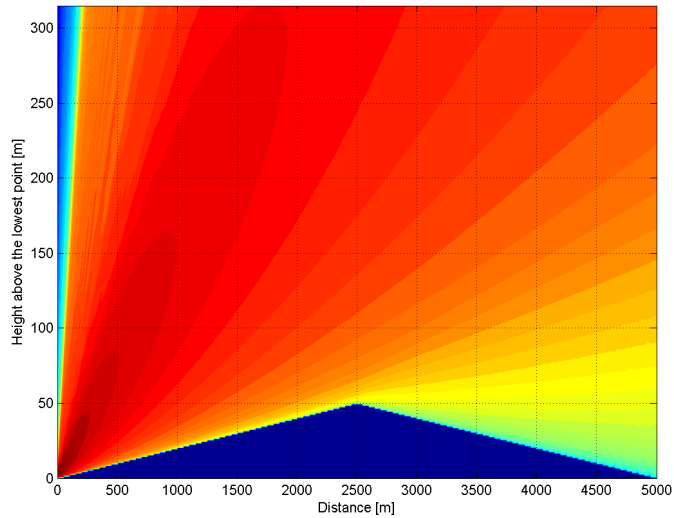


Figure 5.26: Field simulation over the wedge given in figure 5.25 using the SSA. Frequency: 100 MHz.

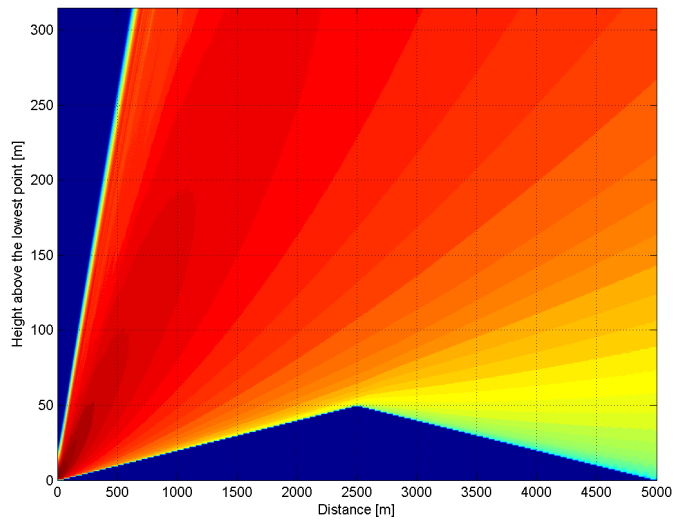
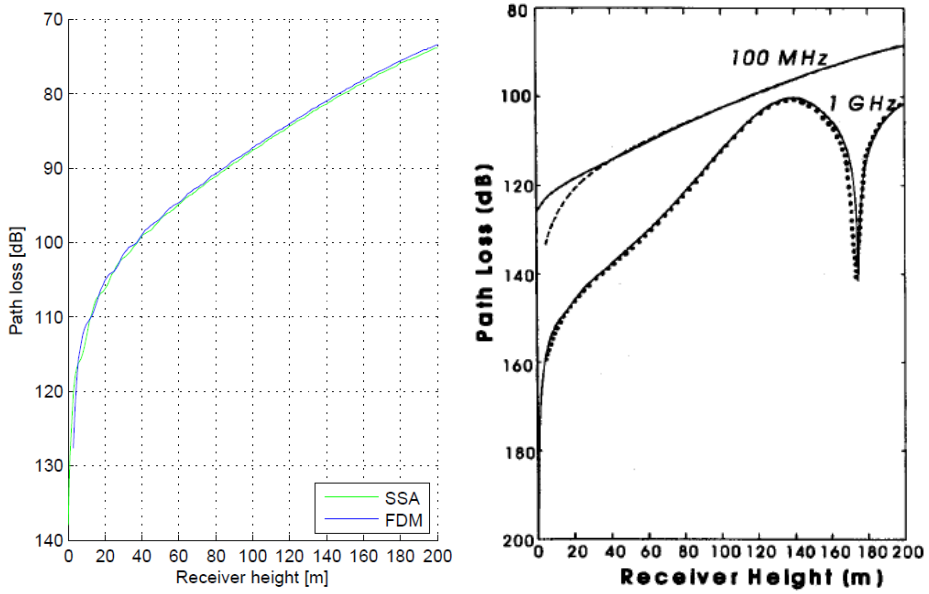


Figure 5.27: Field simulation over the wedge given in figure 5.25 using the FDM. Frequency: 100 MHz.

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.

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

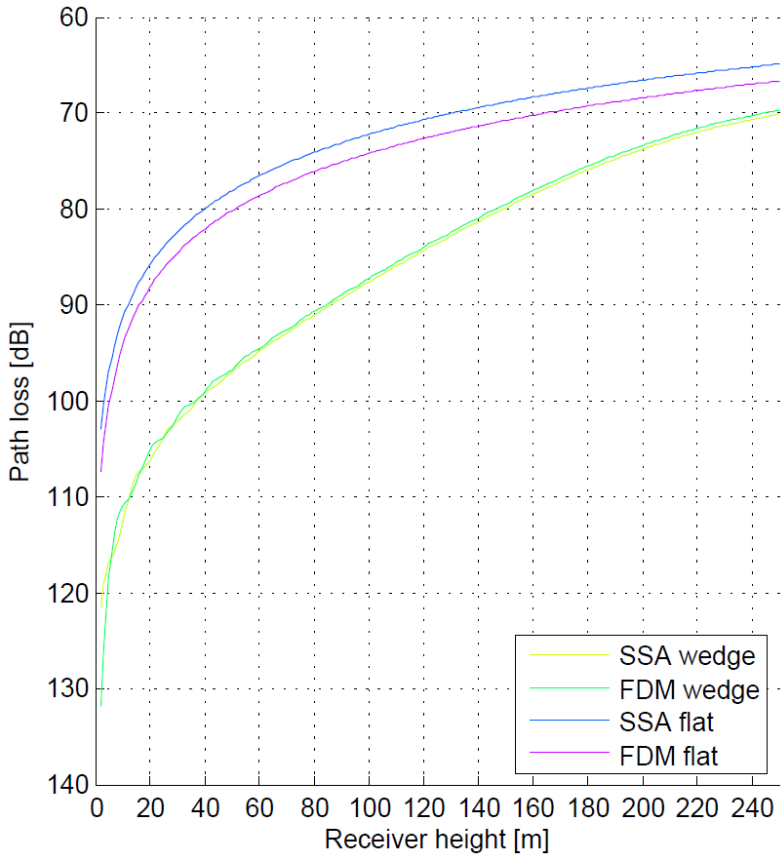
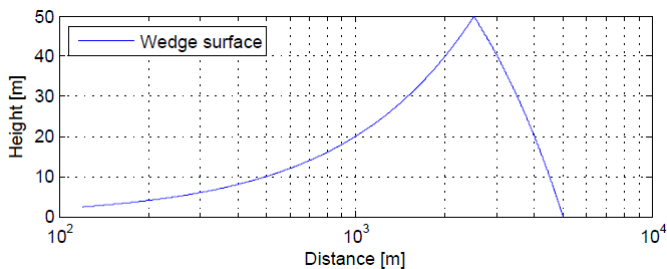
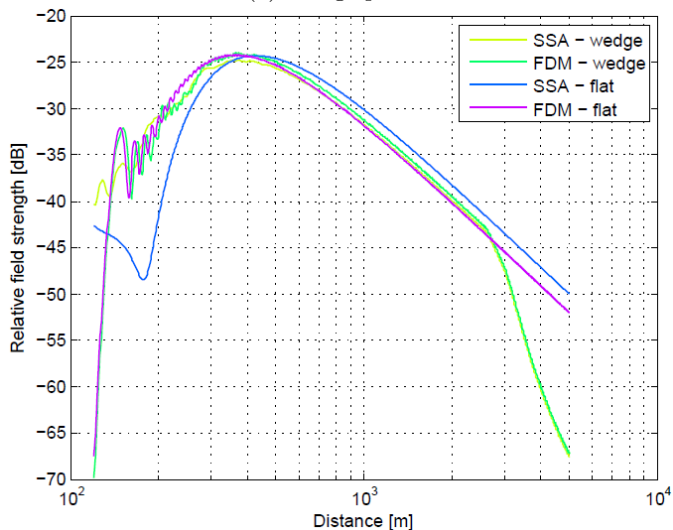


Figure 5.29: Vertical comparison of path loss of the field propagated along a flat surface and behind the wedge, at 5000 m.



(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 undulating 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

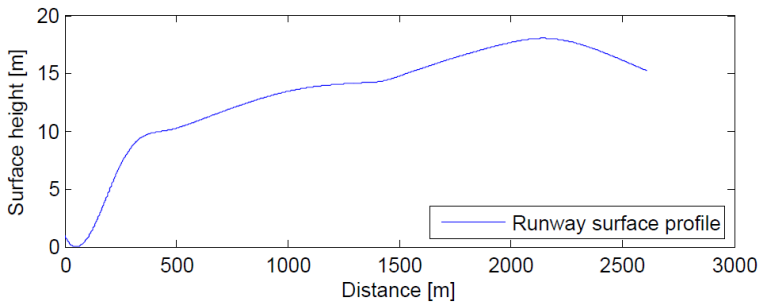


Figure 5.31: Surface profile of the runway at the Braunschweig airport. Please note the scaling difference between the x - and z -axis.

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

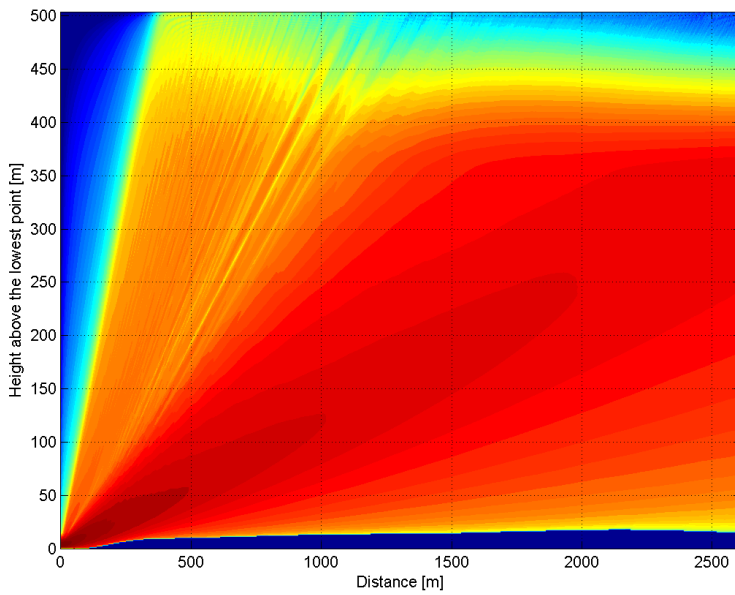


Figure 5.32: Field over the Braunschweig airport, using SSA.

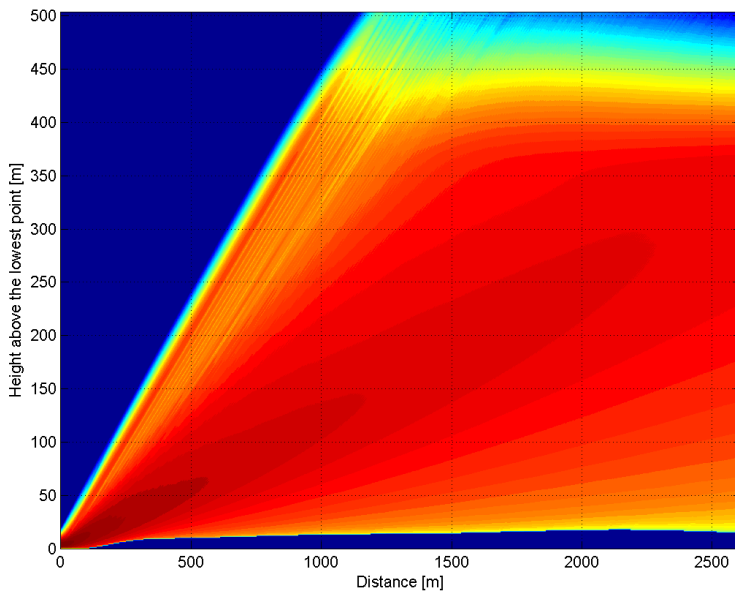


Figure 5.33: Field over the Braunschweig airport, using FDM.

value. This is because the effect of the undulating 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.

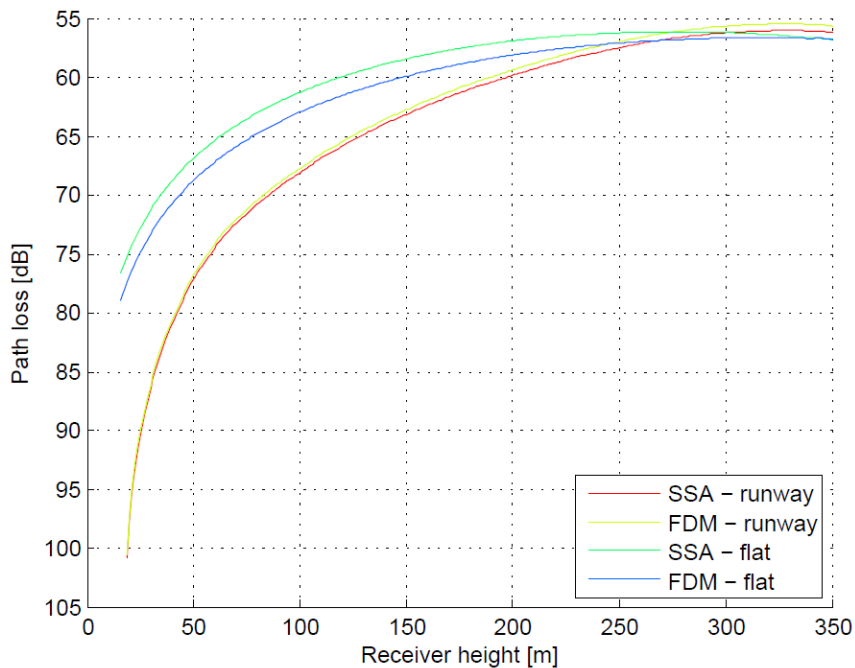
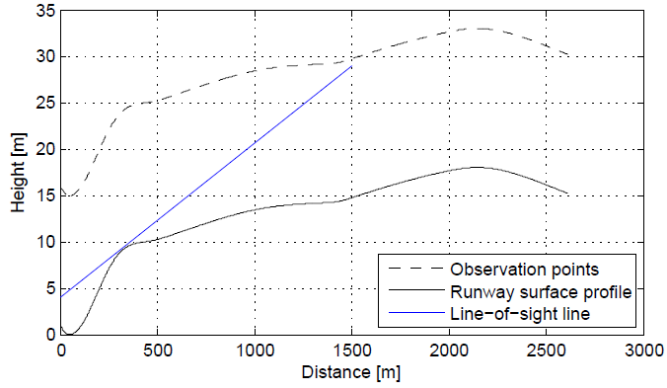


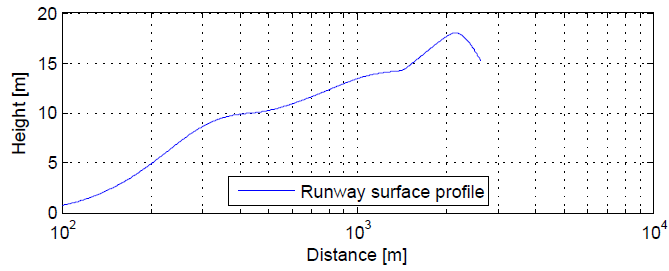
Figure 5.34: Vertical comparison of the path losses of the SSA and FDM at the end of the runway. Compared with flat surface as well.

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.

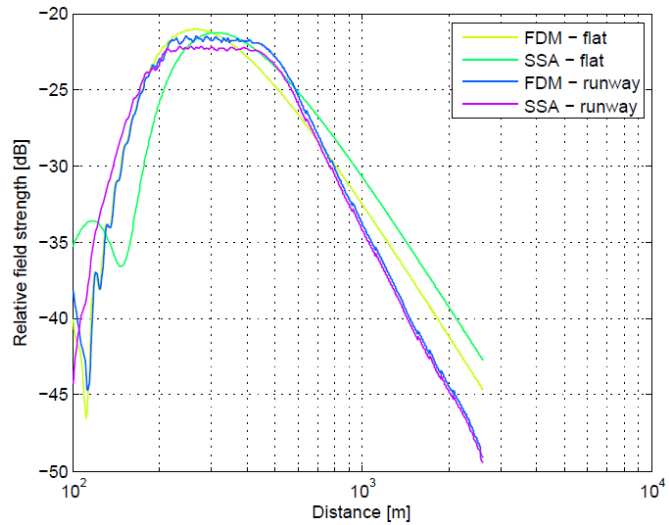
The SSA and FDM over the runway follow each other closely, especially when the field strength starts to decrease, at approximately 500 m. From 500 m, they almost overlap. The SSA and FDM over the runway follow each other closer than SSA and FDM over a flat surface.



(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.

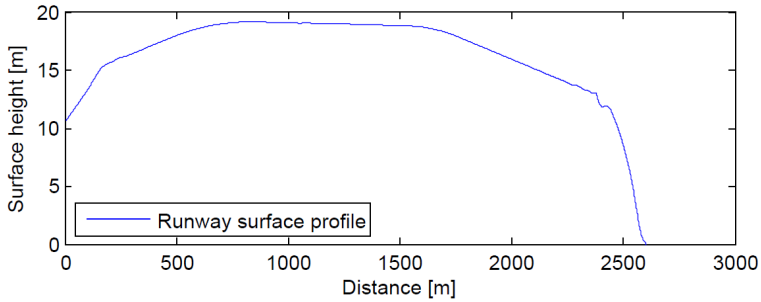


Figure 5.36: The surface profile of the runway at the Luton airport. Please note the scaling difference between the x - and z -axis.

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.

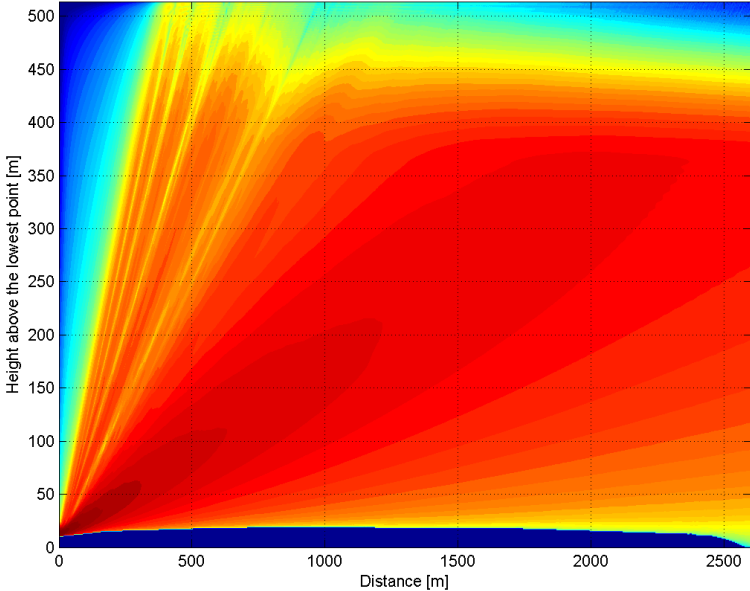


Figure 5.37: Field over the Luton airport, using SSA.

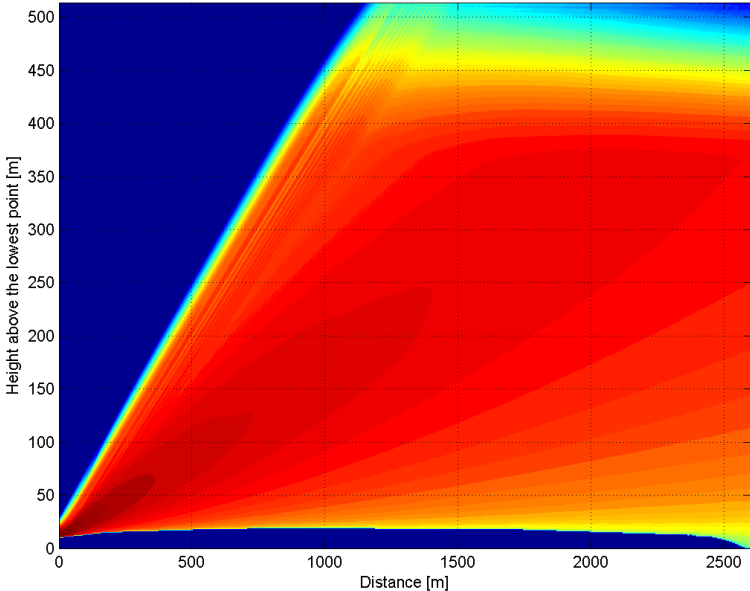


Figure 5.38: Field over the Luton airport, using FDM.

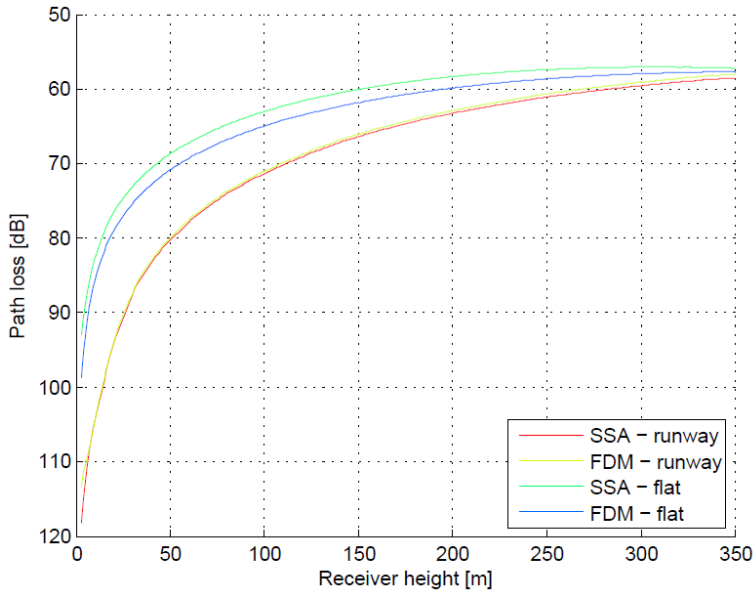
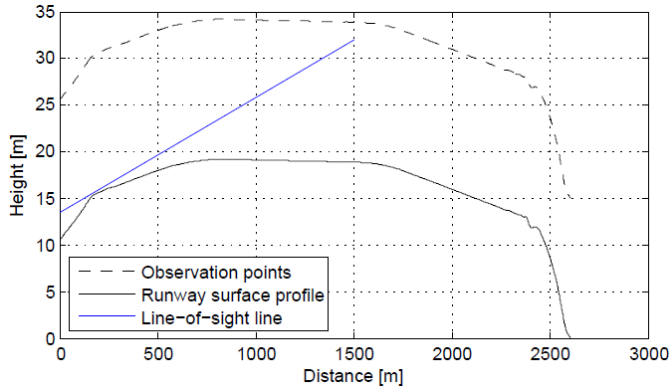


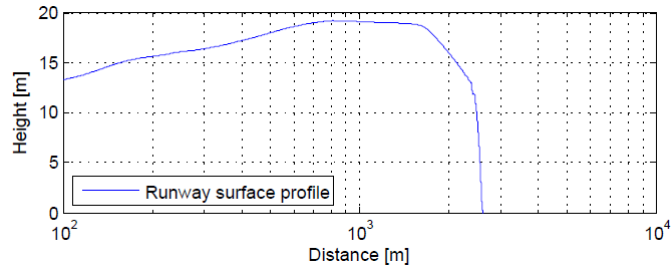
Figure 5.39: Vertical comparison of the path losses of the SSA and FDM at the end of the runway. Compared with flat surface as well.

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.

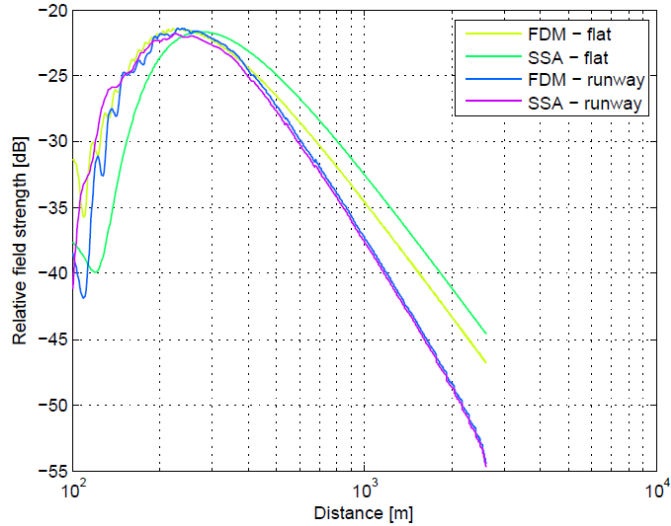
The SSA and FDM for the runway follow each other closely over the entire runway. Up to 300 m they have similar pattern. From 300 m and up, they almost overlap each other.



(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 undulating 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 continues 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° , $\pm 20^\circ$, or narrower. The implemented algorithm has been tested for signals of half-power beamwidth of 55° , $\pm 27.5^\circ$, with good results. It is therefore possible that the results can be reasonable with a beamwidth of $\pm 40^\circ$. 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.

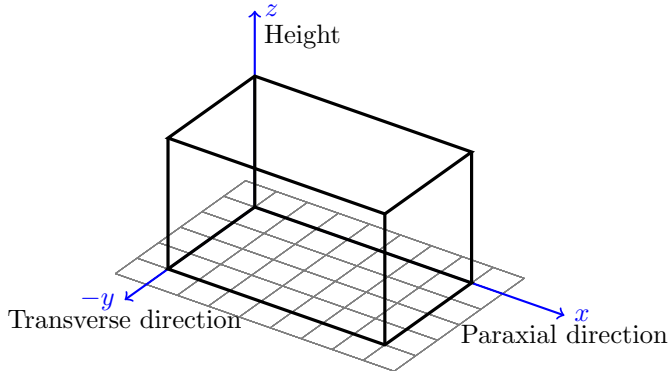


Figure 6.1: Computational domain for simulation using the Parabolic Equation method in 3D.

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

REFERENCES

- 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].

$$U(x, p) = \mathcal{F}\{u(x, z)\} = \int_{-\infty}^{\infty} u(x, z)e^{-i2\pi pz} dz \quad (\text{A.1})$$

$$u(x, z) = \mathcal{F}^{-1}\{U(x, p)\} = \int_{-\infty}^{\infty} U(x, p)e^{i2\pi pz} dp \quad (\text{A.2})$$

A.2 Fourier Sine Transform

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

$$U(x, p) = \mathcal{S}\{u(x, z)\} = \int_0^{+\infty} u(x, z) \sin(2\pi pz) dz \quad (\text{A.3})$$

A.3 Discrete Fourier Sine Transform

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

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

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

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

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].

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

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

Appendix B

Derivations

B.1 Derivation of the Standard Parabolic Equation

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial z^2} + k^2 n^2 \psi = 0$$

x : direction of propagation

z : height

(B.1)

k : wave number in vacuum, $\frac{2\pi}{\lambda}$

n : refractive index, function of x and z , slowly varying

Introducing $u(x, z) = e^{ikx} \psi(x, z)$, [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).

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

$$(E) \iff \left\{ \frac{\partial}{\partial x} + ik(1 - Q) \right\} \left\{ \frac{\partial}{\partial x} + ik(1 + Q) \right\} u = 0 \quad (B.2)$$

$$Q = \sqrt{\frac{1}{k^2} \frac{\partial^2}{\partial z^2} + n^2(x, z)}$$

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].

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

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).

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

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).

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

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).

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

$$\xi_m = \frac{x_{m-1} + x_m}{2} \quad (\text{B.7})$$

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

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

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

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

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

$$\begin{aligned} & u_{m-1}^{j+1} + u_m^{j+1} + u_{m-1}^{j-1} + u_m^{j-1} - 2u_{m-1}^j - 2u_m^j + b(u_m^j - u_{m-1}^j) + \\ & a_m^j(u_{m-1}^j + u_m^j) = 0 \end{aligned} \quad (\text{B.11})$$

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).

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

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.

$$\begin{aligned}
 & \begin{bmatrix} 1 & & & & \\ 1 & \alpha_m^1 & 1 & & \\ & & \ddots & & \\ & & & 1 & \alpha_m^{N-1} & 1 \\ & & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_m \\ \vdots \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ -1 & \beta_m^1 & -1 & & \\ & & \ddots & & \\ & & & -1 & \beta_m^{N-1} & -1 \\ & & & & & -1 \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \vdots \\ U_{m-1} \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}, \\
 & U_m = \begin{bmatrix} u_m^0 \\ u_m^1 \\ \vdots \\ u_m^{N-1} \\ u_m^N \end{bmatrix}, \quad U_{m-1} = \begin{bmatrix} u_{m-1}^0 \\ u_{m-1}^1 \\ \vdots \\ u_{m-1}^{N-1} \\ u_{m-1}^N \end{bmatrix}
 \end{aligned} \tag{B.13}$$

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.

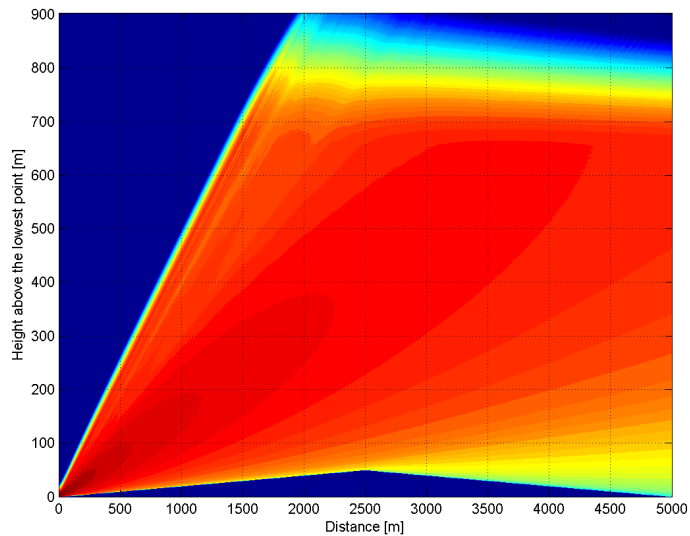


Figure C.1: Field simulation over the wedge given in figure 5.25 using the SSA. Frequency: 100 MHz.

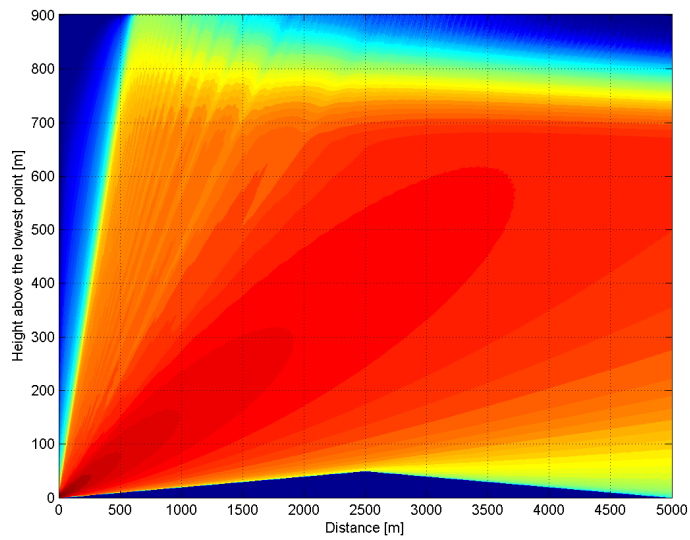


Figure C.2: Field simulation over the wedge given in figure 5.25 using the FDM. Frequency: 100 MHz.

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: *SSAirregularTerrainAbsorptionLayer*
- 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.

```
1 % ParabolicEquation_noGround.m: Script that simulate free-space only, for
2 %                               testing the absorption layer.
3 clear all
4
5 % Setting the parameters:
6 theta0 =0;
7 beta = pi/15;
8 A = 2;
9 frequency = 100*10^6;
10 deltaX =1;
11 maxX = 5000;
12 maxHinterestHeight =20;
13 numLayersArray =10;
14 numPointsPerLayerArray =20;
```

```
15 numElts = length(numPointsPerLayerArray);
16 numPointsInLayer =100;
17 deltaZarr = 1;
18
19 % Looping over the deltaZ values in question:
20 for b = 1: length(deltaZarr)
21     deltaZ = deltaZarr(b);
22
23     antHeight =10;
24
25     % Looping over the antenna heights in question:
26     for a =1:length(antHeight)
27         counter = 0;
28         simulationType = cell(numElts,1);
29         for heightIndex = 1: length(maxHinterestHeight)
30             for n = 1: length(numLayersArray)
31
32                 for m = 1: length(numPointsPerLayerArray)
33                     counter = counter +1;
34                     numLayers = numLayersArray(n);
35                     numPointsPerLayer = numPointsPerLayerArray(m);
36
37                     zs = antHeight(a);
38
39                     maxHinterest =260;
40
41                     % Creating z-vector with absorption layer:
42                     [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2( ...
43                         maxHinterest, deltaZ,numPointsInLayer);
44
45                     % Creatin initial field:
46                     initialFieldFDM =createInitialField(zs,theta0,beta,...
47                         zVectFDM, A, frequency,'gaussian1');
48                     numZpoints = length(zVectFDM);
49
50
51                     % Creating the x vector:
52                     xVect = verticalVector([0:deltaX:maxX]);
53
54
55                     numIterations = ceil(maxX/xVect(numZpoints));
56                     xVectTot = xVect;
57                     L = length(zVectFDM);
58                     sourceIndex = ceil(((L/deltaZ) +1)*(zs/L));
59
60                     % Simulating the field:
61                     tic
62                     [uValuesFDMalne,maxEigVal,antennaSourceIndex] = ...
```

```

63         FDMnoGround(initialFieldFDM, ...
64                     zVectFDM,xVect,HindexFDM,frequency,...
65                     numPointsInLayer,sourceIndex);
66         uValuesTot = uValuesFDMalme;
67
68         toc
69
70         deltaZstr = num2str(deltaZ);
71         yText = 'Height distance [m]';
72         %
73
74         eField = uValuesTot(antennaSourceIndex,:);
75         simulationType{counter,1} = ['FDM'];
76
77         if counter == 1
78             eFieldTot = eField;
79         else
80
81             eFieldTot = vertcat(eFieldTot,eField);
82         end
83
84         tx = zs;
85         rx = zs;
86
87         % Plotting the simulated field:
88         fig = figure('visible','off');
89         uValuesAux = uValuesTot;
90         uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
91         contourf(10.*log10(abs(uValuesAux.^2)),50)
92         hold on
93         contour(10.*log10(abs(uValuesAux.^2)),50)
94
95         part1Title = ['FDM - absorption layer test'];
96         part12Titile = [ ' ', '\Deltaz = ', ' ', deltaZstr, ...
97                         'm, ', '\Deltax = ', ' ', ...
98                         num2str(deltaX), 'm, '];
99         part21Title = ['Distance from center point of ', ...
100                      'source to beginning of absorption layer: ', ...
101                      num2str(maxHinterest-zs), 'm' ];
102         part2Title=['Number of points in absorption layer:', ...
103                   ' ', num2str(numPointsInLayer)];
104
105         titleVal2 = {part1Title;part12Titile;part21Title;...
106                    part2Title};
107         title(titleVal2)
108         xlabel('Distance [m]');
109         ylabel(yText);
110         grid on

```

```

111         titleFig = [...
112             'FDM_noGround/FDM.AbsorptioLayerTest_ant_h-',...
113             '-',num2str(antHeight(a)), ...
114             '-', 'Deltaz', '-', deltaZstr, '-', 'Deltax-', '-', ...
115             num2str(deltaX), '_max_h-',...
116             num2str(maxHinterestHeight(heightIndex)), ...
117             'num_points_layer', num2str(numPointsInLayer),...
118             '.png'];
119         saveas(fig,titleFig,'png');
120     end
121 end
122 end
123 part1Title = ['FDM - absorption layer test'];
124 part12Titile = [ ' ', '\Deltaz = ', ' ', deltaZstr,...
125     'm, ', '\Deltax = ', ' ', ...
126     num2str(deltaX), 'm, '];
127 part21Title = ['Distance from center point of ',...
128     'source to beginning of absorption layer: ',...
129     num2str(maxHinterest-zs), 'm'];
130 part2Title=['Number of points in absorption layer:',...
131     ' ', num2str(numPointsInLayer)];
132
133 plotTitle = {part1Title;part12Titile;part21Title;...
134     part2Title};
135
136 titleFig = ['FDM_noGround/FDM.AbsorptioLayerTest_ant_h-', '-',...
137     num2str(antHeight(a)), ...
138     '-', 'Deltaz', '-', deltaZstr, '-', 'Deltax-', '-', ...
139     num2str(deltaX), '_max_h-',...
140     num2str(maxHinterestHeight(heightIndex)), ...
141     'num_points_layer', num2str(numPointsInLayer), ...
142     'newAbsLAYER4', '.png'];
143 tx = 10000;
144 rx = 10000;
145
146 % Comparing the simulated free-space loss with analytical
147 % free-space loss:
148 freeSpaceLoss_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
149     beta,frequency,simulationType, plotTitle,titleFig);
150
151
152 end
153 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.

```

1 % parabolicEquation_SSA_FDM.m: Script running simulations with varying
2 %           delta x and delta x values over a flat
3 %           surface using the SSA and FDM.
4
5 clear all
6
7 % Setting the parameters:
8 theta0 = 0;
9 beta =pi/18;
10 A = 1;
11 frequency = 110*10^6;
12 maxX = 3000;
13 numPtsAbsorptionLayer = 150;
14
15 deltaZarr =[0.5 1 1.3];
16 antHeight = [15];
17 deltaXvect =[0.5 1 1.3];
18 for a =1:length(antHeight)
19
20
21     zs = antHeight(a);
22     maxHinterestHeight = 10;
23     numElts = 2;
24     simulationType = cell(numElts,1);
25
26     % Looping over the delta x values:
27     for n = 1:length(deltaXvect)
28         clear eFieldTot
29         counter = 0;
30         doubleCounter = 0;
31
32         maxHinterestHeight = 10;
33
34         for heightIndex = 1: length(maxHinterestHeight)
35
36             % Looping over the delta z values
37             for b = 1: length(deltaZarr)
38                 deltaZ = deltaZarr(b);
39
40                 counter = counter +1;
41                 doubleCounter = doubleCounter +1;

```

```

42         deltaX = deltaXvect(n);
43         xVect = verticalVector([0:deltaX:maxX]);
44         maxHinterest = 350 +zs;
45
46         % Creating initial field:
47         [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
48             maxHinterest,deltaZ,numPtsAbsoptionLayer);
49         initialFieldFDM = createInitialField(zs,theta0,beta,...
50             zVectFDM,A, frequency,'gaussian1');
51         numZpoints = length(zVectFDM);
52
53         % Calculating field:
54         tic
55         uValuesSplitStep =splitStepAlgorithmAbsorptionLayer(...
56             initialFieldFDM,zVectFDM,xVect, ...
57             HindexFDM,frequency,numPtsAbsoptionLayer);
58         toc
59         tic
60         [uValuesFDMalne,maxEigVal]=...
61             FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
62             zVectFDM,xVect,HindexFDM,frequency,...
63             numPtsAbsoptionLayer);
64         toc
65
66         deltaZstr = num2str(deltaZ*10);
67         yText = strcat('Height above surface [m]');
68
69         % Extract the simulated fields at the height of interest:
70         eFieldSSA=uValuesSplitStep(...
71             ceil(HindexFDM*(zs/maxHinterest)),:);
72         eFieldFDM=uValuesFDMalne(...
73             ceil(HindexFDM*(zs/maxHinterest)),:);
74
75         simulationType{doubleCounter,1} = ['SSA: \Deltax=',...
76             num2str(deltaX),'m, \Deltaz=',num2str(deltaZ),'m'];
77         doubleCounter = doubleCounter +1;
78         simulationType{doubleCounter,1} = ['FDM:\Deltax=',...
79             num2str(deltaX),'m, \Deltaz=',num2str(deltaZ),'m'];
80         tx = zs;
81         rx = zs;
82
83         if counter == 1
84             eFieldTot = eFieldSSA;
85             eFieldTot = vertcat(eFieldTot,eFieldFDM);
86         else
87             eFieldTot = vertcat(eFieldTot,eFieldSSA);
88             eFieldTot = vertcat(eFieldTot,eFieldFDM);
89         end

```

```

90
91     % Plot the simulated fields:
92     fig2 = figure('visible','off');
93     uValuesAux = uValuesSplitStep;
94     uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
95     contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
96     hold on
97     contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
98
99     xlabel('Distance [m]');
100    ylabel(yText);
101    grid on
102    titleFig = ['Results.to.thesis/SSA_', ...
103              'Deltaz','_',deltaZstr,'_', 'Deltax','_', ...
104              num2str(deltaX*10),'_distance-source-abslayer',...
105              num2str(maxHinterest-zs),'freq-',...
106              num2str(frequency/(10^6)),'_deltaDiff.png'];
107    saveas(fig2,titleFig,'png');
108
109
110    fig = figure('visible','off');
111    uValuesAux = uValuesFDMalne;
112    uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
113    contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
114    hold on
115    contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
116    xlabel('Distance [m]');
117    ylabel(yText);
118    grid on
119    titleFig = ['Results.to.thesis/FDM_', ...
120              'Deltaz','_',deltaZstr,'_', 'Deltax','_', ...
121              num2str(deltaX*10),'freq-',...
122              num2str(frequency/(10^6)),'_deltaDiff.png'];
123    saveas(fig,titleFig,'png');
124
125
126
127
128    end
129
130    tx =zs;
131    rx = zs;
132    part1Title = ['SSA and FDM - flat surface'];
133    titleFig = ['Results.to.thesis/SSA.FDM-', ...
134              'Deltaz','_',deltaZstr,'_', 'Deltax','_', ...
135              num2str(deltaX*10),'freq-',...
136              num2str(frequency/(10^6)),'_deltaDiff.png'];
137    % Comparison between simulated and analytical results:
    
```

```
138         pathLossFlat_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
139             beta,frequency,simulationType, ' ',titleFig);
140
141     end
142 end
143 end
```

E.1.3 ParabolicEquation_SSA_FDM_deltaValueTest.m

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

```
1 % parabolicEquation_SSA_FDM_deltaValueTes.m: Script simulating the 'slow'
2 %     propagation, the value of
3 %     delta x and delta z is small
4
5
6 % Setting the parameters:
7 theta0 = 0;
8 beta = pi/18;
9 A = 1;
10 frequency = 110*10^6;
11 deltaX = 1;
12 maxX = 3000;
13 numPtsAbsorptionLayer = 150;
14 deltaZarr = [0.3];
15 antHeight = [15];
16 deltaXvect = [0.3];
17
18 for a =1:length(antHeight)
19     counter = 0;
20     doubleCounter = 0;
21
22     zs = antHeight(a);
23     maxHinterestHeight = 10;
24     numElts = 2;
25     simulationType = cell(numElts,1);
26
27     % Looping over the delta x values:
28     for n = 1:length(deltaXvect)
29
30         maxHinterestHeight = 10;
31
32         for heightIndex = 1: length(maxHinterestHeight)
33
34             % Looping over the delta z values:
```

```

35     for b = 1: length(deltaZarr)
36         deltaZ = deltaZarr(b);
37
38         counter = counter +1;
39         doubleCounter = doubleCounter +1;
40         deltaX = deltaXvect(n);
41         xVect = verticalVector([0:deltaX:maxX]);
42         maxHinterest = 350 +zs;
43
44         % Creating initial field:
45         [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
46             maxHinterest,deltaZ,numPtsAbsoptioLayer);
47         initialFieldFDM = createInitialField(zs,theta0,beta,...
48             zVectFDM,A, frequency,'gaussian1');
49         numZpoints = length(zVectFDM);
50
51         % Calculating field:
52         tic
53         uValuesSplitStep =splitStepAlgorithmAbsorptionLayer(...
54             initialFieldFDM,zVectFDM,xVect, ...
55             HindexFDM,frequency,numPtsAbsoptioLayer);
56         toc
57         tic
58         [uValuesFDMalne,maxEigVal]=...
59             FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
60             zVectFDM,xVect,HindexFDM,frequency,...
61             numPtsAbsoptioLayer);
62         toc
63
64         deltaZstr = num2str(deltaZ);
65         yText = strcat('Height above surface [m]');
66
67         % Extracting field at the height of interest, the antenna
68         % height:
69         eFieldSSA=uValuesSplitStep(...
70             ceil(HindexFDM*(zs/maxHinterest)),:);
71         eFieldFDM=uValuesFDMalne(...
72             ceil(HindexFDM*(zs/maxHinterest)),:);
73
74         simulationType{doubleCounter,1} = ['SSA:'];
75         doubleCounter = doubleCounter +1;
76         simulationType{doubleCounter,1} = ['FDM:'];
77         tx = zs;
78         rx = zs;
79
80     if counter == 1
81         eFieldTot = eFieldSSA;
82         eFieldTot = vertcat(eFieldTot,eFieldFDM);

```

```

83         else
84             eFieldTot = vertcat(eFieldTot,eFieldSSA);
85             eFieldTot = vertcat(eFieldTot,eFieldFDM);
86         end
87
88         % Plotting the fields:
89         fig2 = figure('visible','off');
90         uValuesAux = uValuesSplitStep;
91         uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
92         contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
93         hold on
94         contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
95         xlabel('The surface');
96         ylabel(yText);
97         grid on
98         titleFig = ['DeltaValueTest.results/SSA', ...
99                 '-','Deltaz','- ',deltaZstr,'-','Deltax','-','- ', ...
100                num2str(deltaX),'freq-',...
101                num2str(frequency/(10^6)),'_deltaTest.png'];
102         saveas(fig2,titleFig,'png');
103
104
105         fig = figure('visible','off');
106
107         uValuesAux = uValuesFDMalne;
108         uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
109         contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
110         hold on
111         contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
112         xlabel('The surface');
113         ylabel(yText);
114         grid on
115         titleFig = ['DeltaValueTest.results/FDM', ...
116                 '-','Deltaz','- ',deltaZstr,'-','Deltax','-','- ', ...
117                num2str(deltaX),'freq-',...
118                num2str(frequency/(10^6)),'_deltaTest.deltaTest.png'];
119         saveas(fig,titleFig,'png');
120
121     end
122
123     tx =zs;
124     rx = zs;
125
126     plotTitle = ' ';
127     titleFig = ['DeltaValueTest.results/SSA.FDM', ...
128             '-','Deltaz','- ',deltaZstr,'-','Deltax','-','- ', ...
129             num2str(deltaX),'freq-',...
130             num2str(frequency/(10^6)),'_deltaTest.png'];

```

```

131
132         % Comparing path loss of simulated field with the analytical
133         % path loss
134         pathLossFlat_beamParam(A,tx,rx,xVect,eFieldTot,zs,...
135             beta,frequency,simulationType, plotTitle,titleFig);
136
137     end
138 end
139 end

```

E.1.4 SSA_FDM_indra_r_loss.m

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

```

1 % SSA.FDM.indra.r.loss.m: Script calculating the electric field along a
2 % flat surface and compare the results with analytical results from Indra.
3 % In the computations, additional loss, (1/r) is added in order to 'make'
4 % a 3D model.
5
6 clear all
7
8 % Setting the parameters:
9 theta0 = 0;
10 beta =(55/(2*360))*(2*pi);
11 A =10;
12 frequency = 110*10^6;
13
14 deltaX = 1;
15 maxX = 3000;
16 compareDistance = 1000;
17 numPtsAbsorptionLayer = 150;
18 deltaZ = 1;
19 antHeight = 3;
20 deltaXvect = [1];
21 counter = 0;
22 doubleCounter = 0;
23 maxHinterestHeight = 10;
24 numElts = 2;
25 simulationType = cell(numElts,1);
26
27 counter = counter +1;
28 doubleCounter = doubleCounter +1;
29
30 xVect = verticalVector([0:deltaX:maxX]);
31 maxHinterest = 350 +antHeight;

```

```

32
33 % Creating initial field:
34 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
35     maxHinterest,deltaZ,numPtsAbsoptioLayer);
36 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
37     zVectFDM,A, frequency,'gaussian1');
38
39 % Finding the gain of the used beam [dBi]:
40 maxValueInitField = max(initialFieldFDM);
41 sumInitField = sum(initialFieldFDM);
42 findRes = find(abs(initialFieldFDM)>0);
43 numResPts = length(findRes);
44 isotropicSource = sumInitField/numResPts;
45 dbiGain = maxValueInitField/isotropicSource
46
47 numZpoints = length(zVectFDM);
48
49 % Calculating field with 1/r-loss added to the results:
50 tic
51 uValuesSplitStep =SSA.addRloss(...
52     initialFieldFDM,zVectFDM,xVect, ...
53     HindexFDM,frequency,numPtsAbsoptioLayer,antHeight);
54 toc
55 tic
56 [uValuesFDMalne]=...
57     FDM.addRloss(initialFieldFDM, ...
58     zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptioLayer,antHeight);
59 toc
60
61 deltaZstr = num2str(deltaZ);
62 yText = strcat('Height above surface [m]');
63
64 % Extracting the simulated values at the height of the antenna:
65 eFieldSSA=uValuesSplitStep(ceil(HindexFDM*(antHeight/maxHinterest)),:);
66 eFieldFDM=uValuesFDMalne(ceil(HindexFDM*(antHeight/maxHinterest)),:);
67
68 simulationType{1,1} = ['SSA'];
69 doubleCounter = doubleCounter +1;
70 simulationType{2,1} = ['FDM'];
71 tx = antHeight;
72 rx = antHeight;
73
74 eFieldTot = eFieldSSA;
75 eFieldTot = vertcat(eFieldTot,eFieldFDM);
76
77
78 %Plot SSA figure:
79 fig2 = figure('visible','off');
```



```

80 part1Title = ['Split-Step Algorithm - flat surface'];
81 titleVal2 = {part1Title};
82
83 uValuesAux = uValuesSplitStep;
84 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
85 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
86 hold on
87 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
88 %title(titleVal2)
89 xlabel('Distance [m]');
90 ylabel(yText);
91 grid on
92 titleFig = ['Indra.r.loss.added/SSA.flat.rLoss.png'];
93 saveas(fig2, titleFig, 'png');
94
95 %Plot FDM figure:
96 fig = figure('visible', 'off');
97 part1Title = ['Finite-Difference Method - flat surface'];
98 titleVal2 = {part1Title};
99
100 uValuesAux = uValuesFDMalne;
101 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
102 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
103 hold on
104 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
105 %title(titleVal2)
106 xlabel('Distance [m]');
107 ylabel(yText);
108 grid on
109 titleFig = ['Indra.r.loss.added/FDM.flat.rLoss.png'];
110 saveas(fig, titleFig, 'png');
111 %
112 % Comparing with results from Indra (along the surface):
113 tx = antHeight;
114 rx = antHeight;
115 rx.xArr = xVect;
116 rx.zArr = ones(length(xVect), 1).*antHeight;
117 part1Title = ['SSA and FDM - flat surface - along the surface'];
118 plotTitle = ' ';
119 titleFig = ['Indra.r.loss.added/SSA.FDM.flat.along.surface.png'];
120 filename = 'IndraWedge2.results/LPDA-u-kile-2.E.xls';
121
122 pathLossIndra_alongX(rx.xArr, rx.zArr, eFieldTot, frequency, ...
123     simulationType, plotTitle, titleFig, filename);
124
125 % Comparing with results from Indra with varying height at the end of the
126 % surface:
127 xIndex = find(xVect >= 1000, 1);

```

```

128 eFieldSSA1 = zeros(1,length(zVectFDM));
129 eFieldFDM1 = zeros(1,length(zVectFDM));
130 for i = 1:length(zVectFDM)
131     eFieldSSA1(1,i) = uValuesSplitStep(i,xIndex);
132     eFieldFDM1(1,i) = uValuesFDMalne(i,xIndex);
133 end
134
135 plotTitle1 = ' ';
136
137 eFieldTot1 = eFieldFDM1;
138 eFieldTot1 = eFieldSSA1;
139 eFieldTot1 = vertcat(eFieldTot1,eFieldFDM1);
140 rx.xArr1 = ones(length(zVectFDM),1).*length(xVect);
141 rx.zArr1 = zVectFDM;
142
143 titleFig1 = ['Indra_r.loss.added/SSA.FDM.flat.height.varying_rLoss.png'];
144 filename1 = 'IndraWedge.results/LPDA-u-kile.E.xls';
145
146 compareHeight = 240;
147 pathLossFlat.Indra(rx.xArr1,rx.zArr1,eFieldTot1,...
148     frequency,simulationType, plotTitle1,titleFig1,filename1,compareHeight)
149
150 % Calculating the field strength without any additional 1/r-loss:
151 tic
152 uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
153     initialFieldFDM,zVectFDM,xVect, ...
154     HindexFDM,frequency,numPtsAbsoptionLayer);
155 toc
156
157 tic
158 [uValuesFDM2,maxEigVal]=FDMAbsorptionLayerNumEfficient2(initialFieldFDM,...
159     zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
160 toc
161
162 % Extracting the field strength at 1000 m in the vertical direction;
163 xIndex = find(xVect >= 1000,1);
164 eFieldSSA2 = zeros(1,length(zVectFDM));
165 eFieldFDM2 = zeros(1,length(zVectFDM));
166 for i = 1:length(zVectFDM)
167     eFieldSSA2(1,i) = uValuesSSA2(i,xIndex); %...
168     %length(xVect));
169     eFieldFDM2(1,i) = uValuesFDM2(i,xIndex); %...
170     %length(xVect));
171 end
172 eFieldTot2 = eFieldSSA2;
173 eFieldTot2 = vertcat(eFieldTot2,eFieldFDM2);
174
175 part1Title2 = ['SSA and FDM - flat surface - receiver height varying '...

```

```

176     '- no (1/r)-loss added'];
177 plotTitle2 = ' ';% {part1Title1};
178
179 rx_xArr2 = ones(length(zVectFDM),1).*length(xVect);
180 rx_zArr2 = zVectFDM;
181
182 titleFig2=['Indra_r_loss_added/SSA_FDM_flat_height_varying_no_rLoss.png'];
183 filename2 = 'IndraWedge.results/LPDA-u-kile.E.xls';
184 % Tests the plotting the results in the same plot:
185 compareHeight = 240;
186 eFieldTot3 = eFieldSSA1;
187 eFieldTot3 = vertcat(eFieldTot3,eFieldFDM1);
188 eFieldTot3 = vertcat(eFieldTot3,eFieldSSA2);
189 eFieldTot3 = vertcat(eFieldTot3,eFieldFDM2);
190
191 simulationType2 = cell(4,1);
192 simulationType2{1,1} = ['SSA - 1/r-loss added'];
193 simulationType2{2,1} = ['FDM - 1/r-loss added'];
194 simulationType2{3,1} = ['SSA - no 1/r-loss added'];
195 simulationType2{4,1} = ['FDM - no 1/r-loss added'];
196
197 % Vertical comparison of relative field strengths with different max
198 % heights:
199 pathLossFlat_Indra(rx_xArr2,rx_zArr2,eFieldTot3,...
200     frequency,simulationType2, plotTitle2,titleFig2,filename2,...
201     compareHeight)
202
203
204 compareHeight = 50;
205 titleFig3=['Indra_r_loss_added/',...
206     'SSA_FDM_flat_height_varying_no_rLoss_zoomed.png'];
207 pathLossFlat_Indra(rx_xArr2,rx_zArr2,eFieldTot2,...
208     frequency,simulationType, plotTitle2,titleFig3,filename2,...
209     compareHeight)
210
211
212 titleFig3=['Indra_r_loss_added/',...
213     'SSA_FDM_flat_height_varying_no_rLoss_zoomed2.png'];
214 pathLossFlat_Indra_minComp(rx_xArr2,rx_zArr2,eFieldTot2,...
215     frequency,simulationType, plotTitle2,titleFig3,filename2,...
216     compareHeight)
217
218 % Due to an error when first implementing the script, the fields without
219 % additional loss is calculated ones more:
220
221 % Setting the parameters:
222 theta0 = 0;
223 A = 10^(0.9);

```

```
224 frequency = 110*10^6;
225 deltaX = 1;
226 numPtsAbsorptionLayer = 150;
227
228 deltaZarr = [1];
229 antHeight = 3;
230 deltaXvect = [1];
231
232 Xn = [0 maxX];
233 Zn = [0 0];
234
235
236 for a = 1:length(antHeight)
237     counter = 0;
238     doubleCounter = 0;
239
240     zs = antHeight(a);
241     maxHinterestHeight = 10;
242     numElts = 2;
243     simulationType = cell(numElts,1);
244
245     % Looping over the delta x values:
246     for n = 1:length(deltaXvect)
247
248         maxHinterestHeight = 10;
249
250         for heightIndex = 1: length(maxHinterestHeight)
251
252             % Looping over the delta z values
253             for b = 1: length(deltaZarr)
254                 deltaZ = deltaZarr(b);
255
256                 counter = counter +1;
257                 doubleCounter = doubleCounter +1;
258                 deltaX = deltaXvect(n);
259
260                 maxHinterest = 350 +zs;
261
262                 % Interpolate the flat surface (algorithm for irregular
263                 % terrain used):
264                 [xVect, zSurfaceVect] = interpolate(Xn, Zn, frequency, ...
265                 'linear', deltaX);
266
267                 % Create initial field:
268                 [zVectFDM, HindexFDM] = createZvectAbsorptionLayer2(...
269                 maxHinterest, deltaZ, numPtsAbsorptionLayer);
270                 initialFieldFDM = createInitialField(zs, theta0, beta, ...
271                 zVectFDM, A, frequency, 'gaussian1');
```

```

272         numZpoints = length(zVectFDM);
273
274         % Calculating the fields:
275         tic
276         uValuesSplitStep3 = SSAirregularTerrainAbsorptionLayer(...
277             initialFieldFDM, zVectFDM, ...
278             xVect, zSurfaceVect, HindexFDM, frequency, ...
279             numPtsAbsorptionLayer, zs);
280         toc
281         tic
282         uValuesFDM3 = FDMirregularTerrainAbsorptionLayer ...
283             (initialFieldFDM, zVectFDM, xVect, zSurfaceVect, ...
284             HindexFDM, frequency, numPtsAbsorptionLayer);
285         toc
286
287         % Extracting the fields ath the given distance:
288         xIndex = find(xVect >= compareDistance, 1);
289         eFieldSSA3 = zeros(1, length(zVectFDM));
290         eFieldFDM3 = zeros(1, length(zVectFDM));
291         for i = 1:length(zVectFDM)
292             eFieldSSA3(1, i) = uValuesSplitStep3(i, ...
293                 xIndex);
294             eFieldFDM3(1, i) = uValuesFDM3(i, ...
295                 xIndex);
296         end
297
298         deltaZstr = num2str(deltaZ);
299         yText = strcat('Height above surface [m]');
300
301         simulationType{doubleCounter, 1} = ['SSA'];
302         doubleCounter = doubleCounter + 1;
303         simulationType{doubleCounter, 1} = ['FDM'];
304         tx = zs;
305         rx = zs;
306
307         if counter == 1
308             eFieldTot3 = eFieldSSA3;
309             eFieldTot3 = vertcat(eFieldTot3, eFieldFDM3);
310         else
311             eFieldTot3 = vertcat(eFieldTot3, eFieldSSA3);
312             eFieldTot3 = vertcat(eFieldTot3, eFieldFDM3);
313         end
314
315         % Plotting the fields:
316         fig2 = figure('visible', 'on');
317
318         part1Title = ['Split-Step Algorithm - wedge'];
319         part12Titile = [ ' ', '\Deltaz = ', ' ', deltaZstr, ...

```

```

320         'm, ', '\Deltax = ', ' ', ...
321         num2str(deltaX), 'm, Source height: ', ...
322         num2str(antHeight(a)), 'm'];
323 part21Title = ['Distance from center point of ',...
324             'source to beginning of absorption layer: ',...
325             num2str(maxHinterest-zs), 'm' ];
326 part2Title=['Number of points in absorption layer:',...
327             ' ', num2str(numPtsAbsoptioLayer)];
328
329 titleVal2 = {part1Title;part12Titile;part21Title;...
330             part2Title};
331
332 uValuesAux = uValuesSplitStep3;
333 uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
334 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
335 hold on
336 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
337 %title(titleVal2)
338 xlabel('Distance [m]');
339 ylabel(yText);
340 grid on
341 titleFig = ['Indra_r_loss_added/SSA_flat_noRloss', '.png'];
342 saveas(fig2, titleFig, 'png');
343
344 fig = figure('visible', 'on');
345
346 part1Title = ['Finite-Difference Method - wedge'];
347 part12Titile = [ ' ', '\Deltaz = ', ' ', deltaZstr,...
348             'm, ', '\Deltax = ', ' ', ...
349             num2str(deltaX), 'm, Source height: ', ...
350             num2str(antHeight(a)), 'm'];
351 part21Title = ['Distance from center point of ',...
352             'source to beginning of absorption layer: ',...
353             num2str(maxHinterest-zs), 'm' ];
354 part2Title=['Number of points in absorption layer:',...
355             ' ', num2str(numPtsAbsoptioLayer)];
356
357 titleVal2 = {part1Title;part12Titile;part21Title;...
358             part2Title};
359
360 uValuesAux = uValuesFDM3;
361 uValuesAux(abs(uValuesAux)<10^-4) = 10^-4;
362 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
363 hold on
364 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
365 %title(titleVal2)
366 xlabel('Distance [m]');
367 ylabel(yText);

```

```

368         grid on
369         titleFig = ['Indra_r_loss_added/FDM_flat_noRloss.png'];
370
371         saveas(fig,titleFig,'png');
372         titleFig2=['Indra_r_loss_added/FDM_ant_h-', '-', ...
373                 num2str(antHeight(a)), ...
374                 '-', 'Deltaz', '-', deltaZstr, '-', 'Deltax', '-', ...
375                 num2str(deltaX), '_distance_source_abslayer', ...
376                 num2str(maxHinterest-zs), 'wedge.pdf'];
377         %print (fig, '-dpdf', titleFig2);
378
379     end
380
381     tx =zs;
382     rx = zs;
383     part1Title = ['SSA and FDM - wedge '];
384     part12Titile = [ ' ', '\Deltaz = ', ' ', deltaZstr, ...
385                    'm, ', '\Deltax = ', ' ', ...
386                    num2str(deltaX), 'm, Source height: ', ...
387                    num2str(antHeight(a)), 'm'];
388     part21Title = ['Distance from center point of ', ...
389                  'source to beginning of absorption layer: ', ...
390                  num2str(maxHinterest-zs), 'm' ];
391     part2Title=['Number of points in absorption layer:', ...
392               ' ', num2str(numPtsAbsorptionLayer)];
393
394     plotTitle = {part1Title;part12Titile;part21Title;...
395                 part2Title};
396
397     titleFig =['Indra_r_loss_added/SSA-FDM_ant_h-', '-', ...
398              num2str(antHeight(a)), ...
399              '-', 'Deltaz', '-', deltaZstr, '-', 'Deltax', '-', ...
400              num2str(deltaX), '_distance_source_abslayer', ...
401              num2str(maxHinterest-zs), 'wedge2.png'];
402
403     filename = 'IndraWedge_results/LPDA-u-kile-E.xls';
404     rx_xArr = ones(length(zVectFDM),1).*length(xVect);
405     rx_zArr = zVectFDM;
406
407     % Vertical comparison:
408     pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,...
409                        eFieldTot3,zs,...
410                        beta,frequency,simulationType,' ',titleFig,filename);
411
412     % Comparing with results from Indra (along the surface):
413     tx =antHeight;
414     rx = antHeight;
415     rx_xArr = xVect;
    
```

```

416         rx.zArr = ones(length(xVect),1).*antHeight;
417         part1Title = ['SSA and FDM - flat surface - along the surface- no additional l
418         plotTitle = ' ';
419         titleFig = ['Indra_r.loss.added/SSA_FDM.flat.along.surface.noLossAdded.png'];
420         filename = 'IndraWedge2_results/LPDA-u-kile-2.E.xls';
421
422         eFieldSSA3=uValuesSplitStep3(ceil(HindexFDM*(antHeight/maxHinterest)),:);
423         eFieldFDM3=uValuesFDM3(ceil(HindexFDM*(antHeight/maxHinterest)),:);
424         eFieldTot3 = eFieldSSA3;
425         eFieldTot3 = vertcat(eFieldTot3,eFieldFDM3);
426
427         pathLossIndra_alongX(rx.xArr,rx.zArr,eFieldTot3,frequency,...
428             simulationType, plotTitle,titleFig,filename);
429
430
431     end
432
433 end
434 end

```

E.1.5 DownwardsInclinedPlane.m

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

```

1 % DownwardsInclinedPlane.m: Perform simulations on a downward inclined
2 %           plane, and then compares the results with a
3 %           flat plane. The beam propagating along the
4 %           inclined plane has the same relative directivity
5 %           as the flat plane. The comparison between the
6 %           fields are done in the height direction.
7
8 clear all
9 % Setting the parameters for the inclined plane:
10 beta =(55/(2*360))*(2*pi);
11 A = 10^(0.9);
12 frequency = 110*10^6;
13
14 xDiff = 1000;
15 zDiff = -20;
16
17 theta0 = abs(asin(zDiff/xDiff)); % The tilt of the beam
18
19 deltaX = 1;
20 deltaZ = 1;

```



```

21 maxX = 3100;
22 minZ = -maxX*sin(theta0);
23 numPtsAbsorptionLayer = 150;
24 xDist = 3000;
25
26 deltaZarr = [1];
27 antHeight = 3;
28 deltaXvect = [1];
29
30 % The irregular terrain:
31 Xn = [0 maxX];
32 Zn = [0 minZ];
33
34 maxZcompare = 270;
35
36 % Interpolating the surface:
37 [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
38     'linear',deltaX);
39 % Shifts the surface:
40 [zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);
41
42 % Adjusting the antenna height:
43 antHeight = antHeight + zSurfaceNorm(1);
44 maxHinterest = 330 +antHeight;
45
46 simulationType = cell(4,1);
47
48 % Plot the surface:
49 surfacePlot = figure();
50 plot(xVect,zSurfaceNorm)
51 xlabel('Distance [m]');
52 ylabel('Surface height [m]');
53 titleFig = [...
54     'DownwardsInclinedPlane.results/DownwardsInclinedPlane.surface.png'];
55 saveas(surfacePlot,titleFig,'png');
56
57 % Creating initial field:
58 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
59     maxHinterest,deltaZ,numPtsAbsorptionLayer);
60 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
61     zVectFDM,A, frequency,'gaussian1');
62 numZpoints = length(zVectFDM);
63
64 % Calculating the field over the downwards inclined plane:
65 tic
66 uValuesSplitStep = SSAirregularTerrainAbsorptionLayer(...
67     initialFieldFDM,zVectFDM,...
68     xVect,zSurfaceNorm,HindexFDM,frequency,...

```

```

69     numPtsAbsorptionLayer, antHeight);
70 toc
71 tic
72 uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
73     (initialFieldFDM, zVectFDM, xVect, zSurfaceNorm, ...
74     HindexFDM, frequency, numPtsAbsorptionLayer);
75 toc
76 deltaZstr = num2str(deltaZ);
77 yText = strcat('Height above the lowest point [m]');
78
79 %maxZ = 250;
80
81 % Extracting the vertical values:
82 eFieldSSA = verticalVector(getVerticalValues(...
83     xVect, zSurfaceNorm, xDiff, zDiff, deltaX, deltaZ, ...
84     xVect, xDist, maxZcompare, uValuesSplitStep, 'down'));
85
86
87 eFieldFDM = verticalVector(getVerticalValues(...
88     xVect, zSurfaceNorm, xDiff, zDiff, deltaX, deltaZ, ...
89     xVect, xDist, maxZcompare, uValuesFDM, 'down'));
90
91 simulationType{1,1} = ['SSA inclined'];
92
93 simulationType{2,1} = ['FDM inclined'];
94 tx = antHeight;
95 rx = antHeight;
96
97 eFieldTot = eFieldSSA;
98 eFieldTot = vertcat(eFieldTot, eFieldFDM);
99
100
101
102 minPlotLevel = 10^(-4);
103 for i = 1:1
104 % Plot SSA figure:
105 plotScale = length(zVectFDM);
106 fig2 = figure('visible', 'off');
107 part1Title = ['Split-Step Algorithm - flat surface'];
108 titleVal2 = {part1Title};
109 tic
110 uValuesAux = uValuesSplitStep(1:plotScale, :);
111 uValuesAux(abs(uValuesAux) < minPlotLevel) = minPlotLevel;
112 minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
113 maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
114 disp('contourf is on')
115 contourf(xVect, zVectFDM(1:plotScale, 1), ...
116     10.*log10(abs(uValuesAux.^2)), 50)

```

```

117 hold on
118 disp('contour is on')
119 contour(xVect,zVectFDM(1:plotScale,1),...
120         10.*log10(abs(uValuesAux.^2)),50)
121 hold on
122 eFieldSSA = verticalVector(getVerticalValues(xVect,...
123         zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
124         xVect,xDist,maxZcompare,uValuesSplitStep,'down'));
125
126 %title(titleVal2)
127 xlabel('Distance [m]');
128 ylabel(yText);
129 grid on
130
131 titleFig = ['DownwardsInclinedPlane-results/SSA.InclDown.png'];
132 disp('saving ...')
133 saveas(fig2,titleFig,'png');
134 toc
135
136 tic
137 % Plot FDM figure:
138 fig = figure('visible','off');
139 part1Title = ['Finite-Difference Method - flat surface'];
140 titleVal2 = {part1Title};
141
142 uValuesAux = uValuesFDM(1:plotScale,:);
143 uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
144 minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
145 maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
146 contourf(xVect,zVectFDM(1:plotScale,1),...
147          10.*log10(abs(uValuesAux.^2)),50)
148 hold on
149 contour(xVect,zVectFDM(1:plotScale,1),...
150         10.*log10(abs(uValuesAux.^2)),50)
151 hold on
152 eFieldFDM = verticalVector(getVerticalValues(xVect,...
153         zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
154         xVect,xDist,maxZcompare,uValuesFDM,'down'));
155 %title(titleVal2)
156 xlabel('Distance [m]');
157 ylabel(yText);
158 grid on
159 titleFig = ['DownwardsInclinedPlane-results/FDM.InclDown.png'];
160 saveas(fig,titleFig,'png');
161 toc
162
163 end % Plot figure, not in use
164

```

```

165 % Comparing with results from Indra (along the surface),
166 % horizontal comparison:
167
168 tx =antHeight;
169 rx = antHeight;
170 rx.xArr = xVect;
171 rx.zArr = ones(length(xVect),1).*antHeight;
172
173 plotTitle = ' ';
174 titleFig = ['DownwardsInclinedPlane_results/SSA_FDM.InclDown.png'];
175 filename = 'IndraSource/wedge1/LPDA-u-kile.E.xls';
176
177
178 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
179 rx.zArr = zVectFDM;
180
181 simulationType1 = cell(2,1);
182 simulationType1{1,1} = 'SSA inclined';
183 simulationType1{2,1} = 'FDM inclined';
184
185 pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx.xArr,rx.zArr,eFieldTot,...
186     antHeight,beta,frequency,simulationType1,plotTitle,titleFig,filename)
187 %zDist = 250;
188
189
190 % Calculating the field over a flat surface
191 antHeight = antHeight - zSurfaceNorm(1);
192 theta0 = 0;
193 %maxHeight = 250;
194 xVect = verticalVector([0:deltaX:xDist]);
195 % Creating initial field:
196 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
197     maxHinterest,deltaZ,numPtsAbsoptionLayer);
198 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
199     zVectFDM,A, frequency,'gaussian1');
200 numZpoints = length(zVectFDM);
201
202 % Calculating field:
203 tic
204 uValuesSplitStep_flat =splitStepAlgorithmAbsorptionLayer(...
205     initialFieldFDM,zVectFDM,xVect, ...
206     HindexFDM,frequency,numPtsAbsoptionLayer);
207 toc
208 tic
209 [uValuesFDMalne_flat,maxEigVal]=...
210     FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
211     zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptionLayer);
212 toc

```

```

213
214 % Extracting the results for vertical comparison:
215 eFieldSSA = zeros(1,maxZcompare+1); %maxHeight+1;
216 eFieldFDM = zeros(1,maxZcompare+1); %maxHeight+1;
217 for i = 1:maxZcompare+1 %maxHeight+1
218     eFieldSSA(1,i) = uValuesSplitStep_flat(i,...
219         length(xVect));
220     eFieldFDM(1,i) = uValuesFDMalnet_flat(i,...
221         length(xVect));
222 end
223
224 simulationType{3,1} = ['SSA flat'];
225 simulationType{4,1} = ['FDM flat'];
226 eFieldTot = vertcat(eFieldTot,eFieldSSA);
227 eFieldTot = vertcat(eFieldTot,eFieldFDM);
228
229 % Comparing with results from Indra (along the surface),
230 % horizontal comparison:
231 tx =antHeight;
232 rx = antHeight;
233 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
234 rx.zArr = zVectFDM;
235 part1Title = ' ';%['SSA and FDM – flat surface – along the surface'];
236 plotTitle = ' '; % {part1Title};
237
238 titleFig_comp = [...
239     'DownwardsInclinedPlane.results/SSA_FDM.compare.DownIncl.png'];
240 filename = 'IndraWedge.results/LPDA-u-kile.E.xls';
241 numCases = 4;
242 startIndex = 4;
243 numElts = length(zVectFDM);
244
245 eFieldTot1 = zeros(numCases,length(eFieldTot(1,:))); %startIndex:numElts));
246 for i = 1:numCases
247     eFieldTot1(i,:) = eFieldTot(i,:); %startIndex:numElts);
248 end
249
250
251 compareHeight = 50;
252 titleFig3 =['DownwardsInclinedPlane.results/',...
253     'SSA_FDM.flat.height.varying.DownIncl.zoomed1.png'];
254 pathLossFlat_Indra_minComp(rx.xArr,rx.zArr,eFieldTot1,...
255     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)
256
257 titleFig4 =['DownwardsInclinedPlane.results/',...
258     'SSA_FDM.flat.height.varying.DownIncl.zoomed2.png'];
259 pathLossFlat_Indra(rx.xArr,rx.zArr,eFieldTot1,...
260     frequency,simulationType, plotTitle,titleFig4,filename,compareHeight)

```

```

261
262 compareHeight = 240;
263
264 pathLossFlat_Indra(rx_xArr,rx_zArr,eFieldTot1,...
265     frequency,simulationType, plotTitle,titleFig_comp,filename,compareHeight)
266
267 % Plot SSA figure:
268 yText = strcat('Height above surface [m]');
269 fig2 = figure('visible','off');
270 part1Title = ['Split-Step Algorithm - flat surface'];
271 titleVal2 = {part1Title};
272
273 uValuesAux = uValuesSplitStep_flat;
274 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
275 contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
276 hold on
277 contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
278 %title(titleVal2)
279 xlabel('Distance [m]');
280 ylabel(yText);
281 grid on
282 titleFig = ['DownwardsInclinedPlane_results/SSA.flat.InclDown.png'];
283 saveas(fig2,titleFig,'png');
284
285 % Plot FDM figure:
286 fig = figure('visible','off');
287 part1Title = ['Finite-Difference Method - flat surface'];
288 titleVal2 = {part1Title};
289
290 uValuesAux = uValuesFDMalne_flat;
291 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
292 contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
293 hold on
294 contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
295 %title(titleVal2)
296 xlabel('Distance [m]');
297 ylabel(yText);
298 grid on
299 titleFig = ['DownwardsInclinedPlane_results/FDM.flat.InclDown.png'];
300 saveas(fig,titleFig,'png');
301
302
303 % Compare along the surface at constant height of 40 m above the lowest
304 % point:
305
306 %eFieldAlongTot
307 height = 40;
308 startIndex = 101; %100;

```

```

309 eSSArunway = (uValuesSplitStep(height, startIndex:length(xVect)));
310 eFieldAlongTot = eSSArunway;
311 eFDMrunway = (uValuesFDM(height, startIndex:length(xVect)));
312 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMrunway);
313
314 eSSAflat = (uValuesSplitStep_flat(height, startIndex:length(xVect)));
315 eFieldAlongTot = vertcat(eFieldAlongTot, eSSAflat);
316 eFDMflat = (uValuesFDMalne_flat(height, startIndex:length(xVect)));
317 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMflat);
318
319 imulationType2 = cell(4,1);
320 simulationType2{1,1} = ['SSA - downwards'];
321 simulationType2{2,1} = ['FDM - downwards'];
322 simulationType2{3,1} = ['SSA - flat'];
323 simulationType2{4,1} = ['FDM - flat'];
324
325 rx.xArr3 = xVect(startIndex:length(xVect));
326 rx.zArr3 = ones(length(rx.xArr3),1).*height;
327 titleFig3 = ['DownwardsInclinedPlane.results/', ...
328             'SSA_FDM_pathloss_horizontal_InclDown.png'];
329 antHeight2 = antHeight - zSurfaceNorm(1);
330
331 titleFig4 = ['DownwardsInclinedPlane.results/', ...
332             'SSA_FDM_pathloss_horizontal2_InclDown.png'];
333
334 pathLossIndra_alongX(rx.xArr3, rx.zArr3, eFieldAlongTot, ...
335                     frequency, simulationType2, ' ', titleFig4, ' ')
336
337
338 % Comparing fields along the surface at constant height above the surface
339 height2 = 15;
340 startIndex2 = 101;
341
342 fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
343 eSSArunway2 = zeros(length(fieldVect2),1);
344 eFDMrunway2 = zeros(length(fieldVect2),1);
345 for i = 1:length(fieldVect2)
346     eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
347         fieldVect2(i));
348     eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
349         fieldVect2(i));
350 end
351
352 eFieldAlongTot2 = eSSArunway2';
353 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMrunway2');
354 eSSAflat2 = (uValuesSplitStep_flat(height, startIndex2:length(xVect)));
355 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eSSAflat2);
356 eFDMflat2 = (uValuesFDMalne_flat(height, startIndex2:length(xVect)));

```

```

357 eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);
358
359 titleFig5 = ['DownwardsInclinedPlane_results/',...
360             'SSA_FDM_pathloss_horizontal.cst_diff_surface_InclDown.png'];
361 rx.xArr4 = xVect(startIndex2:length(xVect));
362 rx.zArr4 = ones(length(rx.xArr4),1).*height;
363
364
365 pathLossIndra_alongX(rx.xArr4,rx.zArr4,eFieldAlongTot2,...
366                     frequency,simulationType2,' ',titleFig5,' ')
367
368 % Plotting the surface profile with logarithmic axes:
369 fig = figure();
370 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
371          (startIndex2:length(xVect))); %,'Parent',ax2);
372
373 legend('Surface profile','Location','SouthWest');
374 xlabel('Distance [m]');
375 ylabel('Height [m]');
376 plotwidth = 560;
377 plotheight = 200;
378 set(fig, 'Position', [500 100 plotwidth plotheight]);
379 grid on
380 titleFig6 = ['DownwardsInclinedPlane_results/',...
381             'Runway_profile_small_InclDown.png'];
382 saveas(fig,titleFig,'png');
383 titleFig6 = titleFig6(1:(length(titleFig6)-4));
384 titleFig6 = horzcat(titleFig6,'.pdf');
385 %print (fig, '-dpdf', titleFig);
386 save2pdf(titleFig6);

```

E.1.6 UpwardsInclinedPlane2

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

```

1 % UpwardsInclinedPlan2e.m: Perform simulations on an upwards inclined
2 %                               plane, and then compares the results with a
3 %                               flat plane. The beam propagating along the
4 %                               inclined plane has the same relative directivity
5 %                               as the flat plane. The comparison between the
6 %                               fields are done in the height direction.
7
8 clear all
9 % The inclined plane:

```



```

10 beta =(55/(2*360))*(2*pi);
11 A = 10^(0.9);
12 frequency = 110*10^6;
13
14 xDiff = 1000;
15 zDiff = 20;
16
17 theta0 = abs(asin(zDiff/xDiff));
18
19 % The delta x and delta z value:
20 deltaX = 1;
21 deltaZ = 1;
22 maxX = 3000;
23 minZ = maxX*sin(theta0);
24 numPtsAbsorptionLayer = 150;
25 xDist = 3000;
26 deltaZarr =[1];
27 antHeight = 3;
28 deltaXvect = [1];
29
30 % The irregular terrain:
31 Xn = [0 maxX];
32 Zn = [0 minZ];
33 maxZcompare = 270;
34
35 % Interpolate the surface profile:
36 [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
37     'linear',deltaX);
38
39 zSurfaceNorm = zSurfaceVect;
40 antHeight = antHeight + zSurfaceNorm(1);
41
42 maxHintertest = 330 +antHeight;
43
44 simulationType = cell(4,1);
45
46 % Plot the surface:
47 surfacePlot = figure();
48 plot(xVect,zSurfaceNorm)
49 xlabel('Distance [m]');
50 ylabel('Surface height [m]');
51 titleFig = ['UpwardsInclinedPlane.results/UpwardsInclinedPlane.surface.png'];
52 saveas(surfacePlot,titleFig,'png');
53
54
55 % Creating initial field:
56 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
57     maxHintertest,deltaZ,numPtsAbsorptionLayer);

```

```

58 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
59     zVectFDM,A, frequency, 'gaussian1');
60 numZpoints = length(zVectFDM);
61
62 % Calculating field:
63 tic
64 uValuesSplitStep = SSAirregularTerrainAbsorptionLayer(...
65     initialFieldFDM,zVectFDM,...
66     xVect,zSurfaceNorm,HindexFDM,frequency,...
67     numPtsAbsorptionLayer,antHeight);
68 toc
69 tic
70 uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
71     (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
72     HindexFDM,frequency,numPtsAbsorptionLayer);
73 toc
74 deltaZstr = num2str(deltaZ);
75 yText = strcat('Height above the lowest point [m]');
76
77 %maxZ = 250;
78
79 % Extracting the vertical values:
80 eFieldSSA = verticalVector(getVerticalValues(xVect,...
81     zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
82     xVect,xDist,maxZcompare,uValuesSplitStep,'up'))';
83
84 eFieldFDM = verticalVector(getVerticalValues(xVect,...
85     zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,...
86     xVect,xDist,maxZcompare,uValuesFDM,'up'))';
87
88 simulationType{1,1} = ['SSA inclined'];
89
90 simulationType{2,1} = ['FDM inclined'];
91 tx = antHeight;
92 rx = antHeight;
93
94 eFieldTot = eFieldSSA;
95 eFieldTot = vertcat(eFieldTot,eFieldFDM);
96
97 minPlotLevel = 10^(-4);
98 for i = 1:1
99 % Plot SSA figure:
100 plotScale =length(zVectFDM); % ceil(2*length(zVectFDM)/3)
101 fig2 = figure('visible','off');
102 part1Title = ['Split-Step Algorithm - flat surface'];
103 titleVal2 = {part1Title};
104 tic
105 uValuesAux = uValuesSplitStep(1:plotScale,:);

```

```

106 uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
107 minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
108 maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
109 disp('contourf is on')
110 contourf(xVect,zVectFDM(1:plotScale,1),...
111     10.*log10(abs(uValuesAux.^2)),50)
112 hold on
113 disp('contour is on')
114 contour(xVect,zVectFDM(1:plotScale,1),...
115     10.*log10(abs(uValuesAux.^2)),50)
116 hold on
117 eFieldSSA = verticalVector(getVerticalValues(xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,
118     xVect,xDist,maxZcompare,uValuesSplitStep,'up'))';
119
120 %title(titleVal2)
121 xlabel('Distance [m]');
122 ylabel(yText);
123 grid on
124
125 titleFig = ['UpwardsInclinedPlane.results/SSA.field.InclUp.png'];
126 disp('saving ...')
127 saveas(fig2,titleFig,'png');
128 toc
129
130 tic
131 % Plot FDM figure:
132 fig = figure('visible','off');
133 part1Title = ['Finite-Difference Method - flat surface'];
134 titleVal2 = {part1Title};
135
136 uValuesAux = uValuesFDM(1:plotScale,:);
137 uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
138 minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
139 maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
140 contourf(xVect,zVectFDM(1:plotScale,1),...
141     10.*log10(abs(uValuesAux.^2)),50)
142 hold on
143 contour(xVect,zVectFDM(1:plotScale,1),...
144     10.*log10(abs(uValuesAux.^2)),50)
145 hold on
146 eFieldFDM = verticalVector(getVerticalValues(xVect,zSurfaceNorm,xDiff,zDiff,deltaX,deltaZ,
147     xVect,xDist,maxZcompare,uValuesFDM,'up'))';
148 %title(titleVal2)
149 xlabel('Distance [m]');
150 ylabel(yText);
151 grid on
152 titleFig = ['UpwardsInclinedPlane.results/FDM.field.InclUp.png'];
153 saveas(fig,titleFig,'png');

```

```

154 toc
155
156 end
157
158 % Comparing with results from Indra (along the surface),
159 % horizontal comparison:
160 tx =antHeight;
161 rx = antHeight;
162 rx.xArr = xVect;
163 rx.zArr = ones (length(xVect),1).*antHeight;
164
165 plotTitle = ' ';
166
167 titleFig = ['UpwardsInclinedPlane.results/SSA_FDM.InclUp.png'];
168 filename = 'IndraSource/wedge1/LPDA-u-kile.E.xls';
169
170 rx.xArr = ones (length(zVectFDM),1).*length(xVect);
171 rx.zArr = zVectFDM;
172
173 simulationType1 = cell(2,1);
174 simulationType1{1,1} = 'SSA inclined';
175 simulationType1{2,1} = 'FDM inclined';
176
177 pathLossWedge_Indra (Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eFieldTot,...
178     antHeight,beta,frequency,simulationType1, plotTitle,titleFig,filename)
179 %zDist = 250;
180
181 % Calculating the field along the flat surface
182 antHeight = antHeight - zSurfaceNorm(1);
183 theta0 = 0;
184 %maxHeight = 250;
185 xVect = verticalVector([0:deltaX:xDist]);
186 % Creating initial field:
187 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
188     maxHinterest,deltaZ,numPtsAbsoptionLayer);
189 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
190     zVectFDM,A, frequency,'gaussian1');
191 numZpoints = length(zVectFDM);
192
193 % Calculating field:
194 tic
195 uValuesSplitStep_flat =splitStepAlgorithmAbsorptionLayer(...
196     initialFieldFDM,zVectFDM,xVect, ...
197     HindexFDM,frequency,numPtsAbsoptionLayer);
198 toc
199 tic
200 [uValuesFDMalnet_flat,maxEigVal]=...
201     FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...

```

```

202     zVectFDM,xVect,HindexFDM,frequency,numPtsAbsorptionLayer);
203 toc
204
205 % Extracting field values for vertical comparison:
206 eFieldSSA = zeros(1,maxZcompare+1);
207 eFieldFDM = zeros(1,maxZcompare+1);
208 for i = 1:maxZcompare+1
209     eFieldSSA(1,i) = uValuesSplitStep_flat(i,...
210         length(xVect));
211     eFieldFDM(1,i) = uValuesFDMalnet_flat(i,...
212         length(xVect));
213 end
214
215
216 simulationType{3,1} = ['SSA flat'];
217 simulationType{4,1} = ['FDM flat'];
218 eFieldTot = vertcat(eFieldTot,eFieldSSA);
219 eFieldTot = vertcat(eFieldTot,eFieldFDM);
220
221 % Comparing with results from Indra (along the surface),
222 % horizontal comparison:
223 tx =antHeight;
224 rx = antHeight;
225 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
226 rx.zArr = zVectFDM;
227 part1Title = ' ';
228 plotTitle = ' ';
229
230 titleFig_comp =['UpwardsInclinedPlane.results/SSA-FDM.compare.InclUp.png'];
231 filename = 'IndraWedge.results/LPDA-u-kile.E.xls';
232 numCases = 4;
233 startIndex = 4;
234 numElts = length(zVectFDM);
235
236 eFieldTot1 = zeros(numCases,length(eFieldTot(1,:)));
237 for i = 1:numCases
238     eFieldTot1(i,:) = eFieldTot(i,:);
239 end
240
241 compareHeight = 50;
242 titleFig3 =['UpwardsInclinedPlane.results/',...
243     'SSA-FDM.flat.height.varying.InclUp.zoomed1.png'];
244 pathLossFlat_Indra_minComp(rx.xArr,rx.zArr,eFieldTot1,...
245     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)
246
247 titleFig4 =['UpwardsInclinedPlane.results/',...
248     'SSA-FDM.flat.height.varying.InclUp.zoomed2.png'];
249 pathLossFlat_Indra(rx.xArr,rx.zArr,eFieldTot1,...

```

```

250     frequency, simulationType, plotTitle, titleFig4, filename, compareHeight)
251
252 compareHeight = 240;
253
254 pathLossFlat_Indra(rx_xArr, rx_zArr, eFieldTot1, ...
255     frequency, simulationType, plotTitle, titleFig_comp, filename, compareHeight)
256
257 % Plot SSA figure:
258 yText = strcat('Height above surface [m]');
259 fig2 = figure('visible','off');
260 part1Title = ['Split-Step Algorithm - flat surface'];
261 titleVal2 = {part1Title};
262
263 uValuesAux = uValuesSplitStep_flat;
264 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
265 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
266 hold on
267 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
268 %title(titleVal2)
269 xlabel('Distance [m]');
270 ylabel(yText);
271 grid on
272 titleFig = ['UpwardsInclinedPlane.results/SSA_flat_InclUp.png'];
273 saveas(fig2, titleFig, 'png');
274
275 % Plot FDM figure:
276 fig = figure('visible','off');
277 part1Title = ['Finite-Difference Method - flat surface'];
278 titleVal2 = {part1Title};
279
280 uValuesAux = uValuesFDMalne_flat;
281 uValuesAux(abs(uValuesAux)<10^-11) = 10^-11;
282 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
283 hold on
284 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
285 %title(titleVal2)
286 xlabel('Distance [m]');
287 ylabel(yText);
288 grid on
289 titleFig = ['UpwardsInclinedPlane.results/FDM_flat_InclUp.png'];
290 saveas(fig, titleFig, 'png');
291
292
293 % Compare along the surface at constant height of 40 m above the lowest
294 % point:
295
296 %eFieldAlongTot
297 height = 40;

```

```

298 startIndex = 101; %100;
299 eSSArunway = (uValuesSplitStep(height, startIndex:length(xVect)));
300 eFieldAlongTot = eSSArunway;
301 eFDMrunway = (uValuesFDM(height, startIndex:length(xVect)));
302 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMrunway);
303
304 eSSAflat = (uValuesSplitStep_flat(height, startIndex:length(xVect)));
305 eFieldAlongTot = vertcat(eFieldAlongTot, eSSAflat);
306 eFDMflat = (uValuesFDM_flat(height, startIndex:length(xVect)));
307 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMflat);
308
309 imulationType2 = cell(4,1);
310 simulationType2{1,1} = ['SSA - upwards'];
311 simulationType2{2,1} = ['FDM - upwards'];
312 simulationType2{3,1} = ['SSA - flat'];
313 simulationType2{4,1} = ['FDM - flat'];
314
315 rx.xArr3 = xVect(startIndex:length(xVect));
316 rx.zArr3 = ones(length(rx.xArr3),1).*height;
317 titleFig3 = ['UpwardsInclinedPlane_results/',...
318             'SSA-FDM_pathloss.horizontal.InclUp.png'];
319 antHeight2 = antHeight - zSurfaceNorm(1);
320
321 titleFig4 = ['UpwardsInclinedPlane_results/',...
322             'SSA-FDM_pathloss.horizontal2.InclUp.png'];
323
324 pathLossIndra_alongX(rx.xArr3, rx.zArr3, eFieldAlongTot, ...
325                     frequency, simulationType2, ' ', titleFig4, ' ');
326
327
328 % Comparing fields along the surface at constant height above the surface
329 height2 = 15;
330 startIndex2 = 101;
331
332 fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
333 eSSArunway2 = zeros(length(fieldVect2),1);
334 eFDMrunway2 = zeros(length(fieldVect2),1);
335 for i = 1:length(fieldVect2)
336     %a = zSurfaceNorm(i);
337     eSSArunway2(i,1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
338         fieldVect2(i));
339     eFDMrunway2(i,1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
340         fieldVect2(i));
341 end
342
343 eFieldAlongTot2 = eSSArunway2';
344 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMrunway2');
345 eSSAflat2 = (uValuesSplitStep_flat(height, startIndex2:length(xVect)));

```

```

346 eFieldAlongTot2 = vertcat(eFieldAlongTot2,eSSAflat2);
347 eFDMflat2 = (uValuesFDMalne_flat(height,startIndex2:length(xVect)));
348 eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);
349
350 titleFig5 = ['UpwardsInclinedPlane_results/',...
351             'SSA-FDM.pathloss.horizontal.cst.diff.surface.InclUp.png'];
352 rx_xArr4 = xVect(startIndex2:length(xVect));
353 rx_zArr4 = ones(length(rx_xArr4),1).*height;
354
355
356 pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2,...
357                     frequency,simulationType2,' ',titleFig5,' ')
358 %get(gcf,'CurrentAxes')
359 ax1 = gca;
360
361 % Plotting the surface with logarithmic axes:
362 fig = figure();
363
364 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
365           (startIndex2:length(xVect))); %,'Parent',ax2);
366 % linkaxes([ax1 ax2],'y');
367 % linkaxes([ax1 ax2],'x');
368 legend('Surface profile','Location','SouthEast');
369 xlabel('Distance [m]');
370 ylabel('Height [m]');
371 plotwidth = 560;
372 plotheight = 200;
373 set(fig, 'Position', [500 100 plotwidth plotheight]);
374 grid on
375 titleFig6 = ['UpwardsInclinedPlane_results/',...
376             'Runway_profile_small.InclUp.png'];
377 saveas(fig,titleFig,'png');
378 titleFig6 = titleFig6(1:(length(titleFig6)-4));
379 titleFig6 = horzcat(titleFig6,'.pdf');
380 %print (fig, '-dpdf', titleFig);
381 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.

```

1 % WedgeComparison.Hviid.m: Simulates the field over the wedge given in the
2 %                               Hviid article using the SSA and FDM, and
3 %                               compares with the results over a flat surface.

```



```

4
5
6 clear all
7 % The inclined plane:
8 beta = pi/18;
9 A = 1;
10 frequency = 100*10^6;
11 theta0 = 0;
12
13 % The delta x and delta z value:
14 deltaX = 1;
15 deltaZ = 1;
16
17 antHeight = 3;
18 maxX = 5000;
19 halfWay = maxX/2;
20 maxZ = 50;
21 numPtsAbsorptionLayer = 250;
22
23 % The irregular terrain:
24 Xn = [0 halfWay maxX];
25 Zn = [0 maxZ 0];
26 maxZcompare = 500;
27
28 % Interpolating the surface:
29 [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
30     'linear',deltaX);
31
32 maxHintInterest = 650+antHeight;
33
34 simulationType = cell(4,1);
35
36 %Plot the surface:
37 surfacePlot = figure();
38 plot(xVect,zSurfaceVect,'k')
39 xlabel('Distance [m]');
40 ylabel('Surface height [m]');
41 plotwidth = 560;
42 plotheight = 200;
43 set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
44 titleFig = ['WedgeComparison.Hviid.Results/WedgeHviid.surface.png'];
45 saveas(surfacePlot,titleFig,'png');
46 titleFig = titleFig(1:(length(titleFig)-4));
47 titleFig = horzcat(titleFig, '.pdf');
48 save2pdf(titleFig);
49
50
51

```

```

52 % Creating initial field:
53 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
54     maxHinterest,deltaZ,numPtsAbsorptionLayer);
55 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
56     zVectFDM,A, frequency,'gaussian1');
57 numZpoints = length(zVectFDM);
58
59 % Calculating field:
60 tic
61 uValuesSplitStep = SSAirregularTerrainAbsorptionLayer(...
62     initialFieldFDM,zVectFDM,...
63     xVect,zSurfaceVect,HindexFDM,frequency,...
64     numPtsAbsorptionLayer,antHeight);
65 toc
66 tic
67 uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
68     (initialFieldFDM,zVectFDM,xVect,zSurfaceVect,...
69     HindexFDM,frequency,numPtsAbsorptionLayer);
70 toc
71 deltaZstr = num2str(deltaZ);
72 yText = strcat('Height above the lowest point [m]');
73
74
75 % Extracting the field at distance of interest:
76 xCoord = find(xVect >= maxX ,1);
77 eFieldSSA = zeros(1,((maxZcompare +1)/deltaZ));
78 eFieldFDM = zeros(1,((maxZcompare +1)/deltaZ));
79 for i = 1:(maxZcompare +1)
80     eFieldSSA(1,i) = uValuesSplitStep(i,xCoord);
81     eFieldFDM(1,i) = uValuesFDM(i,xCoord);
82 end
83 simulationType{1,1} = ['SSA wedge'];
84
85 simulationType{2,1} = ['FDM wedge'];
86 tx = antHeight;
87 rx = antHeight;
88
89 eFieldTot = eFieldSSA;
90 eFieldTot = vertcat(eFieldTot,eFieldFDM);
91
92
93
94
95 minPlotLevel = 10^(-7);
96 for i = 1:1
97     % Plot SSA figure:
98     fig2 = figure('visible','off');
99

```

```

100     plotScale =length(zVectFDM);
101
102     part1Title = ['Split-Step Algorithm - flat surface'];
103     titleVal2 = {part1Title};
104     tic
105     uValuesAux = uValuesSplitStep(1:plotScale,:);
106     uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
107     minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
108     maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
109     disp('contourf is on')
110     contourf(xVect,zVectFDM(1:plotScale,1),...
111             10.*log10(abs(uValuesAux.^2)),50)
112     hold on
113     disp('contour is on')
114     contour(xVect,zVectFDM(1:plotScale,1),...
115            10.*log10(abs(uValuesAux.^2)),50)
116     hold on
117
118     %title(titleVal2)
119     xlabel('Distance [m]');
120     ylabel(yText);
121     grid on
122
123     titleFig = ['WedgeComparison.Hviid.Results/SSA_field_freq-',...
124               num2str(frequency/(10^6)), '_HviidWedge.png'];
125     disp('saving ...')
126     saveas(fig2,titleFig,'png');
127     titleFig = titleFig(1:(length(titleFig)-4));
128     titleFig = horzcat(titleFig,'.pdf');
129     save2pdf(titleFig);
130     toc
131
132     tic
133     % Plot FDM figure:
134     fig = figure('visible','off');
135     part1Title = ['Finite-Difference Method - flat surface'];
136     titleVal2 = {part1Title};
137
138     uValuesAux = uValuesFDM(1:plotScale,:);
139     uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
140     minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
141     maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
142     contourf(xVect,zVectFDM(1:plotScale,1),...
143             10.*log10(abs(uValuesAux.^2)),50)
144     hold on
145     contour(xVect,zVectFDM(1:plotScale,1),...
146            10.*log10(abs(uValuesAux.^2)),50)
147     %title(titleVal2)

```

```

148     hold on
149     xlabel('Distance [m]');
150     ylabel(yText);
151     grid on
152     titleFig = ['WedgeComparison.Hviid.Results/FDM-field-freq-',...
153               num2str(frequency/(10^6)), '_HviidWedge.png'];
154     saveas(fig,titleFig,'png');
155     titleFig = titleFig(1:(length(titleFig)-4));
156     titleFig = horzcat(titleFig, '.pdf');
157     save2pdf(titleFig);
158     toc
159
160 end
161
162 % Zoomed plots:
163 for i = 1:1
164     % Plot SSA figure:
165     fig2 = figure('visible','off');
166
167     plotScale =ceil(0.35*length(zVectFDM)); % ceil(2*length(zVectFDM)/3)
168
169     part1Title = ['Split-Step Algorithm - flat surface'];
170     titleVal2 = {part1Title};
171     tic
172     uValuesAux = uValuesSplitStep(1:plotScale,:);
173     uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
174     minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
175     maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
176     disp('contourf is on')
177     contourf(xVect,zVectFDM(1:plotScale,1),...
178             10.*log10(abs(uValuesAux.^2)),50)
179     hold on
180     disp('contour is on')
181     contour(xVect,zVectFDM(1:plotScale,1),...
182            10.*log10(abs(uValuesAux.^2)),50)
183     hold on
184
185     %title(titleVal2)
186     xlabel('Distance [m]');
187     ylabel(yText);
188     grid on
189
190     titleFig = ['WedgeComparison.Hviid.Results/SSA-field-freq-',...
191               num2str(frequency/(10^6)), '_HviidWedge-zoomed.png'];
192     disp('saving ...')
193     saveas(fig2,titleFig,'png');
194     titleFig = titleFig(1:(length(titleFig)-4));
195     titleFig = horzcat(titleFig, '.pdf');

```

```

196     save2pdf(titleFig);
197     toc
198
199     tic
200     % Plot FDM figure:
201     fig = figure('visible','off');
202     part1Title = ['Finite-Difference Method - flat surface'];
203     titleVal2 = {part1Title};
204
205     uValuesAux = uValuesFDM(1:plotScale,:);
206     uValuesAux(abs(uValuesAux)<minPlotLevel) = minPlotLevel;
207     minVal = 10.*log10(min(min(abs(uValuesAux).^2)));
208     maxVal = 10.*log10(max(max(abs(uValuesAux).^2)));
209     contourf(xVect,zVectFDM(1:plotScale,1),...
210             10.*log10(abs(uValuesAux.^2)),50)
211     hold on
212     contour(xVect,zVectFDM(1:plotScale,1),...
213            10.*log10(abs(uValuesAux.^2)),50)
214     %title(titleVal2)
215     hold on
216     xlabel('Distance [m]');
217     ylabel(yText);
218     grid on
219     titleFig = ['WedgeComparison.Hviid.Results/FDM_field_freq-',...
220              num2str(frequency/(10^6)), '_HviidWedge_zoomed.png'];
221     saveas(fig,titleFig,'png');
222     titleFig = titleFig(1:(length(titleFig)-4));
223     titleFig = horzcat(titleFig,'.pdf');
224     save2pdf(titleFig);
225     toc
226
227 end
228
229 % Comparing with results from Indra (along the surface):
230 tx =antHeight;
231 rx = antHeight;
232 rx.xArr = xVect;
233 rx.zArr = ones(length(xVect),1).*antHeight;
234 plotTitle = ' ';
235 titleFig = ['WedgeComparison.Hviid.Results/SSA-FDM_freq-',...
236           num2str(frequency/(10^6)), '_HviidWedge.png'];
237 filename = 'IndraWedge2.results/LPDA-u-kile-2.E.xls';
238
239 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
240 rx.zArr = zVectFDM;
241
242 simulationType1 = cell(2,1);
243 simulationType1{1,1} = 'SSA';

```

```

244 simulationType1{2,1} = 'FDM';
245 compareHeight_Hviid = 250;
246 close all
247 initCompare = initialFieldFDM(1:maxZcompare +1);
248 eTest = zeros(4,length(initCompare));
249
250 for i = 1:4
251     eTest(1,:) = initCompare'./eFieldTot(1,:);
252 end
253
254 pathLossWedge_Hviid(antHeight, rx_xArr, rx_zArr, eFieldTot, ...
255     frequency, simulationType1, plotTitle, titleFig, ' ', compareHeight_Hviid)
256
257 % Calculating the field along the flat surface
258 theta0 = 0;
259 Xn.flat = [0 maxX];
260 Zn.flat = [0 0];
261
262 % Creating the field (flat surface);
263 [zVectFDM, HindexFDM] = createZvectAbsorptionLayer2(...
264     maxHinterest, deltaZ, numPtsAbsorptionLayer);
265 initialFieldFDM = createInitialField(antHeight, theta0, beta, ...
266     zVectFDM, A, frequency, 'gaussian1');
267 numZpoints = length(zVectFDM);
268
269 % Calculating the fields (flat surface):
270 tic
271 uValuesSplitStepFlat = splitStepAlgorithmAbsorptionLayer(...
272     initialFieldFDM, zVectFDM, xVect, ...
273     HindexFDM, frequency, numPtsAbsorptionLayer);
274 toc
275 tic
276 [uValuesFDMalneFlat, maxEigVal] = ...
277     FDMAbsorptionLayerNumEfficient2(initialFieldFDM, ...
278     zVectFDM, xVect, HindexFDM, frequency, numPtsAbsorptionLayer);
279 toc
280
281 clear eFieldSSA;
282 clear eFieldFDM;
283
284 % Extracting the field for vertical comparison:
285 eFieldSSA = zeros(1,length(eFieldTot(1,:)));
286 eFieldFDM = zeros(1,length(eFieldTot(1,:)));
287 for i = 1:length(eFieldSSA)
288     eFieldSSA(1,i) = uValuesSplitStepFlat(i,...
289         length(xVect));
290     eFieldFDM(1,i) = uValuesFDMalneFlat(i,...
291         length(xVect));

```

```

292
293 end
294
295 simulationType{3,1} = ['SSA flat'];
296 simulationType{4,1} = ['FDM flat'];
297 eFieldTot = vertcat(eFieldTot,eFieldSSA);
298 eFieldTot = vertcat(eFieldTot,eFieldFDM);
299
300 % Comparing with results from Indra, vertical comparison:
301 tx =antHeight;
302 rx = antHeight;
303 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
304 rx.zArr = zVectFDM;
305 part1Title = ' ';%['SSA and FDM – flat surface – along the surface'];
306 plotTitle = ' '; % {part1Title};
307
308 titleFig = [...
309     'WedgeComparison.Hviid.Results/SSA.FDM.compareFlat.HviidWedge.png'];
310 filename = 'IndraWedge.results/LPDA-u-kile.E.xls';
311 numCases = 4;
312 startIndex = 4;
313 numElts = length(eFieldTot(1,:));
314 eFieldTot1 =eFieldTot;
315
316 compareHeight = 500; % maximum height of comparison
317 pathLossWedge.Hviid(antHeight,rx.xArr,rx.zArr,eFieldTot1,...
318     frequency,simulationType, plotTitle,titleFig,' ',compareHeight)
319
320 compareHeight = 50; % new maximum height of comparison
321 titleFig3 =['WedgeComparison.Hviid.Results/',...
322     'SSA.FDM.flat.height.varying.HviidWedge_zoomed1.png'];
323 pathLossFlat.Indra_minComp(rx.xArr,rx.zArr,eFieldTot1,...
324     frequency,simulationType, plotTitle,titleFig3,filename,compareHeight)
325
326 titleFig4 =['WedgeComparison.Hviid.Results/',...
327     'SSA.FDM.flat.height.varying.HviidWedge_zoomed2.png'];
328 pathLossFlat.Indra(rx.xArr,rx.zArr,eFieldTot1,...
329     frequency,simulationType, plotTitle,titleFig4,filename,compareHeight)
330
331 % Comparing along the surface, horizontal comparison:
332
333 simulationType2 = cell(4,1);
334 simulationType2{1,1} = ['SSA – wedge'];
335 simulationType2{2,1} = ['FDM – wedge'];
336 simulationType2{3,1} = ['SSA – flat'];
337 simulationType2{4,1} = ['FDM – flat'];
338 %eFieldAlongTot
339 height = 65;

```

```

340
341 startIndex = 121;
342 eSSAwedge = (uValuesSplitStep(height, startIndex:length(xVect)));
343 eFieldAlongTot = eSSAwedge;
344 eFDMwedge = (uValuesFDM(height, startIndex:length(xVect)));
345 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMwedge);
346
347 eSSAflat = (uValuesSplitStepFlat(height, startIndex:length(xVect)));
348 eFieldAlongTot = vertcat(eFieldAlongTot, eSSAflat);
349 eFDMflat = (uValuesFDMalneFlat(height, startIndex:length(xVect)));
350 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMflat);
351
352 rx.xArr3 = xVect(startIndex:length(xVect));
353 rx.zArr3 = ones(length(rx.xArr3), 1) .* height;
354 titleFig3 = ['WedgeComparison.Hviid.Results/', ...
355             'SSA-FDM_pathloss.horizontal.HviidWedge.png'];
356
357 pathLossWedge.Hviid(antHeight, rx.zArr3, rx.xArr3, eFieldAlongTot, ...
358                  frequency, simulationType2, ' ', titleFig3, ' ', 0)
359 titleFig4 = ['WedgeComparison.Hviid.Results/', ...
360             'SSA-FDM_pathloss.horizontal2.HviidWedge.png'];
361
362 pathLossIndra.alongX(rx.xArr3, rx.zArr3, eFieldAlongTot, ...
363                   frequency, simulationType2, ' ', titleFig4, ' ')
364
365
366 % Comparing fields along the surface at constant height above the surface
367 height2 = 15;
368 startIndex2 = 121;
369 fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
370 eSSAwedge2 = zeros(length(fieldVect2), 1);
371 eFDMwedge2 = zeros(length(fieldVect2), 1);
372 for i = 1:length(fieldVect2)
373     eSSAwedge2(i, 1) = uValuesSplitStep(round(zSurfaceVect(i)+height), ...
374                                       fieldVect2(i));
375     eFDMwedge2(i, 1) = uValuesFDM(round(zSurfaceVect(i)+height), ...
376                                   fieldVect2(i));
377 end
378
379 eFieldAlongTot2 = eSSAwedge2';
380 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMwedge2');
381 eSSAflat2 = (uValuesSplitStepFlat(height, startIndex2:length(xVect)));
382 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eSSAflat2);
383 eFDMflat2 = (uValuesFDMalneFlat(height, startIndex2:length(xVect)));
384 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMflat2);
385
386 titleFig5 = ['WedgeComparison.Hviid.Results/', ...
387             'SSA-FDM_pathloss.horizontal.cst.diff.surface.HviidWedge.png'];

```



```

388 rx_xArr4 = xVect(startIndex2:length(xVect));
389 rx_zArr4 = ones(length(rx_xArr4),1).*height;
390
391
392 pathLossIndra_alongX(rx_xArr4,rx_zArr4,eFieldAlongTot2,...
393     frequency,simulationType2,' ',titleFig5,' ')
394 %get(gcf,'CurrentAxes')
395 ax1 = gca;
396
397 % Plotting the wedge surface with logarithmic scale:
398 fig = figure();
399
400 semilogx((xVect(startIndex2:length(xVect))),zSurfaceVect...
401     (startIndex2:length(xVect)));
402
403 legend('Wedge surface','Location','NorthWest');
404 xlabel('Distance [m]');
405 ylabel('Height [m]');
406 plotwidth = 560;
407 plotheight = 200;
408 set(fig, 'Position', [500 100 plotwidth plotheight]);
409 grid on
410 titleFig6 = ['WedgeComparison.Hviid.Results/',...
411     'Runway_profile_small.HviidWedge.png'];
412 saveas(fig,titleFig,'png');
413 titleFig6 = titleFig6(1:(length(titleFig6)-4));
414 titleFig6 = horzcat(titleFig6,'.pdf');
415 %print (fig, '-dpdf', titleFig);
416 save2pdf(titleFig6);

```

E.1.8 Braunschweig

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

```

1 % Braunschweig.m: Script simulating the electric field over the
2 %     Braunschweig airport.
3 clear all
4 close all
5
6 % Setting the parameters:
7 theta0 = 0;
8 beta = pi/18;
9 A = 1;
10 frequency = 110*10^6;
11

```

```
12 deltaX = 1;
13 maxX = 3000;
14 numPtsAbsorptionLayer = 150;
15
16 deltaZ = 1;
17 antHeight = 3;
18 deltaXvect = [1];
19
20 counter = 0;
21 doubleCounter = 0;
22
23
24 maxHinterestHeight = 10;
25 numElts = 2;
26 simulationType = cell(numElts,1);
27
28 % Get the terrain profile:
29 xColumn = 2;
30 zColumn = 3;
31 fileName = '\IndraSource\braunschweig\Model profile.xls';
32
33 % Importing, interpolating and shifting the surface:
34 [Xn,Zn]=importParametersFromFile(fileName,xColumn,zColumn);
35 [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
36     'curve',deltaX);
37 [zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);
38
39
40 antHeight = antHeight + zSurfaceNorm(1);
41
42 %xVect = verticalVector([0:deltaX:maxX]);
43 maxHinterest = 350 +antHeight;
44
45 % Plot the surface:
46 surfacePlot = figure();
47 plot(xVect,zSurfaceNorm)
48 xlabel('Distance [m]');
49 ylabel('Surface height [m]');
50 plotwidth = 560;
51 plotheight = 200;
52 set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
53 legend('Runway surface profile','Location','SouthEast');
54 titleFig = ['Braunschweig_results/Runway-Braunschweig.png'];
55 saveas(surfacePlot,titleFig,'png');
56 titleFig = titleFig(1:(length(titleFig)-4));
57 titleFig = horzcat(titleFig,'.pdf');
58 %print (fig, '-dpdf', titleFig);
59 save2pdf(titleFig);
```

```

60
61 % Creating initial field:
62 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
63     maxHinterest,deltaZ,numPtsAbsoptioLayer);
64 initialFieldFDM = createInitialField(antHeight,theta0,beta,...
65     zVectFDM,A, frequency,'gaussian1');
66 numZpoints = length(zVectFDM);
67
68 % Calculating field:
69 tic
70 uValuesSplitStep = SSAirregularTerrainAbsoptioLayer(...
71     initialFieldFDM,zVectFDM,...
72     xVect,zSurfaceNorm,HindexFDM,frequency,...
73     numPtsAbsoptioLayer,antHeight);
74 toc
75 tic
76 uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
77     (initialFieldFDM,zVectFDM,xVect,zSurfaceNorm,...
78     HindexFDM,frequency,numPtsAbsoptioLayer);
79 toc
80 deltaZstr = num2str(deltaZ);
81 yText = strcat('Height above the lowest point [m]');
82
83 SurfEndHeight = ceil(zSurfaceNorm(length(zSurfaceNorm)));
84 fieldVect = [ceil(SurfEndHeight/deltaZ):1:floor(length(zVectFDM)/deltaZ)];
85
86 % Extracting field values for vertical comparison:
87 eFieldSSA = zeros(1,length(fieldVect));
88 eFieldFDM = zeros(1,length(fieldVect));
89 for i = 1:length(fieldVect)
90     eFieldSSA(1,i) = uValuesSplitStep(fieldVect(i),...
91         length(xVect));
92     eFieldFDM(1,i) = uValuesFDM(fieldVect(i),...
93         length(xVect));
94 end
95
96 simulationType{1,1} = ['SSA'];
97
98 simulationType{2,1} = ['FDM'];
99 tx = antHeight;
100 rx = antHeight;
101
102 eFieldTot = eFieldSSA;
103 eFieldTot = vertcat(eFieldTot,eFieldFDM);
104
105
106 % Plot SSA figure:
107 fig2 = figure('visible','off');
```

```

108 part1Title = ['Split-Step Algorithm - flat surface'];
109 titleVal2 = {part1Title};
110
111 uValuesAux = uValuesSplitStep;
112 uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
113 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
114 hold on
115 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
116 %title(titleVal2)
117 xlabel('Distance [m]');
118 ylabel(yText);
119 grid on
120 titleFig = ['Braunschweig_results/SSA.flat.along.surface.Braunschweig.png'];
121 saveas(fig2, titleFig, 'png');
122
123 % Plot FDM figure:
124 fig = figure('visible', 'off');
125 part1Title = ['Finite-Difference Method - flat surface'];
126 titleVal2 = {part1Title};
127
128 uValuesAux = uValuesFDM;
129 uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
130 contourf(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
131 hold on
132 contour(xVect, zVectFDM, 10.*log10(abs(uValuesAux.^2)), 50)
133 %title(titleVal2)
134 xlabel('Distance [m]');
135 ylabel(yText);
136 grid on
137 titleFig = ['Braunschweig_results/FDM.flat.along.surface.Braunschweig.png'];
138 saveas(fig, titleFig, 'png');
139
140 % Comparing with results from Indra, vertical comparison:
141 tx = antHeight;
142 rx = antHeight;
143 rx.xArr = ones(length(zVectFDM), 1) .* length(xVect);
144 rx.zArr = fieldVect; % zVectFDM;
145 part1Title = ' '; %['SSA and FDM - flat surface - along the surface'];
146 plotTitle = {part1Title};
147 titleFig = [...
148     'Braunschweig_results/SSA-FDM.flat.along.surface.Braunschweig.png'];
149 filename = 'IndraWedge2.results/LPDA-u-kile-2.E.xls';
150
151 pathLossWedge.Indra(Xn, Zn, deltaX, A, tx, rx.xArr, rx.zArr, eFieldTot, ...
152     antHeight, beta, frequency, simulationType, plotTitle, titleFig, ' ')
153
154 % Calculating field over flat surface:
155 tic

```

```

156 uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
157     initialFieldFDM,zVectFDM,xVect, ...
158     HindexFDM,frequency,numPtsAbsoptioLayer);
159 toc
160
161 tic
162 [uValuesFDM2,maxEigVal]=FDMAbsorptionLayerNumEfficient2(initialFieldFDM,...
163     zVectFDM,xVect,HindexFDM,frequency,numPtsAbsoptioLayer);
164 toc
165
166 % Extracting field values for vertical comparison:
167 eFieldSSA2 = zeros(1,length(fieldVect));
168 eFieldFDM2 = zeros(1,length(fieldVect));
169 for i = 1:length(fieldVect)
170     eFieldSSA2(1,i) = uValuesSSA2(fieldVect(i),...
171         length(xVect));
172     eFieldFDM2(1,i) = uValuesFDM2(fieldVect(i),...
173         length(xVect));
174 end
175
176
177 eFieldTot = vertcat(eFieldTot,eFieldSSA2);
178 eFieldTot = vertcat(eFieldTot,eFieldFDM2);
179
180 rx.xArr2 = ones(length(zVectFDM),1).*length(xVect);
181 rx.zArr2 = fieldVect;
182
183 titleFig2 = ['Braunschweig.results/',...
184     'SSA_FDM_pathloss_vertical.Braunschweig.png'];
185 filename2 = 'IndraWedge.results/LPDA-u-kile.E.xls';
186 simulationType2 = cell(4,1);
187 simulationType2{1,1} = ['SSA - runway'];
188 simulationType2{2,1} = ['FDM - runway'];
189 simulationType2{3,1} = ['SSA - flat'];
190 simulationType2{4,1} = ['FDM - flat'];
191 compareHeight = 200;
192
193 compareHeight_Hviid = 350;
194
195 % Vertical comparison
196 antHeight2 = antHeight - zSurfaceNorm(1);
197 pathLossWedge_Hviid(antHeight2,rx.xArr2,rx.zArr2,eFieldTot,...
198     frequency,simulationType2, ' ',titleFig2, ' ',compareHeight_Hviid)
199
200
201 % Compare along the surface at constant height of 40 m above the lowest
202 % point:
203

```

```

204 %eFieldAlongTot
205 height = 40;
206 startIndex = 101; %100;
207 eSSArunway = (uValuesSplitStep(height, startIndex:length(xVect)));
208 eFieldAlongTot = eSSArunway;
209 eFDMrunway = (uValuesFDM(height, startIndex:length(xVect)));
210 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMrunway);
211
212 eSSAflat = (uValuesSSA2(height, startIndex:length(xVect)));
213 eFieldAlongTot = vertcat(eFieldAlongTot, eSSAflat);
214 eFDMflat = (uValuesFDM2(height, startIndex:length(xVect)));
215 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMflat);
216
217 rx.xArr3 = xVect(startIndex:length(xVect));
218 rx.zArr3 = ones(length(rx.xArr3), 1) .* height;
219 titleFig3 = ['Braunschweig_results/', ...
220             'SSA-FDM_pathloss_horizontal_Braunschweig.png'];
221
222 pathLossWedge_Hviid(antHeight2, rx.zArr3, rx.xArr3, eFieldAlongTot, ...
223                    frequency, simulationType2, ' ', titleFig3, ' ', 0)
224 titleFig4 = ['Braunschweig_results/', ...
225             'SSA-FDM_pathloss_horizontal2_Braunschweig.png'];
226
227 pathLossIndra_alongX(rx.xArr3, rx.zArr3, eFieldAlongTot, ...
228                    frequency, simulationType2, ' ', titleFig4, ' ')
229
230
231 % Comparing fields along the surface at constant height above the surface
232 height2 = 15;
233 startIndex2 = 101;
234 fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
235 eSSArunway2 = zeros(length(fieldVect2), 1);
236 eFDMrunway2 = zeros(length(fieldVect2), 1);
237 for i = 1:length(fieldVect2)
238     %a = zSurfaceNorm(i);
239     eSSArunway2(i, 1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
240                                         fieldVect2(i));
241     eFDMrunway2(i, 1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
242                                   fieldVect2(i));
243 end
244
245 close all
246
247 eFieldAlongTot2 = eSSArunway2';
248 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMrunway2');
249 eSSAflat2 = (uValuesSSA2(height, startIndex2:length(xVect)));
250 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eSSAflat2);
251 eFDMflat2 = (uValuesFDM2(height, startIndex2:length(xVect)));

```

```

252 eFieldAlongTot2 = vertcat(eFieldAlongTot2,eFDMflat2);
253
254 eFieldAlongTot2Inv = eFieldAlongTot2;
255
256 for i =0:3
257     eFieldAlongTot2Inv(i+1,:) = eFieldAlongTot2(4-i,:);
258 end
259 simulationTypeInv = cell(4,1);
260 simulationTypeInv{4,1} = ['SSA - runway'];
261 simulationTypeInv{3,1} = ['FDM - runway'];
262 simulationTypeInv{2,1} = ['SSA - flat'];
263 simulationTypeInv{1,1} = ['FDM - flat'];
264
265 titleFig5 = ['Braunschweig.results/',...
266             'SSA-FDM.pathloss.horizontal.cst.diff.surface.Braunschweig.png'];
267 rx.xArr4 = xVect(startIndex2:length(xVect));
268 rx.zArr4 = ones(length(rx.xArr4),1).*height;
269
270 % Horizontal comparison:
271 pathLossIndra_alongX(rx.xArr4,rx.zArr4,eFieldAlongTot2Inv,...
272                     frequency,simulationTypeInv,' ',titleFig5,' ')
273
274 ax1 = gca;
275
276 % Plotting the surface profile with logarithmic scale:
277 fig = figure();
278
279 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
280          (startIndex2:length(xVect))); %,'Parent',ax2);
281
282 legend('Runway surface profile','Location','South');
283 xlabel('Distance [m]');
284 ylabel('Height [m]');
285 plotwidth = 560;
286 plotheight = 200;
287 set(fig, 'Position', [500 100 plotwidth plotheight]);
288 grid on
289 titleFig6 = ['Braunschweig.results/',...
290             'Runway_profile_small.Braunschweig.png'];
291 saveas(fig,titleFig,'png');
292 titleFig6 = titleFig6(1:(length(titleFig6)-4));
293 titleFig6 = horzcat(titleFig6,'.pdf');
294 %print (fig, '-dpdf', titleFig);
295 save2pdf(titleFig6);
296
297
298 % Plotting the line for field observation:
299 fig = figure();

```

```
300 obsLine = zSurfaceNorm +(15/deltaX);
301 plot(xVect,obsLine,'-k');
302 hold on
303 plot(xVect,zSurfaceNorm,'k');
304 hold on
305 X.slope = [0 1500];
306 Z.slope = [4 29];
307 plot(X.slope,Z.slope);
308 legendName = cell(3,1);
309 legendName{1,1} = 'Observation points';
310 legendName{2,1} = 'Runway surface profile';
311 legendName{3,1} = 'Line-of-sight line';
312 legend(legendName, 'Location', 'SouthEast');
313 %legend('Runway surface profile', 'Location', 'SouthWest');
314 xlabel('Distance [m]');
315 ylabel('Height [m]');
316 plotwidth = 560;
317 plotheight = 300;
318 set(fig, 'Position', [500 100 plotwidth plotheight]);
319 grid on
320 titleFig6 = ['Braunschweig.results/',...
321             'Runway_profile_lineOfSight_Braunschweig.png'];
322 saveas(fig,titleFig,'png');
323 titleFig6 = titleFig6(1:(length(titleFig6)-4));
324 titleFig6 = horzcat(titleFig6, '.pdf');
325 %print (fig, '-dpdf', titleFig);
326 save2pdf(titleFig6);
```

E.1.9 Luton2

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

```
1 % Luton2.m: Script simulating and comparing the electric field over the
2 %           Luton airport.
3 clear all
4
5 % Setting the parameters:
6 theta0 = 0;
7 beta = pi/18;
8 A = 1;
9 frequency = 110*10^6;
10
11 deltaX = 1;
12 maxX = 3000;
13 numPtsAbsorptionLayer = 150;
```



```

14
15 deltaZ = 1;
16 antHeight = 3;
17 deltaXvect = [1];
18
19 counter = 0;
20 doubleCounter = 0;
21 maxHinterestHeight = 10;
22 numElts = 2;
23 simulationType = cell(numElts,1);
24
25 % Get the terrain profile:
26 xColumn = 1;
27 zColumn = 3;
28 fileName = '\IndraSource\luton\Runway-profile-LOC26.xls'; %
29 '%IndraSource/luton/Runway-profile-LOC26.xls';
30
31 % Importing, interpolating, and shifting the surface height:
32 [Xn,Zn]=importParametersFromFile(fileName,xColumn,zColumn);
33
34 [xVect,zSurfaceVect] = interpolate(Xn,Zn,frequency,...
35     'curve',deltaX);
36 [zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect);
37
38 antHeight = antHeight + zSurfaceNorm(1);
39
40 maxHinterest = 350 +antHeight;
41
42 % Plot the surface:
43 surfacePlot = figure();
44 plot(xVect,zSurfaceNorm)
45 xlabel('Distance [m]');
46 ylabel('Surface height [m]');
47 plotwidth = 560;
48 plotheight = 200;
49 set(surfacePlot, 'Position', [500 100 plotwidth plotheight]);
50 legend('Runway surface profile','Location','SouthWest');
51 titleFig = ['Luton_results/Runway_Luton.png'];
52 saveas(surfacePlot,titleFig,'png');
53 titleFig = titleFig(1:(length(titleFig)-4));
54 titleFig = horzcat(titleFig, '.pdf');
55 %print (fig, '-dpdf', titleFig);
56 save2pdf(titleFig);
57
58 % Creating initial field:
59 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
60     maxHinterest,deltaZ,numPtsAbsoptionLayer);
61 initialFieldFDM = createInitialField(antHeight,theta0,beta,...

```

```

62     zVectFDM,A, frequency, 'gaussian1');
63 numZpoints = length(zVectFDM);
64
65 % Calculating field:
66 tic
67 uValuesSplitStep = SSAirregularTerrainAbsorptionLayer(...
68     initialFieldFDM, zVectFDM, ...
69     xVect, zSurfaceNorm, HindexFDM, frequency, ...
70     numPtsAbsorptionLayer, antHeight);
71 toc
72 tic
73 uValuesFDM = FDMirregularTerrainAbsorptionLayer ...
74     (initialFieldFDM, zVectFDM, xVect, zSurfaceNorm, ...
75     HindexFDM, frequency, numPtsAbsorptionLayer);
76 toc
77 deltaZstr = num2str(deltaZ);
78 yText = strcat('Height above the lowest point [m]');
79
80 SurfEndHeight = ceil(zSurfaceNorm(length(zSurfaceNorm)));
81 fieldVect = [...
82     ceil(SurfEndHeight/deltaZ):1:floor(length(zVectFDM)/deltaZ)-1]+1;
83
84 % Extracting the field values for vertical comparison:
85 eFieldSSA = zeros(1, length(fieldVect));
86 eFieldFDM = zeros(1, length(fieldVect));
87 for i = 1:length(fieldVect)
88     eFieldSSA(1,i) = uValuesSplitStep(fieldVect(i), ...
89         length(xVect));
90     eFieldFDM(1,i) = uValuesFDM(fieldVect(i), ...
91         length(xVect));
92 end
93
94 simulationType{1,1} = ['SSA'];
95 simulationType{2,1} = ['FDM'];
96 tx = antHeight;
97 rx = antHeight;
98
99 eFieldTot = eFieldSSA;
100 eFieldTot = vertcat(eFieldTot, eFieldFDM);
101
102
103 % Plot SSA field:
104 fig2 = figure('visible', 'off');
105 part1Title = ['Split-Step Algorithm - flat surface'];
106 titleVal2 = {part1Title};
107
108 uValuesAux = uValuesSplitStep;
109 uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;

```

```

110 contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
111 hold on
112 contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
113 %title(titleVal2)
114 xlabel('Distance [m]');
115 ylabel(yText);
116 grid on
117 titleFig = ['Luton.results/SSA.flat.along.surface-Luton.png'];
118 saveas(fig2,titleFig,'png');
119
120 % Plot FDM field:
121 fig = figure('visible','off');
122 part1Title = ['Finite-Difference Method - flat surface'];
123 titleVal2 = {part1Title};
124
125 uValuesAux = uValuesFDM;
126 uValuesAux(abs(uValuesAux)<10^-8) = 10^-8;
127 contourf(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
128 hold on
129 contour(xVect,zVectFDM,10.*log10(abs(uValuesAux.^2)),50)
130 %title(titleVal2)
131 xlabel('Distance [m]');
132 ylabel(yText);
133 grid on
134 titleFig = ['Luton.results/FDM.flat.along.surface-Luton.png'];
135 saveas(fig,titleFig,'png');
136
137 % Comparing with results from Indra, vertical comparison:
138 tx =antHeight;
139 rx = antHeight;
140 rx.xArr = ones(length(zVectFDM),1).*length(xVect);
141 rx.zArr = fieldVect; % zVectFDM;
142 part1Title = ' ';%['SSA and FDM - flat surface - along the surface'];
143 plotTitle = {part1Title};
144 titleFig = ['Luton.results/SSA.FDM.flat.along.surface-Luton.png'];
145 filename = 'IndraWedge2.results/LPDA-u-kile-2.E.xls';
146
147 pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx.xArr,rx.zArr,eFieldTot,...
148     antHeight,beta,frequency,simulationType,plotTitle,titleFig,' ')
149
150 % Comparing with results from a flat surface
151 antHeight2 = antHeight - zSurfaceNorm(1);
152 [zVectFDM,HindexFDM] =createZvectAbsorptionLayer2(...
153     maxHinterest,deltaZ,numPtsAbsoptioinLayer);
154 initialFieldFDM.flat = createInitialField(antHeight2,theta0,beta,...
155     zVectFDM,A, frequency,'gaussian1');
156
157 % Calculating field over flat surface:

```

```

158 tic
159 uValuesSSA2 =splitStepAlgorithmAbsorptionLayer(...
160     initialFieldFDM_flat, zVectFDM, xVect, ...
161     HindexFDM, frequency, numPtsAbsorptionLayer);
162 toc
163
164 tic
165 [uValuesFDM2, maxEigVal]=FDMAbsorptionLayerNumEfficient2(...
166     initialFieldFDM_flat, ...
167     zVectFDM, xVect, HindexFDM, frequency, numPtsAbsorptionLayer);
168 toc
169
170 % Extracting field values for vertical comparison:
171 eFieldSSA2 = zeros(1, length(fieldVect));
172 eFieldFDM2 = zeros(1, length(fieldVect));
173 for i = 1:length(fieldVect)
174     eFieldSSA2(1, i) = uValuesSSA2(fieldVect(i), ...
175         length(xVect));
176     eFieldFDM2(1, i) = uValuesFDM2(fieldVect(i), ...
177         length(xVect));
178 end
179
180
181 eFieldTot = vertcat(eFieldTot, eFieldSSA2);
182 eFieldTot = vertcat(eFieldTot, eFieldFDM2);
183
184 rx.xArr2 = ones(length(zVectFDM), 1) .* length(xVect);
185 rx.zArr2 = fieldVect;
186
187 titleFig2 = ['Luton_results/', ...
188     'SSA_FDM_pathloss_vertical_Luton.png'];
189 filename2 = 'IndraWedge_results/LPDA-u-kile.E.xls';
190 simulationType2 = cell(4, 1);
191 simulationType2{1, 1} = ['SSA - runway'];
192 simulationType2{2, 1} = ['FDM - runway'];
193 simulationType2{3, 1} = ['SSA - flat'];
194 simulationType2{4, 1} = ['FDM - flat'];
195 compareHeight = 200;
196
197 compareHeight_Hvuid = 350;
198
199 antHeight2 = antHeight - zSurfaceNorm(1);
200 % Vertical comparison:
201 pathLossWedge_Hvuid(antHeight2, rx.xArr2, rx.zArr2, eFieldTot, ...
202     frequency, simulationType2, ' ', titleFig2, ' ', compareHeight_Hvuid)
203
204
205 % Compare along the surface at constant height of 40 m above the lowest

```

```

206 % point:
207
208 height = 40;
209 startIndex = 101;
210 eSSArunway = (uValuesSplitStep(height, startIndex:length(xVect)));
211 eFieldAlongTot = eSSArunway;
212 eFDMrunway = (uValuesFDM(height, startIndex:length(xVect)));
213 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMrunway);
214
215 eSSAflat = (uValuesSSA2(height, startIndex:length(xVect)));
216 eFieldAlongTot = vertcat(eFieldAlongTot, eSSAflat);
217 eFDMflat = (uValuesFDM2(height, startIndex:length(xVect)));
218 eFieldAlongTot = vertcat(eFieldAlongTot, eFDMflat);
219
220 rx.xArr3 = xVect(startIndex:length(xVect));
221 rx.zArr3 = ones(length(rx.xArr3), 1) .* height;
222 titleFig3 = ['Luton_results/', ...
223             'SSA_FDM_pathloss_horizontal_Luton.png'];
224
225 pathLossWedge_Hviiid(antHeight2, rx.zArr3, rx.xArr3, eFieldAlongTot, ...
226                     frequency, simulationType2, ' ', titleFig3, ' ', 0)
227 titleFig4 = ['Luton_results/', ...
228             'SSA_FDM_pathloss_horizontal2_Luton.png'];
229
230 pathLossIndra_alongX(rx.xArr3, rx.zArr3, eFieldAlongTot, ...
231                    frequency, simulationType2, ' ', titleFig4, ' ')
232
233
234 % Comparing fields along the surface at constant height above the surface
235 height2 = 10;
236 startIndex2 = 101;
237 fieldVect2 = [ceil(startIndex2/deltaX):length(xVect)/deltaX];
238 eSSArunway2 = zeros(length(fieldVect2), 1);
239 eFDMrunway2 = zeros(length(fieldVect2), 1);
240 for i = 1:length(fieldVect2)
241     %a = zSurfaceNorm(i);
242     eSSArunway2(i, 1) = uValuesSplitStep(round(zSurfaceNorm(i)+height), ...
243                                         fieldVect2(i));
244     eFDMrunway2(i, 1) = uValuesFDM(round(zSurfaceNorm(i)+height), ...
245                                    fieldVect2(i));
246 end
247
248 eFieldAlongTot2 = eSSArunway2';
249 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMrunway2');
250 eSSAflat2 = (uValuesSSA2(height, startIndex2:length(xVect)));
251 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eSSAflat2);
252 eFDMflat2 = (uValuesFDM2(height, startIndex2:length(xVect)));
253 eFieldAlongTot2 = vertcat(eFieldAlongTot2, eFDMflat2);

```

```

254
255 eFieldAlongTot2Inv = eFieldAlongTot2;
256
257 for i =0:3
258     eFieldAlongTot2Inv(i+1,:) = eFieldAlongTot2(4-i,:);
259 end
260
261 titleFig5 = ['Luton.results/',...
262     'SSA_FDM_pathloss_horizontal_cst_diff_surface_Luton.png'];
263 rx.xArr4 = xVect(startIndex2:length(xVect));
264 rx.zArr4 = ones(length(rx.xArr4),1).*height;
265
266 simulationTypeInv = cell(4,1);
267 simulationTypeInv{4,1} = ['SSA - runway'];
268 simulationTypeInv{3,1} = ['FDM - runway'];
269 simulationTypeInv{2,1} = ['SSA - flat'];
270 simulationTypeInv{1,1} = ['FDM - flat'];
271
272 pathLossIndra_alongX(rx.xArr4,rx.zArr4,eFieldAlongTot2Inv,...
273     frequency,simulationTypeInv,' ',titleFig5,' ')
274
275 ax1 = gca;
276
277 % Plotting the surface with logarithmic scale:
278 fig = figure();
279
280 semilogx((xVect(startIndex2:length(xVect))),zSurfaceNorm...
281     (startIndex2:length(xVect)));
282
283 legend('Runway surface profile','Location','SouthWest');
284 xlabel('Distance [m]');
285 ylabel('Height [m]');
286 plotwidth = 560;
287 plotheight = 200;
288 set(fig, 'Position', [500 100 plotwidth plotheight]);
289 grid on
290 titleFig6 = ['Luton.results/',...
291     'Runway_profile_small_Luton.png'];
292 saveas(fig,titleFig,'png');
293 titleFig6 = titleFig6(1:(length(titleFig6)-4));
294 titleFig6 = horzcat(titleFig6,'.pdf');
295 %print (fig, '-dpdf', titleFig);
296 save2pdf(titleFig6);
297
298 % Plotting the line for field observation:
299 fig = figure();
300 obsLine = zSurfaceNorm +(15/deltaX);
301 plot(xVect,obsLine,'—k');

```

```

302 hold on
303 plot(xVect,zSurfaceNorm,'k');
304 hold on
305 X.slope = [0 1500];
306 Z.slope = [13.525 32.007];
307 plot(X.slope,Z.slope);
308 legendName = cell(3,1);
309 legendName{1,1} = 'Observation points';
310 legendName{2,1} = 'Runway surface profile';
311 legendName{3,1} = 'Line-of-sight line';
312 legend(legendName, 'Location', 'SouthWest');
313 xlabel('Distance [m]');
314 ylabel('Height [m]');
315 plotwidth = 560;
316 plotheight = 300;
317 set(fig, 'Position', [500 100 plotwidth plotheight]);
318 grid on
319 titleFig6 = ['Luton_results/',...
320             'Runway_profile_lineOfSight_Luton.png'];
321 saveas(fig,titleFig,'png');
322 titleFig6 = titleFig6(1:(length(titleFig6)-4));
323 titleFig6 = horzcat(titleFig6, '.pdf');
324 %print (fig, '-dpdf', titleFig);
325 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.

```

1
2 % FDMnoGrond.m : Calculating the electric field in free-space
3 %                 using the finite-difference method.
4 %
5 % Assumption: Propagation in vacuum, no variations in the refractive index,
6 %             n(x,z) = 1 up to the max height of consideration, for all x
7 %             and z.
8
9 % zFieldInit: Vector containing the initial electric field along the z-axis
10 % zVect: Vector containing the z-coordinates of interest

```

```
11 % xVect: Vector containing the x-coordinates of interest
12 % maxHeigthInterestZIndex: the z-index that contains the highest index of
13 %
14 %           interest, above this index there will be
15 %           absorption in order to avoid reflection from the
16 %           sky.
17 % frequency: The frequency of operation
18 % antennaIndex: The index of the center source point of the field in the
19 %           initital field.
20 %
21 % return: uValues: The u-values (electric field) for the entire
22 %           computational domain.
23 %           maxEigVal: The maximum eigenvalue of the ''system'' matrix that
24 %           is raised to the n'th power.
25 %           antennaSourceIndex: The height index of the center point of the
26 %           source for the output field.
27
28 function [uValues,maxEigVal,antennaSourceIndex] = FDMnoGround(zFieldInit, ...
29     zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer,...
30     antennaIndex)
31 c = 3*10^8;
32 lambda = c/frequency;
33 k = 2*pi/lambda;
34 numZpoints = length(zVect);
35 numXpoints = length(xVect);
36 Hindex = maxHeigthInterestZIndex;
37 xVect= verticalVector(xVect);
38 zVect = verticalVector(zVect);
39
40 deltaZ = zVect(2,1)-zVect(1,1);
41 deltaZvect = zeros(numZpoints,1);
42 deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
43 deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);
44
45 deltaX = xVect(2,1)-xVect(1,1);
46 deltaXvect = zeros(numXpoints,1);
47 deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
48 deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);
49
50 % Flipping the Gaussian beam around the source center:
51 noGroundVect = zFieldInit(antennaIndex:numZpoints,1);
52 auxVect = flipud(noGroundVect(2:length(noGroundVect),1));
53 antennaSourceIndex = length(auxVect) +1;
54
55
56 % Creating absorption layer:
57 indexRefraction = createAbsorptionLayer(zFieldInit,...
58     Hindex,numPointsInAbsLayer);
```



```

59 indexRefraction = verticalVector(indexRefraction);
60
61 auxRefraction = indexRefraction(antennaIndex:numZpoints,1);
62 auxRefraction2 = flipud(auxRefraction(2:length(auxRefraction),1));
63 indexRefraction = vertcat(auxRefraction2,auxRefraction);
64
65 zFieldInit = vertcat(auxVect,noGroundVect);
66 numZpoints = length(indexRefraction);
67
68
69
70 % Finding the 'grid' of a- and b-values:
71
72 % Vector a- and b-grid:
73
74 aGrid = zeros(numZpoints,1);
75 bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
76 aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);
77
78
79 % Calculate the u-values for successive x-values:
80 uGrid = zeros(numZpoints,numXpoints);
81 uGrid(:,1) = zFieldInit;
82 onesVect = ones(numZpoints-1,1);
83
84 % Creates VmMat:
85 diagVmat = 2 + bGrid(1,1) - aGrid(:,1);
86 diagVmat(1,1) = 1;
87 diagVmat(numZpoints,1) = 1;
88 VmMat = (diag(diagVmat) + (-1.*diag(onesVect,-1)) + (-1.*diag(onesVect,1)));
89 VmMat(1,2) = 0;
90 VmMat(numZpoints,numZpoints-1) = 0;
91
92 % Creates Amat:
93 diagAmat = -2 + bGrid(1,1) + aGrid(:,1);
94 diagAmat(1,1) = 1;
95 diagAmat(numZpoints,1) = 1;
96 AmMat = (diag(diagAmat) + diag(onesVect,-1) + diag(onesVect,1));
97 AmMat(1,2) = 0;
98 AmMat(numZpoints,numZpoints-1) = 0;
99
100 % Create transition matrix:
101 transitionMat = (VmMat/AmMat);
102
103 % Diagonalization of the transition matrix:
104 %disp('Eigenvalues start');
105 %tic
106 [eigVect,eigValues] = eig(transitionMat);

```

```
107 %toc
108
109 maxEigVal=max(max(abs(eigValues)))
110
111 % Inversing eigVect using pseudo inverse:
112 eigVectInv = sparse(pinv(eigVect));
113 eigVect = sparse(eigVect);
114 eigValues = sparse(eigValues);
115
116 initEigInv = sparse(eigVectInv*zFieldInit);
117
118
119 auxEigValues = eigValues;
120 parfor x = 2:numXpoints
121     auxEigValues = eigValues.^x;
122     uGrid(:,x) = eigVect*auxEigValues*initEigInv;
123
124 end
125
126 uValues = uGrid;
127 end
```

E.2.1.2 splitStepAlgorithmAbsorptionLayer

Field simulation over a flat surface using the SSA.

```
1 % splitStepAlgorithmAbsorptionLayer.m: Calculates the electric field over a
2 %                                     flat surface using the split-step
3 %                                     algorithm. The algorithm includes an
4 %                                     absorption layer at the top.
5 %
6 % Remarks: length(xVect) >= length(zVect)
7
8 % zFieldInit: Vector containing the initial electric field along the z-axis
9 % zVect: Vector containing the z-coordinates of interest
10 % xVect: Vector containing the x-coordinates of interest
11 % maxHeigthInterestZIndex: the z-index that contains the highest index of
12 %                           interest, above this index there will be
13 %                           absorption in order to avoid reflection from the
14 %                           sky.
15 % frequency: The frequency of operation
16 % numPtsAbsoptioinLayer: The number of height points in the absorption
17 %                           layer.
18
19 function uValues = splitStepAlgorithmAbsorptionLayer(zFieldInit,zVect, ...
```

```

20     xVect,maxHeightInterestZIndex,frequency,numPtsAbsorptionLayer)
21
22     c = 3*10^8;
23     lambda = c/frequency;
24     k = 2*pi/lambda;
25     numZpoints = length(zVect);
26     numXpoints = length(xVect);
27     Hindex = maxHeightInterestZIndex;
28     xVect = verticalVector(xVect);
29     zVect = verticalVector(zVect);
30
31
32     deltaZvect = zeros(numZpoints,1);
33     deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
34     deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);
35
36     deltaXvect = zeros(numXpoints,1);
37     deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
38     deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);
39
40     uGrid = zeros(numZpoints,numXpoints);
41     uGrid(:,1)= zFieldInit;
42     indicesZminusOne = verticalVector(linspace(1,(numZpoints-1),...
43         (numZpoints-1)));
44     Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
45         .*deltaXvect(1:1,1)./(2.*k.*(numZpoints.^2)));
46
47
48     Pprime = verticalVector(Pprime);
49
50     % Creating the absorption layer:
51     indexRefraction = createAbsorptionLayer(zFieldInit,...
52         Hindex,numPtsAbsorptionLayer);
53
54     % Calculating the field:
55     for x = 2:numXpoints
56
57         uGrid(1:(numZpoints-1),x) = inverseDiscreteSineTransform( ...
58             Pprime.*discreteSineTransform(uGrid(1:(numZpoints-1),x-1)));
59         uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 -1) ...
60             .*deltaXvect(1,1)./2).*uGrid(1:numZpoints,x);
61     end
62     uValues = uGrid;
63
64 end

```

E.2.1.3 FDMAbsorptionLayerNumEfficient2

Field simulation over a flat surface using the FDM.

```

1
2 % FDMAbsorptionLayerNumEfficient2.m : Calculating the electric field over a
3 %                                     flat surface using the
4 %                                     finite-difference method.
5
6 % zFieldInit: Vector containing the initial electric field along the z-axis
7 % zVect: Vector containing the z-coordinates of interest
8 % xVect: Vector containing the x-coordinates of interest
9 % maxHeigthInterestZIndex: the z-index that contains the highest index of
10 %                            interest, above this index there will be
11 %                            absorption in order to avoid reflection from the
12 %                            sky.
13 % frequency: The frequency of operation
14 % numPointsInAbsLayer: The thickness of the absorption layer in the number
15 %                            of points.
16
17 % return: uValues: The simulated field.
18 %           maxEigVal: The maximum eigenvalue of the transistion matrix,
19 %                       gives the stability of the system.
20
21 function [uValues,maxEigVal] = FDMAbsorptionLayerNumEfficient2(zFieldInit, ...
22     zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer)
23 c = 3*10^8;
24 lambda = c/frequency;
25 k = 2*pi/lambda;
26 numZpoints = length(zVect);
27 numXpoints = length(xVect);
28 Hindex = maxHeigthInterestZIndex;
29 xVect= verticalVector(xVect);
30 zVect = verticalVector(zVect);
31
32 deltaZ = zVect(2,1)-zVect(1,1);
33 deltaZvect = zeros(numZpoints,1);
34 deltaZvect(1:numZpoints-1,1)= zVect(2:numZpoints,1)-zVect(1:numZpoints-1);
35 deltaZvect(numZpoints,1) = deltaZvect(numZpoints-1,1);
36
37 deltaX = xVect(2,1)-xVect(1,1);
38 deltaXvect = zeros(numXpoints,1);
39 deltaXvect(1:numXpoints-1,1)= xVect(2:numXpoints,1)-xVect(1:numXpoints-1);
40 deltaXvect(numXpoints,1) = deltaXvect(numXpoints-1,1);
41
42 % Creating absorption layer:

```

```

43 indexRefraction = createAbsorptionLayer(zFieldInit,...
44     Hindex,numPointsInAbsLayer);
45
46 % Finding the 'grid' of a- and b-values:
47 % Vector a- and b-grid:
48
49 aGrid = zeros(numZpoints,1);
50 bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
51 aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);
52
53
54 % Creating grid for the uValues with the initial field:
55 uGrid = zeros(numZpoints,numXpoints);
56 uGrid(:,1)= zFieldInit;
57 onesVect = ones(numZpoints-1,1);
58
59 % Create Vmat:
60 diagVmat = 2 + bGrid(1,1)- aGrid(:,1);
61 diagVmat(1,1) = 1;
62 diagVmat(numZpoints,1) = 1;
63 VmMat = (diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1)));
64 VmMat(1,2) = 0;
65 VmMat(numZpoints,numZpoints-1) = 0;
66
67 % Create Amat:
68 diagAmat = -2 + bGrid(1,1)+ aGrid(:,1);
69 diagAmat(1,1) = 1;
70 diagAmat(numZpoints,1) = 1;
71 AmMat = (diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1));
72 AmMat(1,2) = 0;
73 AmMat(numZpoints,numZpoints-1) = 0;
74
75 % Creating transition marix:
76 transitionMat = (VmMat/AmMat);
77
78 % Diagonalization of the transition matrix:
79 [eigVect,eigValues] = eig(transitionMat);
80 maxEigVal=max(max(abs(eigValues)))
81
82 % Taking the pseudo-inverse of eigVect:
83 eigVectInv = sparse(pinv(eigVect));
84 eigVect = sparse(eigVect);
85 eigValues = sparse(eigValues);
86
87 initEigInv = sparse(eigVectInv*zFieldInit);
88
89 auxEigValues = eigValues;
90 % Calculating the field:

```

```
91 parfor x = 2:numXpoints
92     auxEigValues = eigValues.^x;
93     uGrid(:,x) = eigVect*auxEigValues*initEigInv;
94 end
95
96 uValues = uGrid;
97 end
```

E.2.1.4 SSA_addRloss

Field simulation over a flat surface using the SSA. Adds $\frac{1}{r}$ -loss to the results.

```
1 % SSA_addRloss.m: Calculates the electric field (u-values) along a flat
2 %                 surface using the split-step algorithm. Additional loss
3 %                 is added, (1/r), to try to create a 3D model out of the
4 %                 2D model.
5 % Remarks: length(xVect) >= length(zVect)
6
7 % zFieldInit: Vector containing the initial electric field along the z-axis
8 % zVect: Vector containing the z-coordinates of interest
9 % xVect: Vector containing the x-coordinates of interest
10 % maxHeigthInterestZIndex: the z-index that contains the highest index of
11 %                          interest, above this index there will be
12 %                          absorption in order to avoid reflection from the
13 %                          sky.
14 % frequency: The frequency of operation
15 % numPtsAbsoptioLayer: The number of height points in the absorption
16 %                      layer.
17 % antHeight: Antenna height [m]
18
19 % return: uValues: calcultaed u-values with 1/r-loss added.
20
21
22 function uValues = SSA_addRloss(zFieldInit,zVect, ...
23     xVect,maxHeigthInterestZIndex,frequency,numPtsAbsoptioLayer,...
24     antHeight)
25
26 c = 3*10^8;
27 lambda = c/frequency;
28 k = 2*pi/lambda;
29 numZpoints = length(zVect);
30 numXpoints = length(xVect);
31 Hindex = maxHeigthInterestZIndex;
32
33 xVect= verticalVector(xVect);
```

```

34 zVect = verticalVector(zVect);
35 deltaX = xVect(2,1)-xVect(1,1);
36
37 % Creating the a grid for the u-values
38 uGrid = zeros(numZpoints,numXpoints);
39 uGrid(:,1)= zFieldInit;
40 indicesZminusOne = verticalVector(linspace(1, (numZpoints-1), (numZpoints-1)));
41 Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
42     .*deltaX./(2.*k.*(numZpoints.^2)));
43 Pprime = verticalVector(Pprime);
44
45 % Creating absorption layer:
46 indexRefraction = createAbsorptionLayer(zFieldInit,...
47     Hindex,numPtsAbsorptionLayer);
48
49 % Calculating the field:
50 for x = 2:numXpoints
51     uGrid(1:(numZpoints-1),x) = inverseDiscreteSineTransform( ...
52         Pprime.*discreteSineTransform(uGrid(1:(numZpoints-1),x-1)));
53     uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 -1) ...
54         .*deltaX./2).*uGrid(1:numZpoints,x);
55
56 end
57
58 parfor x = 2:numXpoints
59     distVect = sqrt(x.^2 + (zVect-antHeight).^2);
60     % Adding additional loss (1/r):
61     uGrid(:,x) = uGrid(:,x).*(1./distVect);
62 end
63 uValues = uGrid;
64
65 end

```

E.2.1.5 FDM_addRloss

Field simulation over a flat surface using the FDM. Adds $\frac{1}{r}$ -loss to the results.

```

1
2 % FDM_addRloss.m : Calculates the electric field along a surface using the
3 %                   finite-difference method. Adds additional loss: (1/r),
4 %                   to compensate for the 2D model ('making' it 3D). Works
5 %                   for a flat surface.
6 % zFieldInit: Vector containing the initial electric field along the z-axis
7 % zVect: Vector containing the z-coordinates of interest
8 % xVect: Vector containing the x-coordinates of interest

```

```

 9 % maxHeigthInterestZIndex: the z-index that contains the highest index of
10 %                               interest, above this index there will be
11 %                               absorption in order to avoid reflection from the
12 %                               sky.
13 % frequency: The frequency of operation [Hz]
14 % maxHeigthInterestZIndex: the z-index that contains the highest index of
15 %                               interest, above this index there will be
16 %                               absorption in order to avoid reflection from the
17 %                               sky.
18 % antHeight: Antenna height [m]
19 % return: uValues: Calculated u-values (with 1/r-loss added)
20
21 function [uValues] = FDM_addRloss(zFieldInit, ...
22     zVect,xVect,maxHeigthInterestZIndex,frequency,numPointsInAbsLayer,...
23     antHeight)
24 c = 3*10^8;
25 lambda = c/frequency;
26 k = 2*pi/lambda;
27 numZpoints = length(zVect);
28 numXpoints = length(xVect);
29 Hindex = maxHeigthInterestZIndex;
30 xVect= verticalVector(xVect);
31 zVect = verticalVector(zVect);
32
33 deltaZ = zVect(2,1)-zVect(1,1);
34 deltaX = xVect(2,1)-xVect(1,1);
35
36 % Creating the absorption layer:
37 indexRefraction = createAbsorptionLayer(zFieldInit,...
38     Hindex,numPointsInAbsLayer);
39
40 % Vector a- and b-grid:
41 aGrid = zeros(numZpoints,1);
42 bGrid = 1j.*4.*k.*(deltaZ.^2)./deltaX;
43 aGrid(:,1) = k.^2.*(indexRefraction.^2 -1).*(deltaZ.^2);
44
45
46 % Preparations for calculation the u-values for successive x-values:
47 uGrid = zeros(numZpoints,numXpoints);
48 uGrid(:,1)= zFieldInit;
49 onesVect = ones(numZpoints-1,1);
50 diagVmat = 2 + bGrid(1,1)- aGrid(:,1);
51 diagVmat(1,1) = 1;
52 diagVmat(numZpoints,1) = 1;
53 VmMat = (diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1)));
54 VmMat(1,2) = 0;
55 VmMat(numZpoints,numZpoints-1) = 0;
56 diagAmat = -2 + bGrid(1,1)+ aGrid(:,1);

```



```

57 diagAmat(1,1) = 1;
58 diagAmat(numZpoints,1) = 1;
59 AmMat = (diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1));
60 AmMat(1,2) = 0;
61 AmMat(numZpoints,numZpoints-1) = 0;
62
63 %disp('Create transition matrix')
64 transitionMat = (VmMat/AmMat);
65
66 % Diagonalization of the transition matrix:
67 [eigVect,eigValues] = eig(transitionMat);
68 eigVectInv = sparse(pinv(eigVect));
69 eigVect = sparse(eigVect);
70 eigValues = sparse(eigValues);
71 initEigInv = sparse(eigVectInv*zFieldInit);
72
73
74 auxEigValues = eigValues;
75 % Calculating the field:
76 parfor x = 2:numXpoints
77     distVect = sqrt(x.^2 + (zVect-antHeight).^2);
78     auxEigValues = eigValues.^x;
79     uGrid(:,x) = eigVect*auxEigValues*initEigInv;
80
81
82 end
83 parfor x = 2:numXpoints
84     distVect = sqrt(x.^2 + (zVect-antHeight).^2);
85     % Adding additional loss (1/r):
86     uGrid(:,x) = uGrid(:,x).*(1./distVect);
87 end
88
89 uValues = uGrid;
90
91 end

```

E.2.1.6 SSAirregularTerrainAbsorptionLayer

Field simulation over irregular terrain using the SSA.

```

1 % SSAirregularTerrainAbsorptionLayer.m: Uses the split-step-algorithm to
2 %                                     calculate the wave propagation over
3 %                                     irregular terrain with an
4 %                                     absorption layer.
5 % zFieldInit: Vector containing the initial electric field along the z-axis

```

Appendix E: Implemented Code

```
6 % zVect: Vector containing the z-coordinates of interest
7 % xVect: Vector containing the x-coordinates of interest
8 % zSurfaceVect: Vector containing the z-coordinates of the surface
9 % maxHeighthInterestZIndex: the z-index that contains the highest index of
10 %
11 %           interest, above this index there will be
12 %           absorption in order to avoid reflection from the
13 %           sky.
14 % frequency: The frequency of operation
15 % numPointsLayer: The number of points in the absorption layer
16 % zs: Transmitter antenna height
17
18 % return: uValues: The calculated u-values.
19
20 function [uValues] = SSAirregularTerrainAbsorptionLayer(zFieldInit,zVect,...
21     xVect,zSurfaceVect,maxHeighthInterestZIndex,frequency,numPointsLayer,zs)
22
23 c = 3*10^8;
24 lambda = c/frequency;
25 k = 2*pi/lambda;
26 zVectInit = zVect;
27 numZpoints = length(zVect);
28 numZpointsInit = numZpoints;
29 numXpoints = length(xVect);
30 zVect = verticalVector(zVect);
31 xVect = verticalVector(xVect);
32 zSurfaceVect = verticalVector(zSurfaceVect);
33 Hindex = maxHeighthInterestZIndex;
34
35 deltaZ = zVect(2,1) - zVect(1,1);
36 deltaX = xVect(2,1) - xVect(1,1);
37
38 % Finding the maximum height of the surface:
39 [maxZ, indexMaxZ] = max(zSurfaceVect);
40
41 % The number of points to add:
42 addVect =verticalVector([zVect(numZpoints,1)+deltaZ:deltaZ:...
43     (zVect(numZpoints,1)+ maxZ+deltaZ)]);
44 zVect = vertcat(zVect,addVect);
45 numZpoints = length(zVect);
46 % Mapping the surface-vector to correspond with the heights in zVect
47 % The values of the new surface vector contains the indices corresponding
48 % to the height in the zVect
49 zSurfaceIndices = zeros(numXpoints,1);
50 for x = 1:numXpoints
51     zSurfaceIndices(x,1) = find(zVect >=zSurfaceVect(x,1),1,'first');
52 end
53
```

```

54 % Creating absorption layer:
55 indexRefraction = createAbsorptionLayer(zFieldInit,...
56     Hindex,numPointsLayer);
57
58 uGrid = zeros(numZpoints,numXpoints);
59 uGrid(1:numZpointsInit,1)= zFieldInit;
60 indicesZminusOne = verticalVector(linspace(1,(numZpointsInit-1), ...
61     (numZpointsInit-1)));
62 Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
63     .*deltaX./(2.*k.*(numZpointsInit.^2)));
64 Pprime = verticalVector(Pprime);
65
66 %% Creating the vGrid:
67 % vGrid = zeros(numZpointsInit,numXpoints);
68 %
69 %% Creating the vector containing the slopes of the terrain
70 % alphaVect = (zSurfaceVect(2:numXpoints,1)- ...
71 %     zSurfaceVect(1:(numXpoints-1),1))./deltaX;
72 % alphaVect = vertcat(0,alphaVect);
73 %
74 % zeta = zeros(numZpoints,1);
75 %
76 % vGrid(:,1) = uGrid(zSurfaceIndices(1):...
77 %     (zSurfaceIndices(1)+numZpointsInit-1),1);
78
79
80 % The staircase model:
81 zVect = zVectInit;
82 numZpoints = length(zVect);
83 uGrid = zeros(numZpoints,numXpoints);
84 uGrid(:,1)= zFieldInit;
85 indexRefraction = createAbsorptionLayer(zFieldInit,...
86     Hindex,numPointsLayer);
87 indicesZminusOne = verticalVector(...
88     linspace(1,(numZpoints-1),(numZpoints-1)));
89 Pprime = exp(-1j.*(pi.^2).*(indicesZminusOne.^2)...
90     .*deltaX./(2.*k.*(numZpoints.^2)));
91 Pprime = verticalVector(Pprime);
92
93 % Calculating the field:
94 for x = 2:numXpoints
95     numZpoints = length(zVect);
96     uGrid(1:(numZpoints-1),x) = inverseDiscreteSineTransform(...
97         Pprime.*discreteSineTransform(uGrid(1:(numZpoints-1),x-1)));
98     uGrid(1:numZpoints,x) = exp(1j.*k.*(indexRefraction.^2 -1) ...
99         .*deltaX./2).*uGrid(1:numZpoints,x);
100
101 % Adjusting to the surface:

```

```
102     if zSurfaceIndices(x,1) > 1
103         uGrid(1:(zSurfaceIndices(x,1)-1),x) = 0;
104     end
105
106 end
107 uValues = uGrid;
108 end
```

E.2.1.7 FDMirregularTerrainAbsorptionLayer

Field simulation over irregular terrain using the FDM.

```
1 % FDMirregularTerrainAbsorptionLayer.m : Calculates the electric field
2 %           using the finite difference method for a given
3 %           terrain profile. The function
4 %           has an absorbing layer at the top, preventing
5 %           the simulations to have reflections from the
6 %           sky.
7
8 % zFieldInit: Vector containing the initial electric field along the z-axis
9 % zVect: Vector containing the z-coordinates of interest
10 % xVect: Vector containing the x-coordinates of interest
11 % zSurfaceVect: Vector containing the z-coordinates of the surface
12 % maxHeigthInterestZIndex: the z-index that contains the highest index of
13 %           interest, above this index there will be
14 %           absorption in order to avoid reflection from the
15 %           sky.
16 % frequency: The frequency of operation
17 % numPointsInAborptionLayer: Number of points in the absorption layer.
18
19 % return: uValues: The calculated u-values.
20
21 function [uValues] = FDMirregularTerrainAbsorptionLayer(zFieldInit, ...
22     zVect,xVect,zSurfaceVect,maxHeigthInterestZIndex,frequency, ...
23     numPointsInAborptionLayer)
24
25 c = 3*10^8;
26 lambda = c/frequency;
27 k = 2*pi/lambda;
28 numZpoints = length(zVect);
29 numXpoints = length(xVect);
30 zSurfaceVect = verticalVector(zSurfaceVect);
31 xVect = verticalVector(xVect);
32 zVect= verticalVector(zVect);
33 Hindex = maxHeigthInterestZIndex;
```

```

34
35 % Mapping the surface-vector to correspond with the heights in zVect
36 % The values of the new surface vector contains the indices corresponding
37 % to the height in the zVect
38 zSurfaceIndices = zeros(numXpoints,1);
39 for x = 1:numXpoints
40     zSurfaceIndices(x,1) = find(zVect >=zSurfaceVect(x,1),1,'first');
41 end
42
43 % Hanning window for absorption layer:
44 % indiceValues = (verticalVector([0:1:numZpoints-Hindex])./(numZpoints-Hindex));
45 % absorptionLayer = (1 +cos(pi.*indiceValues))./2;
46 % indexRefraction = ones(numZpoints,1);
47 %
48 % indiceValues = verticalVector([1:1:(numZpoints-Hindex+1)]);
49 % indiceValues = (indiceValues./(numPointsPerLayer*numLayers));
50 indexRefraction = createAbsorptionLayer(zFieldInit,...
51     Hindex,numPointsInAporptionLayer);
52 %indexRefraction(Hindex:numZpoints,1) = indexRefraction(Hindex:numZpoints,1) ...
53 % + 1j.*absorptionLayer;
54
55 % Finding the 'grid' of a- and b-values:
56 deltaZ = zVect(2,1)-zVect(1,1);
57 deltaX = xVect(2,1)-xVect(1,1);
58 bGrid = 1j.*ones(numZpoints,1).*4.*k.*(deltaZ.^2)./deltaX;
59 aGrid = ones(numZpoints,1).*k.^2.*(indexRefraction.^2 -1).*deltaZ.^2;
60
61
62 % Calculate the u-values for successive x-values:
63 uGrid = zeros(numZpoints,numXpoints);
64 uGrid(:,1)= zFieldInit;
65 onesVect = ones(numZpoints-1,1);
66 VmMat = zeros(numZpoints,numZpoints);
67 VmRes = zeros(numZpoints,1);
68 AmMat = zeros(numZpoints,numZpoints);
69
70 diagVmat = 2 + bGrid(:,1)- aGrid(:,1);
71     diagVmat(1,1) = 1;
72     diagVmat(numZpoints,1) = 1;
73     VmMat = diag(diagVmat)+ (-1.*diag(onesVect,-1))+(-1.*diag(onesVect,1));
74     VmMat(1,2) = 0;
75     VmMat(numZpoints,numZpoints-1) = 0;
76
77     diagAmat = -2 + bGrid(:,1)+ aGrid(:,1);
78     diagAmat(1,1) = 1;
79     diagAmat(numZpoints,1) = 1;
80     AmMat = diag(diagAmat)+diag(onesVect,-1) + diag(onesVect,1);
81     AmMat(1,2)= 0;

```

```
82     AmMat(numZpoints,numZpoints-1) = 0;
83
84 for x = 2:numXpoints
85     % Filling in the Um and Vm matrices for field propagation estimation:
86
87     VmRes = VmMat*uGrid(:,x-1);
88
89
90
91     %AmMat*uGrid(:,x) = VmRes
92     uGrid(:,x) = linsolve(AmMat,VmRes);
93
94     if zSurfaceIndices(x,1) > 1
95         uGrid(1:(zSurfaceIndices(x,1)-1),x) = 0;
96         %uGrid(1:(zSurfaceIndices(x,1)),x) = 0;
97     end
98     %x
99
100 end
101
102 uValues = uGrid;
103 %uValues(abs(uValues)<10^-3) = 10^-3;
104 %uValues(abs(uValues)>10^-1) = 10^-1;
105 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.

```
1 % freeSpaceLoss.m: Calculates the free-space loss. Adjusts the beam pattern
2 %     according to the parameters (works only for rx = tx).
3 % tx: Transimtter antenna height
4 % rx: Receiver antenna height
5 % distance: The distance between the antennas, ignoring height difference
6 % eField: Calculated electric field along the surface at a given height,
7 %     zs, may contain multiple heights of consideration
8 % zs: The height eField is taken from
9 % beta: Beam-width [rad]
10 % frequency: The frequency of operation
11 % simulationType: Cell-array containing strings with the name of the
12 %     simulation types used.
```

```

13 % plotTitle: The title of the plot; Vector of strings.
14 % saveFig: String of path and filename for saving the plot result. To not
15 %         save the result, saveFig should be an empty string (''). File
16 %         extension: .png
17
18
19 function freeSpaceLoss_beamParam(A,tx,rx,xVect,eField,zs,...
20     beta,frequency,simulationType, plotTitle,saveFig)
21 c = 3*10^8;
22 lambda = c/frequency;
23 k = 2*pi/lambda;
24 numXpoints = length(xVect);
25 xVect = verticalVector(xVect);
26 numXpoints = length(xVect);
27 tx.x = xVect(1,1);
28 rx.x = xVect(numXpoints,1);
29 tx.z = tx;
30 rx.z = rx;
31 % Finding the dimensions of the incoming e-field array:
32 [lines,columns] = size(eField);
33 if(lines>columns)
34     eField = eField';
35     numCases = columns;
36 else
37     numCases = lines;
38 end
39 cmap = hsv(numCases+1);
40
41 plotNames = cell(1+length(simulationType),1);
42 plotNames{1,1} = 'Free-space Loss - analytical model';
43 for i = 1: length(simulationType);
44     plotNames{i+1,1} = simulationType{i};
45 end
46 % Calculating the received fields:
47 a = tx.z/rx.z;
48
49 xVectAux = xVect(2:numXpoints,1);
50 thetaDir = atan((rx.z-tx.z)/(abs(tx.x-xVectAux)));
51 Adirect = A.*exp(-2.*log10(2).*((thetaDir/beta).^2));
52 Atotal = Adirect;
53 Pl = (lambda./(4.*pi.*(abs(tx.x-xVectAux)))).*(Atotal./Adirect).^2;
54
55 % Save the figure?
56 saveFigVal = isempty(saveFig);
57
58 switch saveFigVal
59     case 1
60         % Not save figure

```

```

61     figure()
62     case 0
63         % Save figure
64         fig = figure('visible','on');
65     end
66
67
68     % Calculating the plane earth loss (path loss for plane earth):
69
70     Pl_dB = 20.*log10(verticalVector(P1(9:numXpoints-1,1)));
71     semilogx(xVect(9:numXpoints-1,1),Pl_dB,'Color',cmap(1,:));
72     xlabel('Distance [m]');
73     ylabel('Path Loss [-dB]');
74
75     for i = 1: numCases
76         hold on
77
78         % The simulated field:
79         pl1 = verticalVector(20.*log10(abs(eField(i,:))));
80         pl2 = - 10.*log10(xVect);
81
82         pathLoss = pl1 + pl2;
83         semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),...
84             'Color',cmap(i+1,:));
85
86     end
87     legend(plotNames);
88     title(plotTitle);
89     grid on
90
91     switch saveFigVal
92     case 0
93         % Save figure
94         titleFig = verticalVector(saveFig)';
95         regexprep(titleFig,' ','_');
96         regexprep(titleFig,'\\','_');
97         regexprep(titleFig,':','_');
98         regexprep(titleFig,'=','_');
99
100        if (isempty(strfind(titleFig,'.png')) == 1)
101            % File extension needs to be added.
102            titleFig = horzcat(titleFig,'.png');
103        end
104
105        saveas(fig,titleFig,'png');
106    end
107
108    end

```


E.2.2.2 pathLossFlat_beamParam

Compares the path loss over a flat surface from the simulated fields with the analytical path loss.

```

1 % pathLossFlat.m: Calculates the path loss along a flat surface, used for
2 %                   comparing with plane earth loss model, assuming zero
3 %                   reflection loss. Adjusts the beam pattern according to
4 %                   the parameters (works only for rx = tx).
5 % tx: Transmitter antenna height
6 % rx: Receiver antenna height
7 % eField: Calculated electric field along the surface at a given height,
8 %         zs, may contain multiple heights of consideration
9 % zs: The height eField is taken from
10 % beta: Beam-width [rad]
11 % frequency: The frequency of operation
12 % simulationType: Cell-array containing strings with the name of the
13 %                 simulation types used.
14 % plotTitle: The title of the plot; Vector of strings.
15 % saveFig: String of path and filename for saving the plot result. To not
16 %         save the result, saveFig should be an empty string (''). File
17 %         extension: .png
18
19
20 function pathLossFlat_beamParam(A,tx,rx,xVect,eField,zs,...
21     beta,frequency,simulationType, plotTitle,saveFig)
22     c = 3*10^8;
23     lambda = c/frequency;
24     k = 2*pi/lambda;
25     numXpoints = length(xVect);
26     xVect = verticalVector(xVect);
27     numXpoints = length(xVect);
28     tx_x = xVect(1,1);
29     rx_x = xVect(numXpoints,1);
30     tx_z = tx;
31     rx_z = rx;
32 % Finding the dimensions of the incoming e-field array:
33 [lines,columns] = size(eField);
34 if(lines>columns)
35     eField = eField';
36     numCases = columns;
37 else
38     numCases = lines;
39 end
40 cmap = hsv(numCases+1);
41

```

```

42 plotNames = cell(1+length(simulationType),1);
43 plotNames{1,1} = 'Plane Earth Loss';
44 for i = 1: length(simulationType);
45     plotNames{i+1,1} = simulationType{i};
46 end
47 % Calculating the received fields:
48 a = tx.z/rx.z;
49
50 xVectAux = xVect(2:numXpoints,1);
51 thetaDir = atan((rx.z-tx.z)/(abs(tx.x-xVectAux)));
52 Adirect = A.*exp(-2.*log10(2).*((thetaDir/beta).^2));
53
54 % The reflected field:
55
56 % Find theta:
57 d.tx = a.*abs(tx.x-xVectAux)/(1+a);
58 thetaRefl = (pi/2) - atan(d.tx./tx.z);
59 Areflect = A.*exp(-2.*log10(2).*((thetaRefl./beta).^2));
60
61 Atotal = Adirect + Areflect.*exp(1j.*k.*rx.z.*tx.z./(abs(tx.x-xVectAux)));
62
63 Pl = (lambda./(4.*pi.*(abs(tx.x-xVectAux)))).*(abs(Atotal./Adirect));
64
65 % Save the figure?
66 saveFigVal = isempty(saveFig);
67
68 switch saveFigVal
69     case 1
70         % Not save figure
71         figure()
72     case 0
73         % Save figure
74         fig = figure('visible','on');
75 end
76
77
78 % Calculating the plane earth loss (path loss for plane earth):
79 Pl_dB = 20.*log10(verticalVector(Pl(9:numXpoints-1,1)));
80 semilogx(xVect(9:numXpoints-1,1),Pl_dB,'Color',cmap(1,:));
81 xlabel('Distance [m]');
82 ylabel('Path Loss [-dB]');
83 for i = 1: 2:(numCases-1)
84     % Calculating plane earth loss from the simulated results:
85     hold on
86
87     pl1 = verticalVector(20.*log10(abs(eField(i,:))));
88     pl2 = - 10.*log10(xVect);
89

```

```

90     pathLoss = p11 + p12;
91     semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),...
92         'Color',cmap(i+1,:));
93
94     p11 = verticalVector(20.*log10(abs(eField(i+1,:))));
95     p12 = - 10.*log10(xVect);
96
97     pathLoss = p11 + p12;
98     semilogx(xVect(10:numXpoints,1),pathLoss(10:numXpoints,1),'—',...
99         'Color',cmap(i+1,:));
100
101 end
102
103 legend(plotNames,'Location','SouthWest');
104 title(plotTitle);
105 grid on
106
107 switch saveFigVal
108     case 0
109         % Save figure
110         titleFig = verticalVector(saveFig)';
111         regexprep(titleFig,' ','_');
112         regexprep(titleFig,'\\','_');
113         regexprep(titleFig,':','_');
114         regexprep(titleFig,'=','_');
115
116         if (isempty(strfind(titleFig,'.png')) == 1)
117             % File extension needs to be added.
118             titleFig = horzcat(titleFig,'.png');
119         end
120
121         saveas(fig,titleFig,'png');
122         %print (fig, '-dpdf', titleFig);
123         titleFig = titleFig(1:(length(titleFig)-4));
124         titleFig = horzcat(titleFig,'.pdf');
125         save2pdf(titleFig);
126     end
127
128 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.

```
1 % pathLossIndra_alongX.m: Compare the relative field strengths from the
2 %                               simulated results with the results from a
3 %                               given file. Horizontal comparison.
4
5 % rx_xArr: Receiver antenna (observation points) coordinates along the
6 %           x-axis.
7 % rx_zArr: Receiver antenna (observation points) coordinates along the
8 %           z-axis.
9 % eField: Calculated electric field at the coordinates to rx_xArr and
10 %         rx_zArr.
11 % frequency: The frequency of operation
12 % simulationType: Cell-array containing strings with the name of the
13 %                 simulation types used.
14 % plotTitle: Title of the plot.
15 % saveFig: file name of the plot, leave blank ( ' ' ) if the plot should not
16 %         be saved.
17 % compareFile: File name and relative path if necessary of the file to
18 %              compare results with. Leave blank ( ' ' ) if there is no file
19 %              to compare with.
20
21
22 function pathLossIndra_alongX(rx_xArr, rx_zArr, eField, ...
23     frequency, simulationType, plotTitle, saveFig, compareFile)
24 c = 3*10^8;
25 lambda = c/frequency;
26 k = 2*pi/lambda;
27 columnNumber = 2;
28 if compareFile ~= ' '
29
30     [eFieldValues, zVectIndra]=importFieldResultsFromFile(...
31         compareFile, columnNumber);
32 end
33
34 rx_zArr = verticalVector(rx_zArr);
35 numRxPoints = length(rx_zArr);
36
37 thisFontSize = 10;
38
39
40 % Finding the dimensions of the incoming e-field array:
41 [lines, columns] = size(eField);
42 if(lines>columns)
43     eField = eField';
44     numCases = columns;
45 else
46     numCases = lines;
47 end
48 cmap = hsv(numCases+1);
```

```

49
50 if compareFile ~= ' '
51     plotNames = cell(1+length(simulationType),1);
52     plotNames{1,1} = 'E-field Indra';
53     for i = 1: length(simulationType);
54         plotNames{i+1,1} = simulationType{i};
55     end
56 else % No file to compare with
57     plotNames =simulationType;
58 end
59
60 % Save the figure?
61 saveFigVal = isempty(saveFig);
62
63 switch saveFigVal
64     case 1
65         % Not save figure
66         %figure()
67     case 0
68         % Save figure
69         fig = figure('visible','on');
70 end
71 lineWidth = 1;
72 if compareFile ~= ' '
73     Pl =eFieldValues; % In dB already
74     Pl_dB =Pl;
75     semilogx(zVectIndra,Pl_dB,'Color',cmap(1,:), 'LineWidth',lineWidth);
76     minCompareValue = min(Pl_dB);
77 end
78 ax1 = gca;
79 set(ax1,'XColor','k','YColor','k')
80 %legend(plotNames{1,1})
81 set(ax1, 'Box', 'off')
82 set(ax1, 'Color', 'none')
83 for i = 1:numCases
84     if compareFile ~= ' '
85         hold on
86     elseif i > 1
87         hold on
88     end
89     p11 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
90     p12 = 0; % - 10.*log10(xVect);
91     minValueField = min(p11);
92
93     pathLoss = p11 + p12;
94
95
96     semilogx(rx.xArr,pathLoss,'Color',cmap(5-i+1,:), 'LineWidth',lineWidth);

```

```
97
98
99 end
100 grid on
101 set(gca,'FontSize',thisFontSize)
102 legend(plotNames,'Location','NorthEast')
103 ylabel('Relative field strength [dB]');
104 xlabel('Distance [m]');
105
106 switch saveFigVal
107     case 0
108         % Save figure
109         titleFig = verticalVector(saveFig)';
110         regexprep(titleFig,' ','_');
111         regexprep(titleFig,'\\','_');
112         regexprep(titleFig,':','_');
113         regexprep(titleFig,'=','_');
114
115         if (isempty(strfind(titleFig,'.png')) == 1)
116             % File extension needs to be added.
117             titleFig = horzcat(titleFig,'.png');
118         end
119
120         saveas(fig,titleFig,'png');
121         titleFig = titleFig(1:(length(titleFig)-4));
122         titleFig = horzcat(titleFig,'.pdf');
123         save2pdf(titleFig);
124     end
125
126 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.

```
1 % pathLossFlat_Indra.m: Vertical comparison between the relative field
2 %                       strengths of the simulated fields and the results
3 %                       from a given file.
4
5 % rx_xArr: Receiver antenna (observation points) coordinates along the
6 %         x-axis.
7 % rx_zArr: Receiver antenna (observation points) coordinates along the
8 %         z-axis.
9 % eField: Calculated electric field at the coordinates to rx_xArr and
```

```

10 %           rx.zArr.
11 % frequency: The frequency of operation
12 % simulationType: Cell-array containing strings with the name of the
13 %           simulation types used.
14 % plotTitle: Title of the plot.
15 % saveFig: file name of the plot, leave blank ( ' ' ) if the plot should not
16 %           be saved.
17 % compareFile: File name and relative path if necessary of the file to
18 %           compare results with.
19
20
21 function pathLossFlat_Indra(rx.xArr,rx.zArr,eField,...
22     frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
23 c = 3*10^8;
24 lambda = c/frequency;
25 k = 2*pi/lambda;
26 columnNumber = 3;
27 [eFieldValues,zVectIndra]=importFieldResultsFromFile(compareFile,...
28     columnNumber);
29 eFieldValues = verticalVector(eFieldValues);
30 zVectIndra = verticalVector(zVectIndra);
31
32 indexMaxIndra = find(zVectIndra >= compareHeight,1);
33 zVectIndra = zVectIndra(1:indexMaxIndra,1);
34 eFieldValues = eFieldValues(1:indexMaxIndra,1);
35
36 rx.xArr = verticalVector(rx.xArr);
37 rx.zArr = verticalVector(rx.zArr);
38
39 indexMaxZ = find(rx.zArr >= compareHeight,1); %240 ,1);
40 rx.xArr = rx.xArr(1:indexMaxZ,1);
41 rx.zArr = rx.zArr(1:indexMaxZ,1);
42 eField = eField(:,1:indexMaxZ,1);
43
44 rx.zArr = verticalVector(rx.zArr);
45 numRxPoints = length(rx.zArr);
46 thisFontSize = 10;
47
48
49
50 % Finding the dimensions of the incoming e-field array:
51 [lines,columns] = size(eField);
52 if(lines>columns)
53     eField = eField';
54     numCases = columns;
55 else
56     numCases = lines;
57 end

```

```

58 cmap = hsv(numCases+1);
59
60 plotNames = cell(1+length(simulationType),1);
61 plotNames{1,1} = 'E-field Indra';
62 for i = 1: length(simulationType);
63     plotNames{i+1,1} = simulationType{i};
64 end
65
66 % Save the figure?
67 saveFigVal = isempty(saveFig);
68
69 switch saveFigVal
70     case 1
71         % Not save figure
72         figure()
73     case 0
74         % Save figure
75         fig = figure('visible','on');
76 end
77
78 % Plotting the E-field calculated at indra:
79 lineWidth = 1;
80 Pl = eFieldValues; % In dB already
81 Pl_dB = Pl;
82 h11 = line(zVectIndra(7:length(zVectIndra),1),...
83     Pl_dB(7:length(zVectIndra),1), 'Color',cmap(1,:), 'LineWidth',lineWidth);
84 ax1 = gca;
85 set(ax1, 'XColor', 'k', 'YColor', 'k')
86 %legend(plotNames{1,1})
87 set(ax1, 'Box', 'off')
88 set(ax1, 'Color', 'none')
89 maxImportedEfield = max(eFieldValues);
90 set(gca, 'FontSize', thisFontSize)
91 xlabel('Height [m]');
92 ylabel('Relative field strength [dB]');
93
94 ax2 = axes('Position',get(ax1, 'Position'),...
95     'XAxisLocation', 'top',...
96     'YAxisLocation', 'right',...
97     'Color', 'none',...
98     'XColor', 'k', 'YColor', 'k');
99
100 for i = 1: numCases
101     hold on
102
103     % The simulated fields:
104     p11 = smooth(verticalVector(20.*log10(abs(eField(i, :)))));
105     maxThisField = max(p11);

```



```

106     maxDiff = maxImportedEfield - maxThisField;
107     p12 = 0;
108     pathLoss = p11 + p12;
109
110     pathLoss = pathLoss + maxDiff;
111
112     h12 = line(rx_zArr(4:length(rx_zArr),1),pathLoss(4:length(rx_zArr)),...
113             'Color',cmap(5-i+1,:), 'Parent',ax2, 'LineWidth',lineWidth);
114 end
115
116 set(ax2, 'Box', 'off')
117 set(gca, 'FontSize',thisFontSize)
118 legend(ax1,plotNames{1,1}, 'Location', 'South')
119 legend(ax2,plotNames{2:numCases+1,1}, 'Location', 'SouthEast')
120 title(plotTitle);
121
122 grid on
123 linkaxes([ax2 ax1], 'y');
124
125 switch saveFigVal
126     case 0
127         % Save figure
128         titleFig = verticalVector(saveFig)';
129         regexprep(titleFig, ' ', '_');
130         regexprep(titleFig, '\\', '_');
131         regexprep(titleFig, ':', '_');
132         regexprep(titleFig, '=', '_');
133
134         if (isempty(strfind(titleFig, '.png')) == 1)
135             % File extension needs to be added.
136             titleFig = horzcat(titleFig, '.png');
137         end
138
139         saveas(fig,titleFig, 'png');
140         titleFig = titleFig(1:(length(titleFig)-4));
141         titleFig = horzcat(titleFig, '.pdf');
142         %print (fig, '-dpdf', titleFig);
143         save2pdf(titleFig);
144     end
145
146 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.

```
1 % pathLossFlat_Indra_minComp.m: Vertical comparison between the relative
2 %           field strengths of the simulated fields and
3 %           the results from a given file. The
4 %           maximum height of comparison is specified in
5 %           compareHeight variable. The functions shifts
6 %           the fields to have the same minimum value.
7
8 % rx_xArr: Receiver antenna (observation points) coordinates along the
9 %           x-axis.
10 % rx_zArr: Receiver antenna (observation points) coordinates along the
11 %           z-axis.
12 % distance: The distance between the antennas, ignoring height difference
13 % eField: Calculated electric field at the coordinates to rx_xArr and
14 %           rx_zArr.
15 % frequency: The frequency of operation
16 % simulationType: Cell-array containing strings with the name of the
17 %           simulation types used.
18 % plotTitle: Title of the plot.
19 % saveFig: file name of the plot, leave blank ( ' ' ) if the plot should not
20 %           be saved.
21 % compareFile: File name and relative path if necessary of the file to
22 %           compare results with.
23 % compareHeight: The maximum height of comparison.
24
25 function pathLossFlat_Indra_minComp(rx_xArr,rx_zArr,eField,...
26     frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
27 c = 3*10^8;
28 lambda = c/frequency;
29 k = 2*pi/lambda;
30 columnNumber = 3;
31 [eFieldValues,zVectIndra]=importFieldResultsFromFile(compareFile,...
32     columnNumber);
33 eFieldValues = verticalVector(eFieldValues);
34 zVectIndra = verticalVector(zVectIndra);
35
36 indexMaxIndra = find(zVectIndra >= compareHeight,1);
37 zVectIndra = zVectIndra(1:indexMaxIndra,1);
38 eFieldValues = eFieldValues(1:indexMaxIndra,1);
39
40 rx_xArr = verticalVector(rx_xArr);
41 rx_zArr = verticalVector(rx_zArr);
42
43 indexMaxZ = find(rx_zArr >= compareHeight,1);
44 rx_xArr = rx_xArr(1:indexMaxZ,1);
45 rx_zArr = rx_zArr(1:indexMaxZ,1);
```

```

46 eField = eField(:,1:indexMaxZ,1);
47
48 rx_zArr = verticalVector(rx_zArr);
49 numRxPoints = length(rx_zArr);
50
51 thisFontSize = 10;
52
53
54 % Finding the dimensions of the incoming e-field array:
55 [lines,columns] = size(eField);
56 if(lines>columns)
57     eField = eField';
58     numCases = columns;
59 else
60     numCases = lines;
61 end
62 cmap = hsv(numCases+1);
63
64 plotNames = cell(1+length(simulationType),1);
65 plotNames{1,1} = 'E-field Indra';
66 for i = 1: length(simulationType);
67     plotNames{i+1,1} = simulationType{i};
68 end
69
70 % Save the figure?
71 saveFigVal = isempty(saveFig);
72
73 switch saveFigVal
74     case 1
75         % Not save figure
76         figure()
77     case 0
78         % Save figure
79         fig = figure('visible','on');
80
81 end
82
83 % Plotting the E-field calculated at indra:
84 lineWidth = 1;
85 startIndex_importField =14; %7; %13;
86 Pl =eFieldValues; % In dB already
87 Pl_dB =Pl;
88 h11 = line(zVectIndra(startIndex_importField:length(zVectIndra),1),...
89     Pl_dB(startIndex_importField:length(zVectIndra),1), 'Color', ...
90     cmap(1,:), 'LineWidth', lineWidth);
91 ax1 = gca;
92 set(ax1, 'XColor', 'k', 'YColor', 'k')
93 %legend(plotNames{1,1})

```

```

94 set(ax1, 'Box', 'off')
95 set(ax1, 'Color', 'none')
96 maxImportedEfield = min(P1_dB(startIndex:importField:length(zVectIndra),1));
97 set(gca,'FontSize',thisFontSize)
98 xlabel('Height [m]');
99 ylabel('Relative field strength [dB]');
100
101
102 ax2 = axes('Position',get(ax1,'Position'),...
103           'XAxisLocation','top',...
104           'YAxisLocation','right',...
105           'Color','none',...
106           'XColor','k','YColor','k');
107 startIndex = 4;
108 for i = 1: numCases
109     hold on
110
111     % The simulated fields:
112     p11 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
113     maxThisField = min(p11(startIndex:length(rx.zArr)));
114     maxDiff = maxImportedEfield - maxThisField;
115     p12 = 0;
116     pathLoss = p11 + p12;
117     pathLoss = pathLoss + maxDiff;
118
119     h12 = line(rx.zArr(startIndex:length(rx.zArr),1),...
120              pathLoss(startIndex:length(rx.zArr)),...
121              'Color',cmap(5-i+1,:), 'Parent',ax2, 'LineWidth',lineWidth);
122 end
123 set(ax2, 'Box', 'off')
124 set(gca,'FontSize',thisFontSize)
125 legend(ax1,plotNames{1,1},'Location','South')
126 legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
127 title(plotTitle);
128
129 grid on
130 linkaxes([ax2 ax1],'y');
131 linkaxes([ax2 ax1],'x');
132 switch saveFigVal
133     case 0
134         % Save figure
135         titleFig = verticalVector(saveFig)';
136         regexprep(titleFig,' ','_');
137         regexprep(titleFig,'\\','_');
138         regexprep(titleFig,':','_');
139         regexprep(titleFig,'=','_');
140
141         if (isempty(strfind(titleFig,'.png')) == 1)

```

```

142         % File extension needs to be added.
143         titleFig = horzcat(titleFig, '.png');
144     end
145
146     saveas(fig, titleFig, 'png');
147     titleFig = titleFig(1:(length(titleFig)-4));
148     titleFig = horzcat(titleFig, '.pdf');
149     %print (fig, '-dpdf', titleFig);
150     save2pdf(titleFig); % ,handle, dpi
151 end
152
153 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.

```

1 % pathLossWedge_Indra.m: Vertical comparison between the relative field
2 %           strengths of the simulated fields and the results
3 %           from a given file. Shiftes the fields to be
4 %           aligned with the maximim value of the results
5 %           imported from the file.
6 % Xn: Vector of x-values
7 % Zn: Vector of corresponding z-values
8 % deltaX: spacing between the sampling points along the x-axis, if set to 0
9 %           the spacing is according to Nyquist's theorem.
10 % tx: Transimtter antenna height
11 % rx_xArr: Receiver antenna (obseravation points) coordinates along the
12 %           x-axis.
13 % rx_zArr: Receiver antenna (observation points) coordinates along the
14 %           z-axis.
15 % distance: The distance between the antennas, ignoring height difference
16 % eField: Calculated electric field at the coordinates to rx_xArr and
17 %           rx_zArr.
18 % zs: The height eField is taken from
19 % frequency: The frequency of operation
20 % simulationType: Cell-array containing strings with the name of the
21 %           simulation types used.
22 % plotTitle: Title of the plot.
23 % saveFig: file name of the plot, leave blank ( ' ' ) if the plot should not
24 %           be saved.
25 % compareFile: File name and relative path if necessary of the file to
26 %           compare results with. Leave empty ( ' ' ) if there are no

```

```
27 %           results to compare with.
28
29
30 function pathLossWedge_Indra(Xn,Zn,deltaX,A,tx,rx_xArr,rx_zArr,eField,...
31     zs,beta,frequency,simulationType,plotTitle,saveFig,compareFile)
32 c = 3*10^8;
33 lambda = c/frequency;
34 k = 2*pi/lambda;
35 columnNumber = 3;
36 thisFontSize = 10;
37
38 tx_x = Xn(1,1);
39 tx_z = tx;
40
41 indexMaxZ = find(rx_zArr >= 250 ,1);
42 rx_xArr = rx_xArr(1:indexMaxZ);
43 rx_zArr = rx_zArr(1:indexMaxZ);
44 eField = eField(:,1:indexMaxZ);
45
46
47 rx_zArr = verticalVector(rx_zArr);
48 numRxPoints = length(rx_zArr);
49
50
51 % Finding the dimensions of the incoming e-field array:
52 [lines,columns] = size(eField);
53 if(lines>columns)
54     eField = eField';
55     numCases = columns;
56 else
57     numCases = lines;
58 end
59 cmap = hsv(numCases+1);
60
61 % Save the figure?
62 saveFigVal = isempty(saveFig);
63
64 switch saveFigVal
65     case 1
66         % Not save figure
67         figure()
68     case 0
69         % Save figure
70         fig = figure('visible','on');
71 end
72
73 switch compareFile
74
```

```

75     case ' '
76         % Do not try to import anything
77         plotNames = cell(length(simulationType),1);
78         for i = 1: length(simulationType);
79             plotNames{i,1} = simulationType{i};
80         end
81
82         xlabel('Height [m]');
83         ylabel('Relative field strength [dB]');
84         for i = 1: numCases
85             hold on
86             pl1 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
87             pl2 = 0;
88             pathLoss = pl1 + pl2;
89             hl2 = line(rx-zArr(:,1),pathLoss,'Color',cmap(i+1,:));
90         end
91         legend(plotNames,'Location','SouthEast')
92         title(plotTitle);
93         grid on
94
95
96     otherwise
97         [eFieldValues,zVectIndra]=importFieldResultsFromFile(...
98             compareFile,columnNumber);
99
100         plotNames = cell(1+length(simulationType),1);
101         plotNames{1,1} = 'E-field Indra';
102         for i = 1: length(simulationType);
103             plotNames{i+1,1} = simulationType{i};
104         end
105
106         % Plotting the E-field calculated at indra:
107         lineWidth = 1;
108         Pl =eFieldValues; % In dB already
109         Pl_dB =Pl(4:length(Pl));
110         maxValueCompareField = max(Pl_dB);
111         hl1 = line(zVectIndra(4:length(Pl)),Pl_dB,'Color',cmap(1,:), ...
112             'LineWidth',lineWidth);
113         ax1 = gca;
114         set(ax1,'XColor','k','YColor','k')
115         set(ax1,'Box','off')
116         set(ax1,'Color','none')
117         set(gca,'FontSize',thisFontSize)
118         xlabel('Height [m]');
119         ylabel('Relative field strength [dB]');
120         ax2 = axes('Position',get(ax1,'Position'),...
121             'XAxisLocation','top',...
122             'YAxisLocation','right',...

```

```

123         'Color','none',...
124         'XColor','k','YColor','k');
125
126     for i = 1: numCases
127         hold on
128         p11 = smooth(verticalVector(20.*log10(abs(eField(i,:)))));
129         maxValueEfield = max(p11);
130         maxValDiff = maxValueCompareField - maxValueEfield;
131         p12 = 0;
132         pathLoss = p11 + p12;
133         pathLoss = pathLoss + maxValDiff;
134         h12 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i+1,:),...
135                 'Parent',ax2,'LineWidth',lineWidth);
136     end
137     set(ax2, 'Box', 'off')
138     set(gca,'FontSize',thisFontSize)
139     legend(ax1,plotNames{1,1},'Location','East'); %'South')
140     legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
141     title(plotTitle);
142     grid on
143     linkaxes([ax2 ax1],'y');
144
145 end
146
147 switch saveFigVal
148     case 0
149         % Save figure
150         titleFig = verticalVector(saveFig)';
151         regexprep(titleFig, ' ', '_');
152         regexprep(titleFig, '\\', '_');
153         regexprep(titleFig, ':', '_');
154         regexprep(titleFig, '=', '_');
155
156         if (isempty(strfind(titleFig, '.png')) == 1)
157             % File extension needs to be added.
158             titleFig = horzcat(titleFig, '.png');
159         end
160
161         saveas(fig,titleFig, 'png');
162         titleFig = titleFig(1:(length(titleFig)-4));
163         titleFig = horzcat(titleFig, '.pdf');
164         %print (fig, '-dpdf', titleFig);
165         save2pdf(titleFig);
166     end
167
168 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.

```

1 % pathLossWedge.Hviid.m: If there are no results from files to compare
2 %                       with, the path losses of the incoming fields are
3 %                       calculated and plotted for vertical comparison.
4 %                       If there are results from other files to compare
5 %                       with, the relative field strenghts are plotted for
6 %                       vertical comparison.
7
8 % antHeight: The height of the transmitter antenna.
9 % rx_xArr: Receiver antenna (obseravation points) coordinates along the
10 %         x-axis.
11 % rx_zArr: Receiver antenna (observation points) coordinates along the
12 %         z-axis.
13 % eField: Calculated electric field at the coordinates to rx_xArr and
14 %         rx_zArr.
15 % frequency: The frequency of operation
16 % simulationType: Cell-array containing strings with the name of the
17 %                 simulation types used.
18 % plotTitle: Title of the plot.
19 % saveFig: file name of the plot, leave blank ( ' ' ) if the plot should not
20 %         be saved.
21 % compareFile: File name and relative path if necessary of the file to
22 %              compare results with. Leave empty ( ' ' ) if there are no
23 %              results to compare with.
24 % compareHeight: The maximum height of interest for comparison in the
25 %               vertical direction.
26
27
28 function pathLossWedge.Hviid(antHeight, rx_xArr,rx_zArr,eField,...
29     frequency,simulationType, plotTitle,saveFig,compareFile,compareHeight)
30 c = 3*10^8;
31 lambda = c/frequency;
32 k = 2*pi/lambda;
33 columnNumber = 3;
34 thisFontSize = 10;
35 rx_xArr = verticalVector(rx_xArr);
36
37 if (compareHeight > 0)
38 indexMaxZ = find(rx_zArr >= compareHeight ,1);
39 rx_xArr = rx_xArr(1:indexMaxZ);
40 rx_zArr = rx_zArr(1:indexMaxZ);

```

```
41 eField = eField(:,1:indexMaxZ);
42 end
43
44 rx_zArr = verticalVector(rx_zArr);
45 numRxPoints = length(rx_zArr);
46
47
48 % Finding the dimensions of the incoming e-field array:
49 [lines,columns] = size(eField);
50 if(lines>columns)
51     eField = eField';
52     numCases = columns;
53 else
54     numCases = lines;
55 end
56 cmap = hsv(numCases+1);
57
58 % Save the figure?
59 saveFigVal = isempty(saveFig);
60
61 switch saveFigVal
62     case 1
63         % Not save figure
64         figure()
65     case 0
66         % Save figure
67         fig = figure('visible','on');
68 end
69
70 switch compareFile
71
72
73     case ' '
74         % Do not try to import anything
75         plotNames = cell(length(simulationType),1);
76         for i = 1: length(simulationType);
77             plotNames{i,1} = simulationType{i};
78         end
79         %figure (1)
80         hFig = fig; % figure(1);
81         set(gcf,'PaperPositionMode','auto')
82         xwidth =450; % 360;
83         ywidth = 480;
84         set(hFig, 'Position',[800 200 xwidth ywidth])
85         xTick_arr = [0:20:compareHeight];
86         set(gca,'XTick',xTick_arr);
87         set(gca,'YDir','reverse');
88
```

```

89     startIndex = 3;
90     numPointsZ = length(rx_zArr);
91     rx_zArr = rx_zArr(startIndex:numPointsZ,1);
92     rx_xArr = rx_xArr(startIndex:numPointsZ,1);
93     xlabel('Receiver height [m]');
94     ylabel('Path loss [dB]');
95     distVect = (((antHeight-rx_zArr).^2) + (rx_xArr.^2)).^0.5;
96     eField=eField(:,startIndex:numPointsZ);
97     for i = 1: numCases
98         hold on
99         % The simulated fields:
100        p11 = -smooth(verticalVector(20.*log10(abs(eField(i,:)))));
101        p12 = +10.*log10(distVect)+ 20*log10(4*pi)-30*log(lambda);
102        pathLoss = (p11 + p12);
103        % Moving the pathloss up or down:
104        h12 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i,:));
105    end
106    legend(plotNames,'Location','SouthEast')
107    title(plotTitle);
108    grid on
109
110
111    otherwise
112        [eFieldValues,zVectIndra]=importFieldResultsFromFile(...
113            compareFile,columnNumber);
114
115        plotNames = cell(1+length(simulationType),1);
116        plotNames{1,1} = 'E-field Indra';
117        for i = 1: length(simulationType);
118            plotNames{i+1,1} = simulationType{i};
119        end
120
121        % Plotting the E-field calculated at indra:
122        lineWidth = 1;
123        P1 =eFieldValues; % In dB already
124        P1_dB =P1(4:length(P1));
125        maxValueCompareField = max(P1_dB);
126        h11 = line(zVectIndra(4:length(P1)),P1_dB,'Color',cmap(1,:), ...
127            'LineWidth',lineWidth);
128        ax1 = gca;
129        set(ax1,'XColor','k','YColor','k')
130        set(ax1,'Box','off')
131        set(ax1,'Color','none')
132        set(gca,'FontSize',thisFontSize)
133        xlabel('Height [m]');
134        ylabel('Relative field strength [dB]');
135        ax2 = axes('Position',get(ax1,'Position'),...
136            'XAxisLocation','top',...

```

```

137         'YAxisLocation','right',...
138         'Color','none',...
139         'XColor','k','YColor','k');
140
141     for i = 1: numCases
142         hold on
143         % The simulated fields:
144         p11 = smooth(verticalVector(20.*log10(abs(eField(i,:))));
145         maxValueEfield = max(p11);
146         maxValDiff = maxValueCompareField - maxValueEfield;
147         p12 = 0;
148         pathLoss = p11 + p12;
149         pathLoss = pathLoss + maxValDiff;
150         h12 = line(rx_zArr(:,1),pathLoss,'Color',cmap(i+1,:),...
151                 'Parent',ax2,'LineWidth',lineWidth);
152     end
153     set(ax2, 'Box', 'off')
154     set(gca,'FontSize',thisFontSize)
155     legend(ax1,plotNames{1,1},'Location','East'); %'South')
156     legend(ax2,plotNames{2:numCases+1,1},'Location','SouthEast')
157     title(plotTitle);
158     grid on
159     linkaxes([ax2 ax1],'y');
160
161 end
162
163 switch saveFigVal
164     case 0
165         % Save figure
166         titleFig = verticalVector(saveFig)';
167         regexprep(titleFig,' ','_');
168         regexprep(titleFig,'\','_');
169         regexprep(titleFig,':','_');
170         regexprep(titleFig,','=','_');
171
172         if (isempty(strfind(titleFig,'.png')) == 1)
173             % File extension needs to be added.
174             titleFig = horzcat(titleFig,'.png');
175         end
176
177         saveas(fig,titleFig,'png');
178         titleFig = titleFig(1:(length(titleFig)-4));
179         titleFig = horzcat(titleFig,'.pdf');
180         %print (fig, '-dpdf', titleFig);
181         save2pdf(titleFig);
182     end
183
184 end

```

E.2.3 Helping Functions

E.2.3.1 interpolate

Interpolates the given surface coordinates so that they get the correct spacing.

```

1 % interpolate : Interpolates the given points according to the frequency so
2 % that the sampling distance between the points in the x-direction is less
3 % than wavelength/2.
4
5 % Xn: Vector of x-values
6 % Zn: Vector of corresponding z-values
7 % frequency: The frequency of operation [Hz]
8 % mode: 'linear' or 'curve' interpolation
9 % deltaX: spacing between the sampling points along the x-axis, if set to 0
10 %           the spacing is according to Nyquist's theorem.
11
12 % Returns: X- and Z-vectors containg the interpolated values.
13
14 % (tested)
15
16 function[Xni,Zni] = interpolate(Xn,Zn, frequency,mode, deltaX)
17 c = 3*10^8; % Speed of light
18 lambda = c/frequency; % Wavelength
19
20 switch deltaX
21     case 0
22         maxSampleDist = lambda/2;
23         sampleDist = maxSampleDist - 0.1*maxSampleDist;
24     otherwise
25         sampleDist = deltaX;
26 end
27
28 % Creating array containing x-values with appropriate sampling distance:
29 Xni = [Xn(1):deltaX:Xn(length(Xn))];
30
31 % Interpolating the given values:
32 switch mode
33     case 'linear'
34         Zni = interp1(Xn,Zn,Xni);
35     case 'curve'
36         Zni = spline(Xn,Zn,Xni);
37 end
38 end

```

E.2.3.2 normalizeSurface

Shifts the altitudes of a surface so that the lowest point is situated at altitude zero.

```
1 % normalizeSurface.m: Shifts the surface heights so that lowest point is
2 %           situated at height 0.
3 % zSurfaceVect: The initial heights of the surface
4 % return:
5 %     zSurfaceNorm: The normalized heights of the surface.
6 %     truncationValue: The height level the surface is shifted with
7
8
9 function [zSurfaceNorm,truncationValue]=normalizeSurface(zSurfaceVect)
10
11 minVal = min(zSurfaceVect);
12
13 truncationValue = -minVal;
14 zSurfaceNorm = truncationValue + zSurfaceVect;
15 end
```

E.2.3.3 createZvectAbsorptionLayer2

Creates the z-vector with additional height for absorption layer

```
1 % createZvectAbsorptionLayer2.m: Calculates the z-vector based on the
2 %           maximum height of interest. Includes
3 %           additional height for absorption layer.
4 % Usage: To be used in combination with the split-step algorithm and
5 %           finite-difference method where an absorption is included.
6
7 % maxHeightInterest: The maximum height of interest
8 % deltaZ: The spacing between the elements in the z-vector
9 % numPointsInAbsorptionLayer: The number of points in the absorption layer
10
11 % return:
12 %     zVect: Vector containing the z-values
13 %     Hindex: The index of the value closest to the maximum height of
14 %           interest in the zVect.
15
16
17 function [zVect, Hindex]=createZvectAbsorptionLayer2(maxHeightInterest, ...
18     deltaZ,numPointsInAbsorptionLayer)
19 L = ceil((maxHeightInterest/deltaZ)+ (numPointsInAbsorptionLayer));
```

```

20 zVect = verticalVector([0:deltaZ:L]);
21
22 % Finding the closest index to the height of interest:
23 Hindex = ceil((L/deltaZ) +1)*(maxHeighthInterest/L);
24 end

```

E.2.3.4 createInitialField

Creates the initial field.

```

1 % initialField.m: Creates the initial field, using the specified beam
2 %           shape.
3 % zs: source height in meters
4 % theta0: elevation angle in radians
5 % beta: Half-power beamwidth (Gaussian beam)
6 % zVect: Vector containing the heights for field estimation in z-direction
7 % A: parameter for calculating free-space loss
8 % frequency: the frequency of operation
9 % source: 'gaussian1', 'gaussian2' or 'circular'
10 % return: initialField: The initial field for range x=0
11
12
13 function initialField = createInitialField(zs,theta0,beta,zVect,A,...
14     frequency, source)
15 c = 3*10^8;
16 lambda = c/frequency;
17 k = 2*pi/lambda;
18
19
20 numZpoints = length(zVect);
21 initialField = zeros(numZpoints,1);
22
23 switch source
24     case 'gaussian1'
25         initialField = A.*(k.*beta./(2.*sqrt(2.*log10(2)))) ...
26             .*exp(-1j.*k.*theta0.*zVect)...
27             .*exp(-(beta.^2)./(8.*log10(2))).*(k.^2).*((zVect-zs).^2));
28
29     case 'gaussian2'
30         theta1 = beta;
31         theta2 = theta0;
32         initialField = sqrt(k).*tan(theta1).*exp(-(k.^2)/2).*((zVect-zs).^2).* ...
33             (tan(theta1).^2).*exp(1j.*k.*(zVect-zs).*sin(theta2));
34     case 'circular'
35         auxTheta = asin(abs((zVect-zs))./A);

```

```
36 %         for x = 1:length(auxTheta)
37 %             if (auxTheta(x) > pi/2)
38 %                 auxTheta(x) = 0;
39 %             end
40 %         end
41         auxTheta(abs(auxTheta) > (pi/4)) =pi/2;
42         %auxTheta(auxTheta < (-pi/2)) =0; %pi;
43         initialField = A*cos(auxTheta);
44
45 end
46 initialField(1) = 0;
47 end
```

E.2.3.5 verticalVector

Makes a vector vertical if it was not initially.

```
1 % verticalVector.m
2 % Checks wether a vector is horizontal or vertical.
3 % If the vector is horizontal, it is transposed into being vertical.
4 % The returned vector is vertical
5
6 % Vin: Input vector (horizontal or vertical)
7 % Vout: Output vector, vertical
8
9 function [Vout] = verticalVector(Vin)
10
11 if( length(Vin(1,:)) > length(Vin(:,1)))
12     % Vin is a horizontal vector
13     Vout = Vin';
14 else
15     Vout = Vin;
16 end
17 end
```

E.2.3.6 getVerticalValues

Extracts the vertical values from the results on an inclined plane.

```
1 % getVerticalValues.m: Calculates the values in the "vertical" direction,
2 %                       the direction perpendicular to the downward inclined
3 %                       plane.
```



```

4
5 % xDiff: The distance difference in the x-direction
6 % zDiff: The height difference in the z-direction
7 % deltaX: The step size along the x-axis
8 % deltaZ: The step size along the z-axis
9 % xVect: The vector containing the x-values
10 % xDistCompare: The distance along the x-axis (if the surface was flat) where
11 %           the field comparison "takes place".
12 % maxZ: The maximum height for field comparison.
13 % uValues: Calculated grid of u-values.
14 % directionIncl: The direction of inclination: 'up' or 'down'
15
16 % return: zValues: The field values in the 'vertical' direction
17
18
19
20 function zValues = getVerticalValues(Xn,Zn,xDiff,zDiff,deltaX,deltaZ,...
21     xVect,xDistCompare,maxZ,uValues, directionIncl)
22
23 zValues = verticalVector([0:deltaZ:maxZ]); % counts from 0 to maxZ
24 xStartIndex = find(xVect >= xDistCompare ,1);
25
26 switch directionIncl
27
28     case 'down'
29         thetaAux = atan(zDiff/xDiff);
30         xDist2 = xDistCompare*cos(thetaAux);
31         xStartIndex = find(xVect >= xDist2 ,1);
32
33         theta = atan(xDiff/abs(zDiff));
34         phi = (2*pi) - pi - theta;
35         % The height difference at the xDistCompare along the plane to the flat
36         % surface:
37         %zDiffAdd = round(abs(zDiff) - abs((xDistCompare*sin(thetaAux))));
38         zDiffAdd = round(abs(Zn(1)) - abs((xDistCompare*sin(thetaAux))));
39
40     case 'up'
41         theta = atan(zDiff/xDiff);
42         phi = pi - (pi/2) - theta;
43         xDist2 = xDistCompare*cos(theta);
44         xStartIndex = find(xVect >= xDist2 ,1);
45
46         % The height at which is the plane is situated at, at xDistCompare:
47         % (the coordinate)
48         zDiffAdd = round(xDistCompare*sin(theta)/deltaZ);
49         zDiffAdd.noRounded = xDistCompare*sin(theta)/deltaZ;
50         %Finding the endpoint of the vector
51         vectorLength = maxZ;

```

```

52     totLength = vectorLength + zDiffAdd.noRounded;
53     xDiffLength = totLength.*cos(phi);
54     xDiffCoord = round(xStartIndex - (xDiffLength./deltaX));
55     zHeight = totLength.*sin(phi);
56     normVect = [(zHeight-zDiffAdd.noRounded) -xDiffLength];
57     % NB: Convention for this thesis: [z x]
58
59
60 end
61
62 xCoordVect = zeros(length(zValues),1);
63 zCoordVect = zeros(length(zValues),1);
64 for i = 1:length(zValues)
65     xCoord = xStartIndex - round(zValues(i).*deltaX.*cos(phi));
66     zCoord = zDiffAdd + round(zValues(i).*deltaZ.*sin(phi));
67     if(zCoord == 0)
68         zCoord = 1;
69     end
70
71     zValues(i,1) = uValues(zCoord,xCoord);
72
73
74     xCoordVect(i,1) = xCoord;
75     zCoordVect(i,1) = zCoord;
76 end
77
78 % % Verifying that the length of normVect is vectorLength:
79 % norm(normVect)
80 % % Verifying the dot product between the normal and the plane (should be 90
81 % % deg):
82 % surfaceVect = [(Zn(length(Zn))-Zn(1)) -(Xn(length(Xn))-Xn(1)) ];
83 % dotRes = dot(normVect,surfaceVect)/(norm(normVect)*norm(surfaceVect))
84 %
85 % figure()
86 % plot(Xn,Zn);
87 % hold on
88 % plot([xStartIndex xDiffCoord],[zDiffAdd zHeight]);
89
90 % % Finding the length of the segment between the plane and flat surface, if
91 % % the normal vector was extended.
92 % xPosNorm = xDistCompare/cos(theta);
93
94
95
96 %figure()
97 plot(Xn,Zn,'k')
98 hold on
99 plot(xCoordVect,zCoordVect,'k')

```

```

100 grid on
101
102 % Calculating the dot product:
103 surfaceVect = [-(Xn(length(Xn))-Xn(1)) (Zn(length(Zn))-Zn(1))];
104 ninetyDegVect = [-(xCoordVect(length(xCoordVect))-xCoordVect(1)) ...
105     (zCoordVect(length(zCoordVect))-zCoordVect(1))];
106
107 % The result of the dot product should be zero (or very close to)
108 dotResult = dot(surfaceVect,ninetyDegVect)/(norm(surfaceVect).*norm(ninetyDegVect))
109
110
111
112
113 end

```

E.2.3.7 createAbsorptionLayer

Creates absorption layer.

```

1 % createAbsorptionLayer.m: Creates an absorption layer for the given
2 %     parameters.
3 % intitialField: The initial field.
4 % maxHeigthInterestZIndex: The index of the maximum height of interest.
5 % numPointsInLayer: Number of points in the absorption layer
6
7 % return: indexOfRefraction: Vector containing the index of refraction in
8 %     the entire z-direction
9
10 function [indexOfRefraction] = createAbsorptionLayer(intitialField,...
11     maxHeigthInterestZIndex,numPointsInLayer)
12 numZpoints = length(intitialField);
13 Hindex = maxHeigthInterestZIndex;
14
15 indiceValues = verticalVector([1:1:(numZpoints-Hindex+1)]);
16 indiceValues = (indiceValues./(numPointsInLayer));
17 gamma0 = 0.15*0.098;
18 absorptionLayer = gamma0.*(indiceValues);
19
20 indexOfRefraction = ones(numZpoints,1);
21 indexOfRefraction(Hindex:numZpoints,1) = ...
22     indexOfRefraction(Hindex:numZpoints,1) + 1j.*absorptionLayer;
23
24 end

```

E.2.3.8 discreteSineTrans

Performs the discrete sine transform.

```
1 % discreteSineTrans.m: Performs a discrete sine transform of given set of
2 %                       values, xIn
3 % xIn: Vector containing the input values
4
5 % return: dst: Vector containing the calculated values.
6
7 function dst = discreteSineTrans(xIn)
8 numPoints = length(xIn);
9 dst = zeros(numPoints,1);
10 indices = verticalVector(linspace(1,numPoints,numPoints));
11
12 for k = 1:numPoints
13     dst(k) = sum(xIn.*sin(pi.*k.*indices./(numPoints+1)));
14 end
15 end
```

E.2.3.9 inverseDiscreteSineTrans

Performs the inverse discrete sine transform.

```
1 % inverseDiscreteSineTrans.m: Performs the inverse sine transformation on a
2 %                       given set of values, yIn
3 % yIn: Vector containing the input values
4
5 % return: idst: Vector containing the calculated values
6
7 function idst = inverseDiscreteSineTrans(yIn)
8 numPoints = length(yIn);
9 idst = zeros(numPoints,1);
10 indices = verticalVector(linspace(1,numPoints,numPoints));
11
12 for k = 1:numPoints
13     idst(k) = (2./(numPoints+1)).*sum(yIn.*sin(pi.*k.*indices./(numPoints+1)));
14 end
15 end
```

E.2.3.10 importFieldResultsFromFile

Imports results from file.

```

1 % importFieldResultsFromFile.m:Imports field results from a specified file,
2 %                               with the e-field values in column 4, the
3 %                               z-values in a column to be specified. The
4 %                               function is adjusted to the format of the
5 %                               results from Indra.
6
7 % fileName: String containing the file name
8 % column: Column number containing desired range column.
9
10 function [eFieldValues,zVect]=importFieldResultsFromFile(fileName,column)
11
12 fileValues = xlsread(fileName);
13
14 eFieldValues = fileValues(:,4);
15 zVect = fileValues(:,column);
16
17 end

```

E.2.3.11 importParametersFromFile

Imports surface coordinates from file.

```

1 % importParametersFromFile.m: Imports the surface profile
2 %                               coordinates from file.
3
4 % fileName: String containing the file name
5 % xColumnNb: The column number containing the x-coordinates.
6 % zColumnNB: The column number containing the z-coordinates.
7
8 % return: Xn: Vector containing the x-coordinates of the surface
9 %         Zn: Vector containing the z-coordinates of the surface
10
11
12 function [Xn,Zn]=importParametersFromFile(fileName,xColumnNb,zColumnNB)
13 fileValues = xlsread(fileName);
14 Xn = fileValues(:,xColumnNb);
15 Zn = fileValues(:,zColumnNB);
16 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.

```
1 %SAVE2PDF Saves a figure as a properly cropped pdf
2 %
3 %   save2pdf(pdfFileName,handle,dpi)
4 %
5 %   - pdfFileName: Destination to write the pdf to.
6 %   - handle:   (optional) Handle of the figure to write to a pdf.  If
7 %               omitted, the current figure is used. Note that handles
8 %               are typically the figure number.
9 %   - dpi: (optional) Integer value of dots per inch (DPI). Sets
10 %          resolution of output pdf. Note that 150 dpi is the Matlab
11 %          default and this function's default, but 600 dpi is typical for
12 %          production-quality.
13 %
14 %   Saves figure as a pdf with margins cropped to match the figure size.
15
16 %   (c) Gabe Hoffmann, gabe.hoffmann@gmail.com
17 %   Written 8/30/2007
18 %   Revised 9/22/2007
19 %   Revised 1/14/2007
20
21 function save2pdf(pdfFileName,handle,dpi)
22
23 % Verify correct number of arguments
24 error(nargchk(0,3,nargin));
25
26 % If no handle is provided, use the current figure as default
27 if nargin<1
28     [fileName,pathName] = uiputfile('*.pdf','Save to PDF file:');
29     if fileName == 0; return; end
30     pdfFileName = [pathName,fileName];
31 end
32 if nargin<2
33     handle = gcf;
34 end
35 if nargin<3
36     dpi = 150;
37 end
38
39 % Backup previous settings
40 prePaperType = get(handle,'PaperType');
41 prePaperUnits = get(handle,'PaperUnits');
```

```
42 preUnits = get(handle, 'Units');
43 prePaperPosition = get(handle, 'PaperPosition');
44 prePaperSize = get(handle, 'PaperSize');
45
46 % Make changing paper type possible
47 set(handle, 'PaperType', '<custom>');
48
49 % Set units to all be the same
50 set(handle, 'PaperUnits', 'inches');
51 set(handle, 'Units', 'inches');
52
53 % Set the page size and position to match the figure's dimensions
54 paperPosition = get(handle, 'PaperPosition');
55 position = get(handle, 'Position');
56 set(handle, 'PaperPosition', [0,0,position(3:4)]);
57 set(handle, 'PaperSize', position(3:4));
58
59 % Save the pdf (this is the same method used by "saveas")
60 print(handle, '-dpdf', pdfFileName, sprintf('-r%d', dpi))
61
62 % Restore the previous settings
63 set(handle, 'PaperType', prePaperType);
64 set(handle, 'PaperUnits', prePaperUnits);
65 set(handle, 'Units', preUnits);
66 set(handle, 'PaperPosition', prePaperPosition);
67 set(handle, 'PaperSize', prePaperSize);
```