Electromagnetic Scattering

A Surface Integral Equation Formulation

Rune Øistein Aas

Master of Science in Physics and Mathematics
Submission date: July 2012
Supervisor: Ingve Simonsen, IFY

Norwegian University of Science and Technology
Department of Physics

NTNU

Faculty of Natural Sciences
and Technology
Department of Physics

MASTER THESIS

RUNE ØISTEIN AAS

Electromagnetic Scattering
A Surface Integral Equation Formulation

Supervisor: Prof. Ingve Simonsen

July 2012

Preface

This Master's thesis was prepared at the Department of Physics at the Norwegian University of Science and Technology in Trondheim, Norway. The thesis completes a five year study for a Master of Science Degree in the field of Applied Physics and Mathematics with Applied Physics as the main profile.

The thesis is an extension of my specialization project report, sharing the same title, written in the autumn semester of 2011. Sections 1.1, 1.2, 2.1- 2.3, 3.1-3.4, 3.6 and 5 and Appendix A, B, C and G are based on the specialization project work. The main extensions include the code testing, the Method of Weighted Residuals, the improved integration formula, the new recursive discretization algorithm, the parallelization of the code, the evaluation criteria and all the results generated, presented and discussed.

I would like to express special thanks to my supervisor Prof. Ingve Simonsen for giving me very helpful advice throughout the process. I would also like to thank my friends for five fantastic years in Trondheim and my family for their everlasting support.

Abstract

A numerical approach to solving the problem of electromagnetic (EM) scattering on a single scatterer is studied. The problem involves calculating the total EM field in arbitrary observation points when a planar EM wave is scattered. The method considered is a surface integral equation (SIE) formulation involving the use of a dyadic Green's function. A theoretical derivation of the magnetic field integral equation and the electric field integral equation from Maxwell's equations are shown. The Method of Weighted Residuals (MWR) and Kirchhoff's Approximation (KA) with their respective domains of application are studied as ways of estimating the surface current densities.

A parallelized implementation of the SIE method including both the KA and the MWR is written using the FORTRAN language. The implementation is applied in three concrete versions of the scattering problem, all involving a spherical perfectly conducting scatterer, namely $\lambda \ll \rho$, $\lambda = \rho$ and $\lambda \gg \rho$, where λ and ρ denote respectively the wavelength of the incoming EM wave and the radius of the scatterer. The problems are divided into two separate solution categories, separated by whether or not the KA is assumed valid. A recursive discretization algorithm was found to be superior to a Delaunay triangulation algorithm due to less spread in element shape and area. The produced results fitted well considering the interference pattern and symmetry requirements with relative errors in the order of magnitude 10^{-5} and less. The case of having large wavelength compared to the radius was also compared with Rayleigh scattering theory considering the far field dependence on wavelength, scattering angle and distance from the scatterer. This resulted in relative errors of 2.1% and less.

The main advantage of the SIE method is only requiring the surface of the scatterer to be discretized thus saving computational time and memory compared to methods requiring discretization of volume. The method is also capable of producing accurate results for observation points arbitrary close to the scatterer surface. A brief discussion on how the program may be modified in order to extend its capabilities is also included.

Sammendrag

En numerisk metode for løsning av det elektromagnetiske (EM) spredningsproblemet med ett enkelt spredningsobjekt er studert. Problemet innebærer å beregne det totale EM feltet i konkrete observasjonspunkter når den inkommende EM bølgen er en planbølge. Metoden som er studert er en formulering basert på et sett overflateintegrallikninger (OIL) som innebærer utnyttelsen av en dyadisk Greens funksjon. En teoretisk utledning av integrallikningen for det elektriske feltet og integrallikningen for det magnetiske feltet er vist. Metoden for Vektede Residualer (MVR) og Kirchhoffs Approksimasjon (KA) med deres respektive bruksområder er studert som måter å estimere de tenkte overflatestrømtetthetene.

En parallellisert implementasjon av OIL-metoden inkludert både KA og MVR er skrevet i programmeringsspråket FORTRAN. Implementasjonen er anvendt på tre konkrete versjoner av spredningsproblemet hvor alle innebærer et perfekt ledende spredningsobjekt, $\lambda \ll \rho$, $\lambda = \rho$ og $\lambda \gg \rho$, hvor λ og ρ representerer henholdsvis bølgelengden til den inkommende EM bølgen og radien til spredningsobjektet. Problemene er delt inn i to løsningskategorier avhengig av om KA er gyldig. En rekursiv diskretiseringsalgoritme viste seg å være overlegen en Delaunay trianguleringsalgoritme grunnet mindre spredning i elementform og areal. Resultatene passet bra med forventningene når det gjaldt interferensmønster og symmetrikrav, med relative feil av størrelsesorden 10^{-5} og mindre. I tilfellet hvor bølgelengden var stor i forhold til radien ble resultatene også sammenlignet med Rayleigh teori for fjernfeltsavhengigheten i bølgelengde, spredningsvinkel og avstand fra spredningsobjektet. Dette resulterte i relative feil på 2.1% og mindre.

Hovedfordelen til OIL-metoden er at kun overflaten til spredningsobjektet må diskretiseres, hvilket reduserer behov for minne og prosessortid i forhold til metoder som krever diskretisering av volum. Metoden er også i stand til å produsere nøyaktige resultater for observasjonspunkter vilkårlig nær spredningsobjektet. En kort diskusjon rundt hvordan programmet kan modifiseres for å utvide dets bruksområder er også inkludert.

Contents

Chapter 1

Background and Introduction

1.1 Motivation

The theory behind scattering of electromagnetic (EM) waves explains some beautiful and mysterious natural phenomena, as for instance why the sunset looks red and the sky above looks blue. How EM waves interact with its surroundings is essential knowledge for a wide variety of professions. The radar engineer, optician, camera designer, solar cell researcher and so on all need to have fundamental knowledge of the behaviour of EM waves. Especially when it comes to sensing and detection being able to create accurate models for the process of EM scattering is very useful.

The area of nanotechnology has seen great progress during the last decades. Experimental techniques have made it possible to fabricate and characterize particles at nano scale. One of the interesting effects discovered is called Localized Surface Plasmon Resonance (LSPR), which is collective oscillations of the sea of electrons at the surface of a nanoparticle. This effect was found in 1978 to be the explanation for the huge cross section observed in Raman scattering from a pyridine adsorbed silver electrode, see Ref. [1]. This has led to the technique called surface-enhanced Raman spectroscopy. The technique utilizes the great magnification of the EM field close to a nanoparticle due to LSPR at the resonance wavelengths. Because of the large amplification of the field the method is so sensitive it may even detect single molecules.

Experimental breakthroughs as the one mentioned have urged for improvements also in the theoretical methods and solving Maxwell's equations for scattering problems has found new interest. Analytical solutions are known for a few simple geometrical shapes of the scatterer, for example the Mie solution for a spherical scatterer, see Ref. [2]. However, for more complex shapes numerical methods are needed. Especially with the advancement in computer technology the numerical techniques is becoming increasingly powerful. As technology advances the need for accurate numerical methods is growing.

Today there are several different methods for modelling scattering problems numerically. They can be divided into two main groups; volume methods and surface methods. The latter includes surface integral equation (SIE) methods which only require discretization of the surface of the scatterer, as oppose to volume methods where the scatterer volume or in some methods all space needs to be discretized. SIE methods thus save computational time and memory. One particular SIE formulation is therefore the chosen method for this project. The aim of this thesis is to present the derivation and numerical solution of one set of SIEs derived from Maxwell's equations with the ability to reproduce the LSPR of a metal nanosphere, which lately has become the primary validity criterion for the numerical methods for nanostructure scattering problems. Presenting a program implemented from scratch capable of solving special cases of the EM scattering problem is a second aim of the thesis.

The theory is presented in Chapter 2. Furthermore, Chapters 3 and 4 presents respectively the numerical implementation and the results with corresponding discussion. The appendices cover some of the most important concepts applied in the main text and also include the complete program code written in FORTRAN.

1.2 The Electromagnetic Scattering Problem

This section presents the geometry of the problem and the assumptions regarding the materials.

1.2.1 Scattering Geometry

The geometry of the EM scattering problem is illustrated in Fig. 1.1. The space is divided into two regions, with V_1 and V_2 denoting the volumes of each region respectively. Region 2 is bounded by the surface S on the outside. Region 1 is bounded by the surface S on the inside and by the surface S_{inf} on the outside. The latter is stretched out to infinity. The vectors $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are the outward directed unit normal vectors to the boundary surfaces. Therefore, at the surface, S, the relationship, $\hat{\mathbf{n}}_1 = -\hat{\mathbf{n}}_2$, is satisfied.

The source of the scattered field is an incident EM wave, $\mathbf{E}^{\text{inc}}$ and $\mathbf{H}^{\text{inc}}$. It is assumed that region 2 is non-emitting, so all incident EM waves originates from region 1. Given the scattering material properties, namely the permittivity, ϵ, and permeability, μ, and the incident EM field, $\mathbf{E}^{\text{inc}}$ and $\mathbf{H}^{\text{inc}}$, the problem consists of finding the resulting EM field, $\mathbf{E}$ and $\mathbf{H}$, everywhere in both regions.

In Fig. 1.1 the combined effect of the scattering material and the incident EM wave is absorbed into the electric and magnetic surface current densities, denoted as $\mathbf{J}$ and $\mathbf{M}$ respectively. These are imaginary currents inducing the same electric and magnetic fields as the actual physical system would generate. Thus, the problem is essentially transformed into finding the surface current densities, $\mathbf{J}$ and $\mathbf{M}$.

Fig. 1.1: Geometry of the problem:

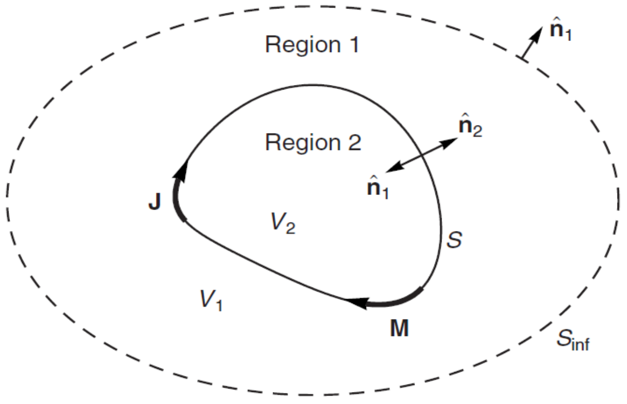

Figure 1.1: Geometry of the scattering problem considered. (From Ref. [3]).

1.2.2 Assumptions for the Scattering Material

Each region is occupied by a material satisfying the following set of properties. First, the materials are assumed homogeneous, that is the permittivity, ϵ, and the permeability, μ, are independent of position, $\mathbf{r}$, within the same region. Next, linearity is assumed,

$$D_i(\mathbf{k},\omega) = \sum_j \epsilon_{ij}(\mathbf{k},\omega)E_j(\mathbf{k},\omega), \qquad j,i \in \{x,y,z\}, \tag{1.1}$$

where $\mathbf{k}$ and ω denote respectively the wave vector and angular frequency of an EM wave travelling through the material. Furthermore, $D_i(\mathbf{k},\omega)$ and $E_j(\mathbf{k},\omega)$ are the component of the electric displacement in the direction i and the electric field in the direction j respectively. Moreover, $\epsilon_{ij}(\mathbf{k},\omega)$ denotes the permittivity tensor. Next, it is assumed that locality is satisfied, meaning the permittivity tensor, ϵ_{ij}, is independent of the wave vector, $\mathbf{k}$, which implies that the displacement, $\mathbf{D}(\mathbf{r}, \omega)$, is only dependent on the electric field, $\mathbf{E}(\mathbf{r}, \omega)$, at position $\mathbf{r}$. Also assumed is the materials being isotropic, which transforms the permittivity tensor, ϵ_{ij}, to a scalar, ϵ. The materials are thus assumed to be linear, isotropic, local and dispersive, giving the following relation between the electric field, $\mathbf{E}$, and the displacement, $\mathbf{D}$ in the space and frequency domain,

$$\mathbf{D}(\mathbf{r}, \omega) = \epsilon(\omega)\mathbf{E}(\mathbf{r}, \omega). \tag{1.2}$$

The same assumptions are made considering the permeability, μ, giving the following relation between the magnetic flux density, $\mathbf{B}$, and the magnetic field, $\mathbf{H}$, in the space and frequency domain,

$$\mathbf{B}(\mathbf{r}, \omega) = \mu(\omega)\mathbf{H}(\mathbf{r}, \omega). \tag{1.3}$$

Note that the transformation from the time domain to the frequency domain is accomplished through the Fourier transform, see appendix A for details.

Chapter 2

Theoretical Derivation of the SIE Method

This section includes the derivation of the formulas needed to perform the numerical calculation of the EM field by the SIE method.

2.1 Maxwell's Equations

Maxwell's equations are the foundation for solving all EM problems. The first equation is Gauss' law,

$$\nabla \cdot \mathbf{D}(\mathbf{r},t) = \rho_f(\mathbf{r},t)\ ,\ \text{(Gauss' law)}, \tag{2.1}$$

where $\mathbf{D}$ is the electric displacement field and ρ_f is the density of the free charge. The next equation is Faraday's law,

$$\nabla \times \mathbf{E}(\mathbf{r},t) = -\frac{\partial \mathbf{B}(\mathbf{r},t)}{\partial t}\ ,\ \text{(Faraday's law)}, \tag{2.2}$$

where $\mathbf{E}$ is the electric field and $\mathbf{B}$ is the magnetic flux density. Then, there is the equation for the divergence of the magnetic flux density,

$$\nabla \cdot \mathbf{B}(\mathbf{r},t) = 0. \tag{2.3}$$

The fourth equation is Amperes law,

$$\nabla \times \mathbf{H}(\mathbf{r},t) = \mathbf{j}(\mathbf{r},t) + \frac{\partial \mathbf{D}(\mathbf{r},t)}{\partial t}\ ,\ \text{(Ampere's law)}, \tag{2.4}$$

where $\mathbf{H}$ is the magnetic field and $\mathbf{j}$ is the free current density.

When having the linear relations between the fields in Maxwell's equations in the frequency domain given by Eqs. (1.2) and (1.3) a transformation of Maxwell's equations to the frequency domain is useful. Using the definition from Eq. (A-1) and the property given by Eq. (A-5) gives the following result for Maxwell's equations in the frequency domain,

$$\nabla \cdot \mathbf{D}(\mathbf{r},\omega) = \rho_f(\mathbf{r},\omega), \tag{2.5}$$

$$\nabla \times \mathbf{E}(\mathbf{r},\omega) = i\omega\mathbf{B}(\mathbf{r},\omega), \tag{2.6}$$

$$\nabla \cdot \mathbf{B}(\mathbf{r},\omega) = 0, \tag{2.7}$$

$$\nabla \times \mathbf{H}(\mathbf{r},\omega) = \mathbf{j}(\mathbf{r},\omega) - i\omega\mathbf{D}(\mathbf{r},\omega). \tag{2.8}$$

2.2 Derivation of the Relation Between $\mathbf{E}^{\text{inc}}$, E, J and M

The first objective is to find an equation relating the electric field, $\mathbf{E}(\mathbf{r})$ in all locations, $\mathbf{r}$, to the incoming electric field $\mathbf{E}^{\text{inc}}(\mathbf{r})$. The starting point of the derivation is Eq. (2.6), Faraday's law in the frequency domain. Applying the curl operator on both sides of the equation gives,

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r},\omega) = i\omega\nabla \times \mathbf{B}(\mathbf{r},\omega). \tag{2.9}$$

Inserting relation (1.3) into the right-hand side of this equation gives,

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r},\omega) = i\omega\mu(\omega)\nabla \times \mathbf{H}(\mathbf{r},\omega). \tag{2.10}$$

Using Ampere's law, Eq.(2.8), on the right-hand side of Eq. (2.10) now leads to

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r},\omega) = i\omega\mu(\omega)\mathbf{j}(\mathbf{r},\omega) + \mu(\omega)\omega^2\mathbf{D}(\mathbf{r},\omega). \tag{2.11}$$

Now, Equation (1.2) can be used to obtain an equation in the electric field only. The equation must be satisfied at all locations in both regions. An index $i \in \{1,2\}$ may be added indicating the region considered[1],

$$\nabla \times \nabla \times \mathbf{E}_i(\mathbf{r},\omega) - k_i^2\mathbf{E}_i(\mathbf{r},\omega) = i\omega\mu_i(\omega)\mathbf{j}(\mathbf{r},\omega), \tag{2.12}$$

where $k_i = (\epsilon_i(\omega)\mu_i(\omega))^{1/2}\omega$ is the wavenumber of the EM field in region i. Now we have a linear differential equation in position for the electric field. The ω-dependence is from now on dropped in the notation for simplicity. In

[1]Note that when i is used as a factor in an expression it denotes the imaginary unit, while used as a subscript it denotes the index of the region.

the further derivation the Green's function is introduced. The basic idea of the Green's function is shown in appendix C.1. For Eq. (2.12) the Green's function is well known, see Ref. [3]. It takes the form of a dyadic function of the position, $\mathbf{r}$, with a source at the position, $\mathbf{r}'$. The properties of a dyadic tensor are given in appendix B. The equation containing the dyadic Green's function for equation (2.12) is analogous with the general definition of a scalar Green's function given in appendix C.1,

$$\nabla \times \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - k_i^2 \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') = \bar{\mathbf{1}}\delta(\mathbf{r}-\mathbf{r}'), \tag{2.13}$$

where $\bar{\mathbf{1}} = \delta_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j)$ is the identity dyad (see appendix B.3.1 for the properties of the identity dyad) and $\delta(\mathbf{r}-\mathbf{r}')$ is the three-dimensional Dirac delta function. Multiplying this equation with $\mathbf{E}_i(\mathbf{r})$ from the left and equation (2.12) with $\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ from the right and then subtracting the resulting equations leads to

$$\begin{aligned} &\nabla \times \nabla \times \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - k_i^2 \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \\ &- (\mathbf{E}_i(\mathbf{r}) \cdot \nabla \times \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - k_i^2 \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')) \\ &= i\omega\mu_i \mathbf{j}(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{1}}\delta(\mathbf{r}-\mathbf{r}'). \end{aligned} \tag{2.14}$$

The terms with $k_i^2 \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ are cancelling. Using the property, $\mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{1}} = \mathbf{E}_i(\mathbf{r})$, therefore leads to

$$\begin{aligned} &\nabla \times \nabla \times \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r}) \cdot \nabla \times \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \\ =& i\omega\mu_i \mathbf{j}(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r})\delta(\mathbf{r}-\mathbf{r}'). \end{aligned} \tag{2.15}$$

By setting $\mathbf{a} = \nabla$, $\mathbf{b} = \nabla \times \mathbf{E}_i(\mathbf{r})$ and $\bar{\mathbf{C}} = \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ the first term of this equation may be converted by equation (B-23) to the following expression,

$$\nabla \times \nabla \times \mathbf{E}_i(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') = \nabla \cdot (\nabla \times \mathbf{E}_i(\mathbf{r}) \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')). \tag{2.16}$$

In the same way, by using equation (B-23), the second term of equation (2.15) may be converted to the following,

$$-\mathbf{E}_i(\mathbf{r}) \cdot \nabla \times \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') = \nabla \cdot (\mathbf{E}_i(\mathbf{r}) \times \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')), \tag{2.17}$$

where $\mathbf{a} = \mathbf{E}_i(\mathbf{r})$, $\mathbf{b} = \nabla$ and $\bar{\mathbf{C}} = \nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ was inserted into Eq. (B-23). Inserting Eqs. (2.16) and (2.17) into Eq. (2.15) and integrating over the whole volume of each region in unmarked coordinates, gives

$$\begin{aligned} &\int_{V_i} \nabla \cdot ([\nabla \times \mathbf{E}_i(\mathbf{r})] \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') + \mathbf{E}_i(\mathbf{r}) \times [\nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')])\, \mathrm{d}^3\mathbf{r} \\ =& \int_{V_i} \left[i\omega\mu_i \mathbf{j}(\mathbf{r}) \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r})\delta(\mathbf{r}-\mathbf{r}') \right]\, \mathrm{d}^3\mathbf{r}. \end{aligned} \tag{2.18}$$

Applying the divergence theorem on the left-hand side gives

$$\begin{aligned}&\int_{V_i} \nabla\cdot\left([\nabla\times\mathbf{E}_i(\mathbf{r})]\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')+\mathbf{E}_i(\mathbf{r})\times[\nabla\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\right)\mathrm{d}^3\mathbf{r}\\&=\int_{\partial V_i}\hat{\mathbf{n}}_i\cdot\left([\nabla\times\mathbf{E}_i(\mathbf{r})]\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')+\mathbf{E}_i(\mathbf{r})\times[\nabla\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\right)\mathrm{d}^2\mathbf{r},\end{aligned}\tag{2.19}$$

where ∂V_i is the boundary surface of region i and $\hat{\mathbf{n}}_i$ is the outward directed normal vector of the boundary surface, ∂V_i. The first term on the right-hand side of Eq. (2.18) may be transformed by Eqs. (B-33) and (C-8) in the appendix, leading to

$$\begin{aligned}i\omega\mu_i\int_{V_i}\mathbf{j}(\mathbf{r})\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\,\mathrm{d}^3\mathbf{r}&=i\omega\mu_i\int_{V_i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')^T\cdot\mathbf{j}(\mathbf{r})\,\mathrm{d}^3\mathbf{r}\\&=i\omega\mu_i\int_{V_i}\bar{\mathbf{G}}_i(\mathbf{r}',\mathbf{r})\cdot\mathbf{j}(\mathbf{r})\,\mathrm{d}^3\mathbf{r}.\end{aligned}\tag{2.20}$$

Comparing the last term of this equation with equation (C-3) from the appendix makes it clear that the term is of the same type as the unknown variable of the original differential equation, which in this case is an electric field. The field must be a function of $\mathbf{r}'$ as the Green's function has a source point at $\mathbf{r}$, the source, $\mathbf{j}$, is a function of $\mathbf{r}$ and the integral is over $\mathbf{r}$. Furthermore, the source term, $i\omega\mu_i\mathbf{j}(\mathbf{r})$, on the right-hand side of Eq. (2.12) involves the *free* current density, meaning the current is "put there" by the one performing the experiment. One must therefore not confuse it with the surface current densities, $\mathbf{J}$ and $\mathbf{M}$, which are the imaginary sources of the scattered EM field. With this distinction in mind it is clear that the electric field represented by the term of Eq. (2.20) is the *incident* electric field, $\mathbf{E}_i^{\mathrm{inc}}(\mathbf{r}')$, of region i,

$$i\omega\mu_i\int_{V_i}\mathbf{j}(\mathbf{r})\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\,\mathrm{d}^3\mathbf{r}=\mathbf{E}_i^{\mathrm{inc}}(\mathbf{r}').\tag{2.21}$$

The second term on the right-hand side of Eq. (2.18) is easily identified as, $-\mathbf{E}_i(\mathbf{r}')$, when $\mathbf{r}'\in V_i$ and 0 otherwise, by using the properties of the delta function. Eq. (2.18) may therefore be written in the following way,

$$\begin{aligned}&\int_{\partial V_i}\hat{\mathbf{n}}_i\cdot\left([\nabla\times\mathbf{E}_i(\mathbf{r})]\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')+\mathbf{E}_i(\mathbf{r})\times[\nabla\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\right)\mathrm{d}^2\mathbf{r}\\&=\mathbf{E}_i^{\mathrm{inc}}(\mathbf{r}')-\begin{cases}\mathbf{E}_i(\mathbf{r}'), & \mathbf{r}'\in V_i\\ 0 & \text{otherwise}\end{cases}.\end{aligned}\tag{2.22}$$

The boundary surface, ∂V_2, is simply the surface, S, in Fig. 1.1. The boundary surface, ∂V_1, on the other hand, consists of both S and S_{inf}. In the limit where the surface, S_{inf}, approaches infinity the EM field approaches 0 at the surface, reducing the border ∂V_1 to S, but with opposite normal vector compared to ∂V_2, $\hat{\mathbf{n}}_1 = -\hat{\mathbf{n}}_2$.

Now, the first term of the integral in Eq. (2.22) may be converted by using Eq. (B-23) in the following way,

$$\hat{\mathbf{n}}_i \cdot ([\nabla \times \mathbf{E}_i(\mathbf{r})] \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') = \hat{\mathbf{n}}_i \times [\nabla \times \mathbf{E}_i(\mathbf{r})] \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}'). \tag{2.23}$$

Using Faraday's law for the frequency domain, Eq. (2.6), together with Eq. (1.3) connecting $\mathbf{B}_i(\mathbf{r})$ with $\mathbf{H}_i(\mathbf{r})$ and Eqs. (B-33) and (C-8) regarding the transpose of the dyadic Green's function gives

$$\begin{aligned}\hat{\mathbf{n}}_i \times [\nabla \times \mathbf{E}_i(\mathbf{r})] \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') &= \hat{\mathbf{n}}_i \times [i\omega \mathbf{B}_i(\mathbf{r})] \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \\ &= i\omega \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')^T \cdot [\hat{\mathbf{n}}_i \times \mathbf{B}_i(\mathbf{r})] \\ &= i\omega \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')^T \cdot [\hat{\mathbf{n}}_i \times \mu_i \mathbf{H}_i(\mathbf{r})] \\ &= i\omega\mu_i \bar{\mathbf{G}}_i(\mathbf{r}',\mathbf{r}) \cdot [\hat{\mathbf{n}}_i \times \mathbf{H}_i(\mathbf{r})].\end{aligned} \tag{2.24}$$

Likewise, by using Eqs. (B-23), (B-33) and (C-9) the second term of the integral in Eq. (2.22) may be converted in the following way,

$$\begin{aligned}\hat{\mathbf{n}}_i \cdot (\mathbf{E}_i(\mathbf{r}) \times [\nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]) &= [\hat{\mathbf{n}}_i \times \mathbf{E}_i(\mathbf{r})] \cdot [\nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \\ &= [\nabla \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]^T \cdot [\hat{\mathbf{n}}_i \times \mathbf{E}_i(\mathbf{r})] \\ &= -[\nabla \times \bar{\mathbf{G}}_i(\mathbf{r}',\mathbf{r})] \cdot [\hat{\mathbf{n}}_i \times \mathbf{E}_i(\mathbf{r})].\end{aligned} \tag{2.25}$$

Inserting these results into equation (2.22) and considering the case of the equation where $\mathbf{r}' \in V_i$, gives

$$\begin{aligned}\int_S i\omega\mu_i \bar{\mathbf{G}}_i(\mathbf{r}',\mathbf{r}) \cdot [\hat{\mathbf{n}}_i \times \mathbf{H}_i(\mathbf{r})] - [\nabla \times \bar{\mathbf{G}}_i(\mathbf{r}',\mathbf{r})] \cdot [\hat{\mathbf{n}}_i \times \mathbf{E}_i(\mathbf{r})]\, \mathrm{d}^2\mathbf{r} \\ = \mathbf{E}_i^{\text{inc}}(\mathbf{r}') - \mathbf{E}_i(\mathbf{r}').\end{aligned} \tag{2.26}$$

Defining the electric surface current density, $\mathbf{J}(\mathbf{r}) = \hat{\mathbf{n}}_2 \times \mathbf{H}_i(\mathbf{r})$ and the magnetic surface current density, $\mathbf{M}(\mathbf{r}) = \hat{\mathbf{n}}_1 \times \mathbf{E}_i(\mathbf{r})$ gives the following equation for the case of $i = 1$,

$$\begin{aligned}&\int_S i\omega\mu_1 \bar{\mathbf{G}}_1(\mathbf{r}',\mathbf{r}) \cdot [\hat{\mathbf{n}}_1 \times \mathbf{H}_1(\mathbf{r})] - [\nabla \times \bar{\mathbf{G}}_1(\mathbf{r}',\mathbf{r})] \cdot [\hat{\mathbf{n}}_1 \times \mathbf{E}_1(\mathbf{r})]\, \mathrm{d}^2\mathbf{r} \\ &= \int_S i\omega\mu_1 \bar{\mathbf{G}}_1(\mathbf{r}',\mathbf{r}) \cdot (-\mathbf{J}(\mathbf{r})) - [\nabla \times \bar{\mathbf{G}}_1(\mathbf{r}',\mathbf{r})] \cdot \mathbf{M}(\mathbf{r})\, \mathrm{d}^2\mathbf{r} \\ &= \mathbf{E}_1^{\text{inc}}(\mathbf{r}') - \mathbf{E}_1(\mathbf{r}').\end{aligned} \tag{2.27}$$

In the same way, the case of $i = 2$ gives

$$\begin{aligned}&\int_S i\omega\mu_2 \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r}) \cdot [\hat{\mathbf{n}}_2 \times \mathbf{H}_2(\mathbf{r})] - [\nabla \times \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r})] \cdot [\hat{\mathbf{n}}_2 \times \mathbf{E}_2(\mathbf{r})]\,\mathrm{d}^2\mathbf{r} \\ &= \int_S i\omega\mu_2 \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r}) \cdot \mathbf{J}(\mathbf{r}) - [\nabla \times \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r})] \cdot (-\mathbf{M}(\mathbf{r}))\,\mathrm{d}^2\mathbf{r} \\ &= \mathbf{E}_2^{\mathrm{inc}}(\mathbf{r}') - \mathbf{E}_2(\mathbf{r}').\end{aligned} \tag{2.28}$$

In the current scattering problem all incident EM waves originate from region 1, thus $\mathbf{E}_2^{\mathrm{inc}}(\mathbf{r}') = 0$. Therefore the case of $i = 2$ may be displayed in the following way,

$$\int_S -i\omega\mu_2 \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r}) \cdot \mathbf{J}(\mathbf{r}) - [\nabla \times \bar{\mathbf{G}}_2(\mathbf{r}',\mathbf{r})] \cdot \mathbf{M}(\mathbf{r})\,\mathrm{d}^2\mathbf{r} = \mathbf{E}_2(\mathbf{r}). \tag{2.29}$$

By swapping $\mathbf{r}$ with $\mathbf{r}'$ and using $-i = 1/i$, both cases may therefore be summed up in the following way,

$$\begin{aligned}&\int_S \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \times \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \cdot \mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}' \\ &= \begin{cases} \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}) - \mathbf{E}_i(\mathbf{r}), & i = 1\ ,\mathbf{r} \in V_1 \\ \mathbf{E}_i(\mathbf{r}), & i = 2\ ,\mathbf{r} \in V_2 \end{cases} .\end{aligned} \tag{2.30}$$

By rearranging the terms of this equation and using Eqs. (C-10) and (B-28) transforming the second term in the integral the following expression for the electric field, $\mathbf{E}_i(\mathbf{r})$, in any position in space is obtained,

$$\begin{aligned}\mathbf{E}_i(\mathbf{r}) = &\begin{Bmatrix} - \\ + \end{Bmatrix} \int_S \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}' \\ &+ \begin{cases} \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}), & i = 1\ ,\mathbf{r} \in V_1 \\ 0, & i = 2\ ,\mathbf{r} \in V_2 \end{cases} ,\end{aligned} \tag{2.31}$$

which corresponds to Eq. (35) in Ref. [3] without expanding the surface current densities, $\mathbf{J}(\mathbf{r}')$ and $\mathbf{M}(\mathbf{r}')$. An analogous derivation starting with a differential equation in the magnetic field only instead of Eq. (2.12) gives the following expression for the magnetic field, $\mathbf{H}_i(\mathbf{r})$,

$$\begin{aligned}\mathbf{H}_i(\mathbf{r}) = &\begin{Bmatrix} - \\ + \end{Bmatrix} \int_S \frac{\omega\epsilon_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{M}(\mathbf{r}') + [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{J}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}' \\ &+ \begin{cases} \mathbf{H}_1^{\mathrm{inc}}(\mathbf{r}), & i = 1\ ,\mathbf{r} \in V_1 \\ 0, & i = 2\ ,\mathbf{r} \in V_2 \end{cases} ,\end{aligned} \tag{2.32}$$

It is convenient to make a full stop at this point and contemplate the implications of Eqs. (2.31) and (2.32). The electric field at the position, **r**, is now expressed as an integral containing the Green's function and the surface current densities, **J** and **M**. If the observation point is located in region 1 the incoming electric field, $\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r})$, must also be added. The Green's function is well known and given in appendix C.2. So if the current densities are known the electric field may be calculated by solving the surface integral of Eq. (2.31) running over the surface of the scatterer. Correspondingly the magnetic field may be calculated by solving the surface integral of Eq. (2.32). This can be done accurately numerically if the integrand is "behaving properly". This subject is further discussed in Section 3.4. Further more Section 2.5 includes a neat trick for handling situations where the integrand does not behave well.

The surface current densities, **J** and **M**, are however generally unknown. Therefore the problem is now transformed into finding these surface current densities. When found they may be inserted into the integral of Eqs. (2.31) and (2.32) which in turn makes it possible to calculate the EM field. There are several different procedures available for finding the surface current densities. One of the possible ways is shown in Ref. [4]. In this paper the equation for the magnetic field is evaluated at two points slightly above and slightly below the boundary surface separating the two regions. These expressions are summed and the cross product of the normal vector corresponding to $\hat{\mathbf{n}}_2$ with the result of the added terms is calculated. This leads to a pair of coupled matrix equations which when solved gives the surface current densities at the points of a 2D grid. The following two sections shows a similar approach by deriving the electric field integral equation (EFIE) and the magnetic field integral equation (MFIE) and finding an approximation of the surface current densities by transforming the integral equations into two matrix equation.

2.3 Derivation of the EFIE and the MFIE

In the derivation of Section 2.2 only the case of Eq. (2.22) where the observation point was located inside the considered region was applied. Now the opposite case is considered, namely the observation point being located outside the considered region making the second term on the right-hand side of Eq. (2.22) zero. This transforms equation (2.30) to the following,

$$\int_S \frac{\omega\mu_i}{i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}') - [\nabla'\times\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\cdot\mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}' = \begin{cases}\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}), & i=1\ ,\mathbf{r}\in V_2\\ 0, & i=2\ ,\mathbf{r}\in V_1\end{cases}. \tag{2.33}$$

Using the same surface current densities for $i = 1$ and $i = 2$ in Eq. (2.33), forces the following equations to be satisfied,

$$\hat{\mathbf{n}}_2\times\mathbf{H}_1(\mathbf{r}) = \hat{\mathbf{n}}_2\times\mathbf{H}_2(\mathbf{r}) = -\hat{\mathbf{n}}_1\times\mathbf{H}_2(\mathbf{r}) = -\hat{\mathbf{n}}_1\times\mathbf{H}_1(\mathbf{r}), \tag{2.34}$$

$$\hat{\mathbf{n}}_1\times\mathbf{E}_1(\mathbf{r}) = \hat{\mathbf{n}}_1\times\mathbf{E}_2(\mathbf{r}) = -\hat{\mathbf{n}}_1\times\mathbf{E}_2(\mathbf{r}) = -\hat{\mathbf{n}}_1\times\mathbf{E}_1(\mathbf{r}). \tag{2.35}$$

Equations (2.34) and (2.35) correspond to the following boundary conditions for the EM field at the boundary surface, S,

$$\hat{\mathbf{n}}_i\times(\mathbf{H}_1(\mathbf{r})-\mathbf{H}_2(\mathbf{r})) = 0, \tag{2.36}$$

$$\hat{\mathbf{n}}_i\times(\mathbf{E}_1(\mathbf{r})-\mathbf{E}_2(\mathbf{r})) = 0. \tag{2.37}$$

The tangential component of the EM field is continuous across the boundary surface, S. This continuity allows taking the limit of equation (2.33) as $\mathbf{r}\to S$ from both regions and evaluating the tangential component of the equation. Transforming the second term of the integral by using Eqs. (C-10) and (B-28) gives therefore the following equation for the tangential component of Eq. (2.33) at the surface S,

$$\left(\int_S \frac{\omega\mu_i}{i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}') - [\nabla'\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\times\mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}'\right)_{\mathrm{tan}} = \begin{cases}(\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}))_{\mathrm{tan}}, & i=1\ ,\mathbf{r}\in S\\ 0, & i=2\ ,\mathbf{r}\in S\end{cases}. \tag{2.38}$$

This equation is called the EFIE. Correspondingly, for the magnetic field one may get the following equation (from Ref. [3]),

$$\left(\int_S \frac{\omega\epsilon_i}{i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{M}(\mathbf{r}') + [\nabla'\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\times\mathbf{J}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}'\right)_{\mathrm{tan}} = \begin{cases}(\mathbf{H}_1^{\mathrm{inc}}(\mathbf{r}))_{\mathrm{tan}}, & i=1\ ,\mathbf{r}\in S\\ 0, & i=2\ ,\mathbf{r}\in S\end{cases}, \tag{2.39}$$

which is called the MFIE. These sets of equations may be solved in terms of the surface current densities, **J** and **M**. One available method for finding an approximation of the surface current densities is called the Method of Weighted Residuals (MWR) and is presented in general form in Appendix D.

2.4 Using the MWR to Approximate the Surface Current Densities

As mentioned in the end of Section 2.2 the surface current densities, $\mathbf{J}$ and $\mathbf{M}$, in Eqs. (2.31) and (2.32) are generally unknown and thus need to be found before calculating the integral. The previous section derived a set of integral equations in these unknown current densities. This section presents how the MWR can be used for approximating the current densities based on these derived integral equations. This method transforms the problem to a matrix equation which may be solved easily by linear algebra libraries. The transformation for a general linear operator equation is described in detail in Appendix D.

Let us investigate this transformation when the EFIE, Eq. (2.38) is the equation we want to solve for the unknown functions $\mathbf{J}$ and $\mathbf{M}$. The first step of the MWR is to approximate the boundary surface, S, by a discrete mesh, S^{d}, of flat geometrical figures. The most common shapes used in this respect are triangles or rectangles. Also possible is using a combination of different shapes. The next step is to approximate the unknown functions as linear combinations of N linearly independent basis functions as shown in Eq. (D-10),

$$\mathbf{J}(\mathbf{r}) \approx \sum_{n=1}^{N} \alpha_n \mathbf{f}_n(\mathbf{r}), \tag{2.40}$$

$$\mathbf{M}(\mathbf{r}) \approx \sum_{n=1}^{N} \beta_n \mathbf{f}_n(\mathbf{r}), \tag{2.41}$$

where α and β are the constant expansion coefficients and $\mathbf{f}_n(\mathbf{r})$ denote the n'th basis function. These basis functions may be chosen in different ways, but the most common are using so called *rooftop functions*. These are functions having non-zero value on two discretization elements sharing one common edge, see Figs. 2.1 and 2.2. Thus the number of basis functions, N, equals the number of edges, N_{ed}, in the discretization.

Fig. 2.1: Triangular Rooftop Function:

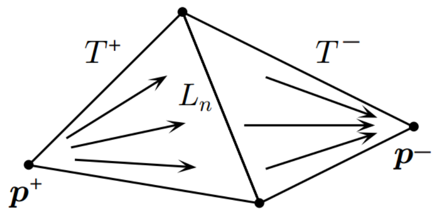

Figure 2.1: Illustration of an RWG rooftop basis function (From Ref. [5]).

Fig. 2.2: Rectangular Rooftop Function:

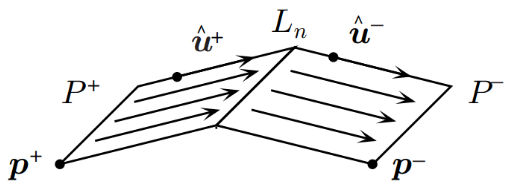

Figure 2.2: Illustration of a rectangular rooftop basis function (From Ref. [5]).

The basis functions shown in Figs. 2.1 and 2.2 are defined in the following way (from Ref. [5]),

$$\mathbf{f}_n^t(\mathbf{r}) = \begin{cases} \pm\frac{L_n}{2A^{\pm}}(\mathbf{r}-\mathbf{p}^{\pm}), & \mathbf{r}\in T^{\pm} \\ 0, & otherwise \end{cases}, \tag{2.42}$$

$$\mathbf{f}_n^r(\mathbf{r}) = \begin{cases} \pm\frac{L_n}{2A^{\pm}}(\mathbf{r}-\mathbf{p}^{\pm})\cdot\hat{\mathbf{u}}^{\pm}, & \mathbf{r}\in P^{\pm} \\ 0, & otherwise \end{cases}, \tag{2.43}$$

where L_n denote the length of the common edge, T^+, T^-, P^+ and P^- denote the two triangular elements and the two rectangular elements respectively. Furthermore, A^x, where $x \in \{+,-\}$, denote the area of the respective element. The remaining symbols may be understood by inspecting Figs. 2.1 and 2.2.

The next step consists of choosing the weighting functions. There are several possibilities, each choice giving rise to a different sub type of the MWR. The Collocation method, the Least square method and Galerkin's method are some examples. Now, the final result of the MWR is given in Eqs. (D-18)-(D-20) in Appendix D. By comparing Eq. (2.38) with Eq. (D-1) it is clear that the function $l(\mathbf{x})$ and the operator L corresponds to the following,

$$l(\mathbf{x}) \rightarrow \mathbf{l}(\mathbf{r}) = \begin{cases} (\mathbf{E}_1^{\text{inc}}(\mathbf{r}))_{\tan}, & i=1,\ \mathbf{r} \in S \\ 0, & i=2,\ \mathbf{r} \in S \end{cases} \tag{2.44}$$

$$\begin{aligned} L\phi(\mathbf{x}) &\rightarrow L_1\boldsymbol{\phi}_1(\mathbf{r}) + L_2\boldsymbol{\phi}_2(\mathbf{r}) \\ &= \left(\int_S \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{M}(\mathbf{r}')\, \mathrm{d}^2\mathbf{r}' \right)_{\tan}, \end{aligned} \tag{2.45}$$

where $\boldsymbol{\phi}_1(\mathbf{r}) = \mathbf{J}(\mathbf{r})$ and $\boldsymbol{\phi}_2(\mathbf{r}) = \mathbf{M}(\mathbf{r})$. By inserting the current density expansions of Eqs. (2.40) and (2.41) and letting the integral run over the discretization mesh, S^{d}, one obtains,

$$\begin{aligned} L_1\boldsymbol{\phi}_1(\mathbf{r}) + L_2\boldsymbol{\phi}_2(\mathbf{r}) &\approx \left(\int_{S^{\text{d}}} \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \sum_{n=1}^{N} \alpha_n \mathbf{f}_n(\mathbf{r}') \right. \\ &\qquad \left. - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \sum_{n=1}^{N} \beta_n \mathbf{f}_n(\mathbf{r}')\, \mathrm{d}^2\mathbf{r}' \right)_{\tan} \\ &= \sum_{n=1}^{N} \left(\int_{S^{\text{d}}} \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \alpha_n \mathbf{f}_n(\mathbf{r}') \mathrm{d}^2\mathbf{r}', \right. \\ &\qquad \left. - \int_{S^{\text{d}}} [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \beta_n \mathbf{f}_n(\mathbf{r}')\, \mathrm{d}^2\mathbf{r}', \right) \end{aligned} \tag{2.46}$$

Note that both sides of Eq. (D-1) now represent complex vector functions. Therefore, the residual, R, in Eq. (D-12) represents a complex vector function, $\mathbf{R}$. If the weighting functions, w_m, also are vector functions Eq. (D-13) is rewritten as,

$$\int_Q \mathbf{w}_m(\mathbf{x}) \cdot \mathbf{R}(\mathbf{x})\, \mathrm{d}\mathbf{x} = 0 = [\mathbf{w}_m, \mathbf{R}], \quad m \in [1, N], \tag{2.47}$$

which defines the inner product of two vector functions on the domain, Q, which here represents where we want to find the solution of Eq. (2.38), namely the discretized boundary surface, S^{d}. It is now possible to identify the matrix and vector elements of the transformed problem,

$$A\mathbf{v} = \mathbf{b}. \tag{2.48}$$

The unknown expansion coefficients, α_n and β_n, make up the vector, $\mathbf{v}$, in the following way,

$$\mathbf{v} = [\alpha_1, \alpha_2, \dots, \alpha_N, \beta_1, \beta_2, \dots, \beta_N]. \tag{2.49}$$

This divides the matrix elements, A_{mn}, in two. The first N columns are connected to the terms in Eq. (2.46) which include α_n and the last N columns are connected to the terms of Eq. (2.46) which include β_n. There are also a total of $2N$ equations in the linear system, N equations for each of the two regions. Let the first N equations represent the equations for Region 1. We may now split the matrix, A, into four sub matrices, D^1, D^2, K^1 and K^2, where the superscript denotes the region, and the letter indicates whether the element are among the first or the last N columns,

$$A = \begin{bmatrix} D^1 & -K^1 \\ D^2 & -K^2 \end{bmatrix}. \tag{2.50}$$

The matrix elements are given by Eq. (D-19),

$$D^i_{mn} = [\mathbf{w}_m, L^i_1 \mathbf{f}_n] = \frac{\omega\mu_i}{i} \int_{S^\mathrm{d}} \mathrm{d}^2\mathbf{r}\, \mathbf{w}_m \cdot \int_{S^\mathrm{d}_n} \mathrm{d}^2\mathbf{r}' \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{f}_n(\mathbf{r}'), \tag{2.51}$$

$$K^i_{mn} = [\mathbf{w}_m, L^i_2 \mathbf{f}_n] = \int_{S^\mathrm{d}} \mathrm{d}^2\mathbf{r}\, \mathbf{w}_m \cdot \int_{S^\mathrm{d}_n} \mathrm{d}^2\mathbf{r}' [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}'). \tag{2.52}$$

Note that the inner integrals are over the surface of the n'th element pair, for example in case of a triangular mesh we have, $S^\mathrm{d}_n = T^+_n \cup T^-_n$. This is sufficient because the basis function $\mathbf{f}_n$ is zero on all other elements. If we for instance choose Galerkin's Method of Weighted Residuals, which is briefly described in Appendix D.3, we get the following weighting functions, $\mathbf{w}_m$,

$$\mathbf{w}_m = \mathbf{f}_m(\mathbf{r}), \quad m \in \{1, 2, .., N_\mathrm{ed}\} \quad \text{(Galerkin's Method)}. \tag{2.53}$$

By inserting these weighting functions into Eqs.(2.51) and (2.52) the outer integral is reduced to run over the m'th pair of elements, S^d_m,

$$D^i_{mn} = [\mathbf{f}_m, L^i_1 \mathbf{f}_n] = \frac{\omega\mu_i}{i} \int_{S^\mathrm{d}_m} \mathrm{d}^2\mathbf{r}\, \mathbf{f}_m(\mathbf{r}) \cdot \int_{S^\mathrm{d}_n} \mathrm{d}^2\mathbf{r}' \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{f}_n(\mathbf{r}'), \tag{2.54}$$

$$K^i_{mn} = [\mathbf{f}_m, L^i_2 \mathbf{f}_n] = \int_{S^\mathrm{d}_m} \mathrm{d}^2\mathbf{r}\, \mathbf{f}_m(\mathbf{r}) \cdot \int_{S^\mathrm{d}_n} \mathrm{d}^2\mathbf{r}' [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}'), \tag{2.55}$$

which corresponds to Eqs. (20) and (21) in Ref. [3]. The elements of the vector, $\mathbf{b}$, are given by Eq. (D-20), where the known function, l, is given by

Eq. (2.44). This gives the following vector elements, b_m,

$$b_m = [\mathbf{l}, \mathbf{w}_m] = \begin{cases} \int_{S^\mathrm{d}} \mathrm{d}^2\mathbf{r}\, \mathbf{w}_m(\mathbf{r}) \cdot (\mathbf{E}_1^\mathrm{inc}(\mathbf{r}))_\mathrm{tan}, & m \in \{1, 2, .., N_\mathrm{ed}\} \\ 0, & m \in \{N_\mathrm{ed}+1, N_\mathrm{ed}+2, .., 2N_\mathrm{ed}\} \end{cases} . \tag{2.56}$$

If again Galerkin's Method is applied the following expression is obtained,

$$b_m = [\mathbf{l}, \mathbf{f}_m] = \begin{cases} \int_{S_m^\mathrm{d}} \mathrm{d}^2\mathbf{r}\, \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{E}_1^\mathrm{inc}(\mathbf{r}), & m \in \{1, 2, .., N_\mathrm{ed}\} \\ 0, & m \in \{N_\mathrm{ed}+1, N_\mathrm{ed}+2, .., 2N_\mathrm{ed}\} \end{cases} , \tag{2.57}$$

where the integral is reduced to cover only the surface of the m'th element pair for the same reason as above. Note that the tangential component is automatically the only contribution to the integral due to the dot product operation and the fact that the basis functions, $\mathbf{f}_m(\mathbf{r})$, are parallel to the discretized boundary surface, S^d. Equation (2.57) corresponds to Eq. (23) in Ref. [3].

Starting out with the MFIE instead gives by an analogous derivation a similar linear system presented in Ref. [3]. Both the EFIE and the MFIE leads to a set of $2N$ equations. Therefore either one of these equations could in principle be chosen for approximating the total of $2N$ unknown variables, $\{\alpha_n\}$ and $\{\beta_n\}$. This is however not always the case. At the resonant frequencies of the cavity formed by the boundary surface, S, the solutions are not unique, see Ref. [5]. This problem is approached by solving a linear combination of the EFIE and the MFIE, resulting in one unique solution. This is however not an issue for the problems studied in this thesis where the unique solution may be found by solving either linear system.

Note also that the Green's function and its gradient are singular for $\mathbf{r} = \mathbf{r}'$. This makes the double integrals of this section for overlapping integration domains, S_m^d and S_n^d, problematic numerically. This may be solved in the same way as described in Section 2.5, splitting the integrands into analytical terms and smooth numerically solvable terms. Another way which is applied in the implementation is setting all Gaussian quadrature terms where $\mathbf{r} = \mathbf{r}'$ equal to zero, thereby actually calculating the Cauchy Principal Value. How the integrals are converted into sums using the Gaussian quadrature method is presented in Sections 3.4 and 3.5

2.5 Observation Point Close to the Scatterer Surface

From Eqs. (3.4)-(3.7) in Section 3.2 it is clear that each element of the Green's function behaves qualitatively as one over the distance between the observation point and the reference point cubed, $\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}') \sim \frac{1}{R^3}$. The integral is a surface integral, and therefore blows up when the observation point, $\mathbf{r}$, approaches the surface of the scatterer. This problem is handled in a neat way in Ref. [5]. The basic idea is to integrate the singular terms of the Taylor expansion of the Green's function analytically thereby leaving a smoothened, numerically integrable integrand for arbitrary small distances, R. In the following equation K denotes the integrand blowing up for small R. Furthermore, A, denotes the terms separated out which may be integrated analytically,

$$\begin{aligned}\int_D K(\mathbf{r}, \mathbf{r}')\, \mathrm{d}^2\mathbf{r}' &= \int_D K(\mathbf{r}, \mathbf{r}') - A(\mathbf{r}, \mathbf{r}') + A(\mathbf{r}, \mathbf{r}')\, \mathrm{d}^2\mathbf{r}' \\ &= \int_D K(\mathbf{r}, \mathbf{r}') - A(\mathbf{r}, \mathbf{r}')\, \mathrm{d}^2\mathbf{r}' + \int_D A(\mathbf{r}, \mathbf{r}')\, \mathrm{d}^2\mathbf{r}'.\end{aligned} \tag{2.58}$$

The first integral of the last line of Eq. (2.58) is then possible to solve accurately numerically while the second integral may be solved analytically.

Chapter 3

Numerical Implementation

In order to solve an EM scattering problem with the method derived in Section 2 the surface current densities **J** and **M** must be calculated and inserted into equation (2.31) and/or (2.32). In most cases the integral equations of Section 2.3 need to be solved numerically, but in some special cases the surface current densities may be approximated without having to solve the integral equations. This section deals specifically with two slightly different case studies. The first case study belongs to the category of problems where the surface current densities may be approximated in a simple way. The second case study is identical to the first except for a crucial dimensional assumption which is left out. This makes the problem more general and the surface current densities must in this case be estimated numerically. A numerical solution of the first case study was first implemented and tested before moving on to the second. These implementations are presented in this section along with general considerations which are important regarding the numerical implementation.

3.1 The Case Studies

The case studies are both examples of the EM scattering problem shown in Fig. 1.1. They are only slightly different from each other, but the difference is crucial for the numerical implementation of the solution.

Case study one was chosen to have the following specifications:

1. Region 1 contains vacuum, $\epsilon_1 = \epsilon_0$ and $\mu_1 = \mu_0$.

2. Region 2, containing the scatterer, is a perfect conductor having spherical shape with radius, ρ.

3. The incoming EM wave is a plane wave with wavelength, $\lambda \ll \rho$.

Case study two was chosen to have the following specifications:

1. Region 1 contains vacuum, $\epsilon_1 = \epsilon_0$ and $\mu_1 = \mu_0$.
2. Region 2, containing the scatterer, is a perfect conductor having spherical shape with radius, ρ.
3. The incoming EM wave is a plane wave with wavelength, λ.

The constants, ϵ_0 and μ_0, are the permittivity and permeability of vacuum, respectively. In both cases the chosen task was to calculate the resulting total electric field amplitude, $|\mathbf{E}_i(\mathbf{r})|$, for a set of observation points, $\mathbf{r} \in O$.

The first problem is more precisely identified as a special case of the second problem, with the restriction of the wavelength being much smaller than the radius of the spherical scatterer. This allows a simple "short cut", which is described below, for estimating the surface current densities. Therefore the first problem serves excellently as a way of testing the implementation of the discretization and numerical integration parts of the program before implementing the solution for the second problem involving setting up and solving a linear system. The reason for separating them as two different problems is to make it clear that they demand separate solution methods. However, the main structure of the program implementations is common. This is presented in more detail in Section 3.3.

As mentioned in the introduction the EM scattering problem involving a spherical scatterer is useful for testing the SIE method for several reasons. First, the solution must have certain symmetries which may give a quick indication of the correctness of the results. The problem even has an analytical solution which could further help in the assessment of the results. The Rayleigh theory was applied in this respect. Furthermore, there are easily available discretization libraries online for creating the discretization of a sphere.

The above specifications have some immediate implications. First, the incoming EM wave is not attenuated during its travel through region 1. Furthermore, the magnetic surface current density, $\mathbf{M}$, is zero because of the boundary condition given by Eq. (2.35) and the fact that the electric field inside a perfect conductor is zero. Therefore, we already know the solution in region 2.

Thus far the implications are the same for both problems. Specification 3 however introduces an important difference. In the case of problem two there are no restrictions in the wavelength of the incoming plane wave. The solution therefore must include an implementation of the MWR presented in Section 2.4. For case study one however, specification 3 implies that for the incoming EM wave the boundary surface "looks" approximately flat because the wavelength is much smaller than the scatterer radius. One may when the wavelength of the incoming EM wave is sufficiently small use Kirchhoff's Approximation (KA), also known as the tangent-plane method. This is the above mentioned "short cut" for estimating the surface current density, **J**, of case study one. The approximation is given by the following equations,

$$\mathbf{E}(\mathbf{r}) \approx 2\mathbf{E}^{\mathrm{inc}}(\mathbf{r})\ , \qquad \mathbf{r} \in S, \tag{3.1}$$

$$\mathbf{H}(\mathbf{r}) \approx 2\mathbf{H}^{\mathrm{inc}}(\mathbf{r})\ , \qquad \mathbf{r} \in S, \tag{3.2}$$

from Eq. 3 in Ref. [6]. The total EM field at the surface of the scatterer is thus approximated as twice the size of the incoming field. In our case only Eq. (3.2) is regarded valid because of the perfectly conducting scatterer and is used directly to find $\mathbf{J}(\mathbf{r})$ by its definition,

$$\begin{aligned}\mathbf{J}(\mathbf{r}) &= \hat{\mathbf{n}}_2 \times \mathbf{H}_i(\mathbf{r}),\\ &\approx \hat{\mathbf{n}}_2 \times 2\mathbf{H}^{\mathrm{inc}}(\mathbf{r})\ , \qquad \mathbf{r} \in S^{\mathrm{d}}.\end{aligned} \tag{3.3}$$

3.2 The Green's Function

Before being able to use the Green's function we must calculate the double derivate represented by the two ∇ operators in Eq. (C-7). This gives the following expression for the elements of the Green's tensor,

$$[\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]_{lm} = G_i(\mathbf{r},\mathbf{r}')[\delta_{lm} + Z(l,m,R,k_i)], \tag{3.4}$$

where l and m are the index of the row and column respectively of the element considered in the matrix representation of the tensor (see appendix B.1 for details). The distance, $|\mathbf{r}-\mathbf{r}'|$, is denoted by R and $G_i(\mathbf{r},\mathbf{r}')$ is the scalar Green's function given by,

$$G_i(\mathbf{r},\mathbf{r}') = \frac{\exp(ik_iR)}{4\pi R}. \tag{3.5}$$

Moreover, the factor, $Z(l,m,\mathbf{r},\mathbf{r}',k_i)$, is given by the following expression,

$$Z(l,m,\mathbf{r},\mathbf{r}',k_i) = \begin{cases} \dfrac{R}{k_i^2}(x_l - x_l')(x_m - x_m')\alpha(R,k_i), & l \neq m \\ \dfrac{R}{k_i^2}\left[(x_l - x_l')^2\alpha(R,k_i) + \dfrac{1}{R^2}(ik - \dfrac{1}{R})\right], & l = m \end{cases}, \tag{3.6}$$

where x_l and x'_l is the l'th component of the observation point vector, $\mathbf{r}$, and the source point vector, $\mathbf{r}'$, respectively. Furthermore, the factor, $\alpha(R, k_i)$, is given by,

$$\alpha(R, k_i) = \frac{3}{R^5} - \frac{3ik_i}{R^4} - \frac{k_i^2}{R^3}. \tag{3.7}$$

3.3 Numerical Solution of the Case Studies

This section presents the structure of the program implemented to solve the case studies of Section 3.1 and gives, together with Sections 3.4-3.6, a detailed presentation of how the theory of Section 2 is converted into programmable algorithms.

The main structure of the implemented numerical solution is shown in Fig. 3.1. The first module, *Discretization*, performs the actual discretization of the scattering structure and converts the data to a custom made data structure. The choices made in this part of the program are the mesh fineness of the discrete structure and which discretization algorithm to use. More details on this module is presented in Section 3.6.1.

Thus far there are no differences in the two methods (except possibly the chosen mesh fineness). But at this point the program flow is separated in two distinct routes dependent on whether or not the KA is valid. If the KA is valid the rightmost route of Fig. 3.1 is executed. This approximates the current density, $\mathbf{J}$, very quickly by the KA formula of Eq. (3.3). If however the KA is *not* valid the program must follow the more time consuming leftmost route of Fig. 3.1. This involves estimating the expansion coefficients of Eq. (2.40) by solving the linear system given by Eq. (2.48).

The remaining steps of the program are equal for the two different problems. When the surface current density, $\mathbf{J}$, is estimated it may be inserted into the integral of Eq. (2.31), for the case of $i = 1$, which then must be calculated numerically. The electric field amplitude is then calculated by taking the absolute value of the resulting complex vector $\mathbf{E}_1(\mathbf{r})$ in all observation points, $\mathbf{r}$. All the numerical calculation up to this point is performed by the program written in the FORTRAN language. The program outputs the electric field amplitude to a file which is imported to Matlab and used for making a contour plot.

Fig. 3.1: Flow diagram of the numerical implementation:

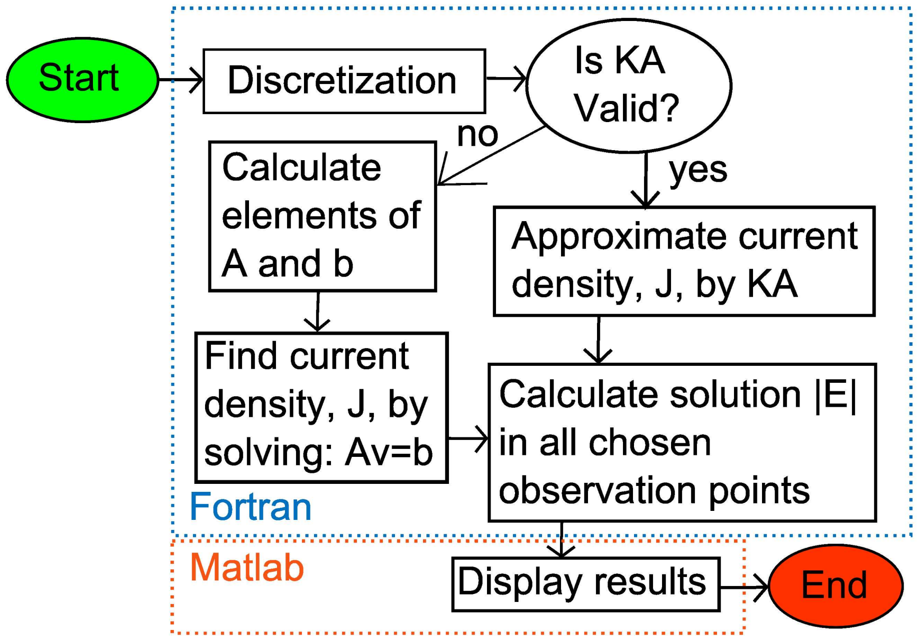
Start
Discretization
Is KA Valid?
no
yes
Calculate elements of A and b
Approximate current density, J, by KA
Find current density, J, by solving: Av=b
Calculate solution |E| in all chosen observation points
Fortran
Matlab
Display results
End

Figure 3.1: Flow diagram showing the two possible program routes dependent on whether or not the KA is valid.

3.4 Numerical Solution of Case Study One

When the surface current densities, **J** and **M** are identified they are inserted into Eq. (2.31) and/or (2.32). The elements of the Green's function given by Eqs. (3.4)-(3.7) are also inserted into the integral equation(s) which are then solved numerically. In this thesis Eq. (2.31) is applied for the implementation. There are many methods for performing the numerical integration. Irrespective of the method the first step is to chop up the integral of Eq.(2.31) (alternatively the integral of Eq. (2.32)) into N pieces in the following way,

$$\begin{aligned} I(\mathbf{r}) &= \int_S \frac{\omega\mu_i}{i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}') - [\nabla'\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\times\mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}' \\ &= \sum_{n=1}^{N}\int_{S_n} \frac{\omega\mu_i}{i}\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}') - [\nabla'\cdot\bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\times\mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}', \end{aligned} \tag{3.8}$$

where S_n is the area of the curved surface of element n on the boundary surface S. So far the expression is exact. No approximations have been done. The integral is only split into N distinct parts. This is done by picking out a set of points on the surface, S, and connecting them by lines on the surface giving rise to $N_{\mathrm{el}} = N$ elements of different geometrical shapes with arbitrary numbers of corners and edges. It is important however (for this particular integration method) to make sure all corners of a given element is located on a common plane. If all elements have only three corners this is trivially satisfied. The point is that the first approximation is to integrate over the surface consisting of the plane connecting the corners. This surface is bounded by the edges of the element. This new element surface is denoted by T_n. The expression of Eq. (3.8) is therefore transformed to,

$$I(\mathbf{r}) \approx \sum_n \int_{T_n} \frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{M}(\mathbf{r}')\,\mathrm{d}^2\mathbf{r}'. \tag{3.9}$$

The elements are assumed to be so small that the integrand is changing slowly on the surface of each element. Gaussian quadrature integration is a method of approximating a definite integral by a weighted sum of function values evaluated inside the integration domain. The method is exact for polynomial integrands of up to a limiting degree dependent on the number of evaluation points. Appendix E introduces Gaussian quadrature integration and presents the formulas chosen for the implementation.

The basic rule is the better estimate of the shape of the integrand function wanted, the more evaluation points and computational time is needed. The accuracy must be weighed against the computational time at hand. Also the complexity of the implementation is a factor. Furthermore, the accuracy can also be increased by increasing the number of elements in the discretization. Therefore, a simple solution, which also may give high accuracy, is to approximate the integral by the simplest shape possible, namely a constant. This require only one evaluation of the integrand per element. The integral is when applying this approximation converted to the following sum,

$$\begin{aligned} I(\mathbf{r}) &\approx \sum_n \left[\frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{M}(\mathbf{r}')\right]\Bigg|_{\mathbf{r}'=\mathbf{r}_n} \int_{T_n} \mathrm{d}^2\mathbf{r} \\ &= \sum_n \left[\frac{\omega\mu_i}{i} \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{J}(\mathbf{r}') - [\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \times \mathbf{M}(\mathbf{r}')\right]\Bigg|_{\mathbf{r}'=\mathbf{r}_n} A(T_n), \end{aligned} \tag{3.10}$$

where $A(T_n)$ is the area of element n and $\mathbf{r}_n$ is an arbitrary point inside element n, often chosen to be the centroid of the geometrical figure.

Generally the derivation $\nabla' \cdot \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ needs to be performed analytically before the implementation. However, in the case studies, $\mathbf{M} = \mathbf{0}$, so the following equation is ready to be implemented,

$$I_{\mathrm{S}}(\mathbf{r}) \approx \frac{\omega\mu_i}{i} \sum_n \bar{\mathbf{G}}_i(\mathbf{r},\mathbf{r}_n) \cdot \mathbf{J}(\mathbf{r}_n) A(T_n), \tag{3.11}$$

where I_{S} denote the special version of the integral where $\mathbf{M} = \mathbf{0}$, valid in both case studies. The Green's function, $\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}_n)$, is given by Eqs. (3.4)-(3.7).

Another slightly more complex alternative is choosing the 3-point Gaussian quadrature integration formula from Ref. [7] which also is presented in Appendix E. This formula introduces a new sum over the evaluation points resulting in the following expression for the integral, $I_{\mathrm{S}}(\mathbf{r})$,

$$I_{\mathrm{S}}(\mathbf{r}) \approx \frac{\omega\mu_i}{i} \sum_n A(T_n) \frac{1}{3} \sum_{p=1}^{3} \bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}_{n,p}) \cdot \mathbf{J}(\mathbf{r}_{n,p}), \tag{3.12}$$

where $\mathbf{r}_{n,p}$ denote the p'th evaluation point of the n'th discretization element. The advantage of this method compared to the 1-point formula is the reduced number of elements required to get a good integral approximation. This is especially useful when the memory demand increases rapidly with the number of elements. This is however more relevant in the second case study and will be further discussed in Section 3.5.

By taking the vector cross product of the normal vector, $\hat{\mathbf{n}}_2$, with Eq. (3.2) from the left one obtains the following surface current density, $\mathbf{J}$,

$$\mathbf{J}(\mathbf{r}_{n,p}) = 2\mathbf{J}^{\mathrm{inc}}(\mathbf{r}_{n,p}), \qquad \mathbf{r}_{n,p} \in \{\mathbf{c}_{n,p}\}, \tag{3.13}$$

where $\{\mathbf{c}_{n,p}\}$ represents the set of all evaluation points of the discretization. Now, to sum up, the solution of the first case study for the electric field, $\mathbf{E}_1$, in region 1 is given by the following expression,

$$\mathbf{E}_1(\mathbf{r}_j) = \mathbf{E}^{\mathrm{inc}}(\mathbf{r}_j) - I_{\mathrm{S}}(\mathbf{r}_j), \ \mathbf{r}_j \in V_1 \tag{3.14}$$

where $\mathbf{r}_j$ is an observation point in region 1, $\mathbf{E}^{\mathrm{inc}}(\mathbf{r}_j)$ denote the known incoming electric field at that point and the integral, I_{S}, is given by Eqs. (3.11) or (3.12), and (3.13).

3.5 Numerical Solution of Case Study Two

The wavelength of the incoming plane wave is now without restrictions. If the wavelength, λ, is small compared with the radius, ρ, of the sphere the KA is applicable and the solution given by Eq. (3.14) may be implemented directly. However, this is generally not the case and the surface current densities, $\mathbf{J}$ and $\mathbf{M}$, must be estimated. This is done by calculating the matrix elements of A and $\mathbf{b}$ in Eq. (2.48) and solving for the vector, $\mathbf{v}$.

The discretization elements are for simplicity chosen as triangles and the Rao-Wilton-Glisson (RWG) functions given by Eq. (2.42) are chosen as the set of basis functions. Note that in the case study the matrices, K^i, of Eq. (2.50) are superfluous because $\mathbf{M} = \mathbf{0}$ and thus all β_n of Eq. (2.49) are known to be zero. Thus, the system has got only N unknown variables, $\alpha_1, \alpha_2, .., \alpha_N$, where $N = N_{\mathrm{ed}}$ is the number of triangle edges in the discretization. Therefore only N equations are needed for the linear system. The matrix, A, may as a consequence be set equal to the matrix, D^1, given by Eq. (2.54) and the right-hand side of the linear system is given by the first N_{ed} elements of Eq. (2.57). This gives the following linear system,

$$D^1\mathbf{v} = \mathbf{b}, \tag{3.15}$$

consisting of N_{ed} equations, one for each unknown variable, α_n. Thus, in order to estimate the current density, the elements D^1_{mn} and b_m must be calculated by respectively the double integral of Eq. (2.54) and the single integral of Eq. (2.57). These integrals may be calculated numerically by either of the two methods applied in the previous section, namely the 1-point and the 3-point Gaussian quadrature formula. The former method approximates the elements of the matrix, D^1, in the following way,

$$\begin{aligned} D^1_{mn} &= \frac{\omega\mu_1}{i}\int_{S^{\mathrm{d}}_m} \mathrm{d}^2\mathbf{r}\,\mathbf{f}_m(\mathbf{r})\cdot\int_{S^{\mathrm{d}}_n}\mathrm{d}^2\mathbf{r}'\bar{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')\cdot\mathbf{f}_n(\mathbf{r}') \\ &\approx \frac{\omega\mu_1}{i}\sum_{t=1}^{2} A(T^t_m)\mathbf{f}_m(\mathbf{r}^t_m)\cdot\sum_{l=1}^{2} A(T^l_n)\bar{\mathbf{G}}_1(\mathbf{r}^t_m,\mathbf{r}^l_n)\cdot\mathbf{f}_n(\mathbf{r}^l_n), \end{aligned} \tag{3.16}$$

where $A(x)$ denote the area of the triangle x. For each matrix element, D^1_{mn}, one must evaluate two basis functions, each function must be evaluated in two different points, $\mathbf{r}^j_i$, which represents the centroid of the j'th element of the i'th basis function. Figure 3.2 shows a section of the discretization including a pair of non-overlapping basis functions, $\mathbf{f}_m$ and $\mathbf{f}_n$ together with all the the four evaluation points necessary to calculate the matrix element, D^1_{mn}.

Instead applying the 3-point formula results in the following approximation,

$$\begin{aligned} D^1_{mn} &= \frac{\omega\mu_1}{i}\int_{S^{\mathrm{d}}_m} \mathrm{d}^2\mathbf{r}\,\mathbf{f}_m(\mathbf{r})\cdot\int_{S^{\mathrm{d}}_n}\mathrm{d}^2\mathbf{r}'\bar{\mathbf{G}}_1(\mathbf{r},\mathbf{r}')\cdot\mathbf{f}_n(\mathbf{r}') \\ &\approx \frac{\omega\mu_1}{i}\sum_{t=1}^{2} A(T^t_m)\frac{1}{3}\sum_{p=1}^{3}\mathbf{f}_m(\mathbf{r}^t_{m,p}) \\ &\cdot\sum_{l=1}^{2} A(T^l_n)\frac{1}{3}\sum_{q=1}^{3}\bar{\mathbf{G}}_1(\mathbf{r}^t_{m,p},\mathbf{r}^l_{n,q})\cdot\mathbf{f}_n(\mathbf{r}^l_{n,q}), \end{aligned} \tag{3.17}$$

Fig. 3.2: A section of the discretization:

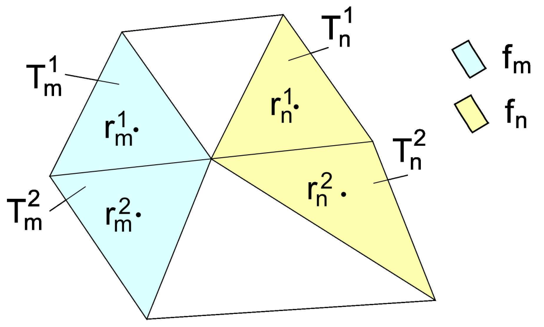

Figure 3.2: Illustration of the terms of Eq. (3.16). The basis function, $\mathbf{f}_m$, is nonzero on the triangles T_m^1 and T_m^2, correspondingly the basis function, $\mathbf{f}_n$ is nonzero on the triangles T_n^1 and T_n^2. The points denoted by, $\mathbf{r}_j^i$, represent respectively the centroid of triangle i of basis function j.

where $\mathbf{r}_{m,p}^t$ denotes the p'th evaluation point of element t in the m'th discretization pair, where the m'th basis function is non-zero. The factor, $\frac{1}{3}$, comes from the weighting constants in the Gaussian quadrature formula, which for all three terms in this formula equals $\frac{1}{3}$. The situation is only different from Fig. 3.2 in that each centroid point is substituted by three evaluation points.

This 3-point formula is a very useful alternative to the 1-point formula for this particular double integral. The reason is mainly connected to the size of the resulting matrix, D^1. The number of matrix elements increases with the number of discretization edges, N_{ed}, as N_{ed}^2. Even though the time to calculate each matrix element increases, the number of required integrals to calculate and most importantly the memory need decrease significantly. This is important because the single most limiting factor in this case study regarding the tools for solving the problem is probably the large amount of memory needed, especially when the wavelength becomes small compared to the radius. Also noting the fact that the algorithm solving the linear system needs a number of operations in the order of N_{ed}^3 underlines the fact that there is a lot to gain by using for instance the 3-point formula instead of the simpler 1-point formula.

The elements on the right-hand side of Eq. (3.15) may be calculated using the same integration methods as above. The 1-point formula gives,

$$\begin{aligned} b_m &= \int_{S_m^{\mathrm{d}}} \mathrm{d}^2\mathbf{r}\, \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}) \\ &\approx \sum_{t=1}^{2} A(T_m^t)\mathbf{f}_m(\mathbf{r}_m^t) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}_m^t), \end{aligned} \tag{3.18}$$

where each element, b_m, demands integrating over only one pair of triangles, T_m^1 and T_m^2. This integration method involves evaluating the integrand in two points for each basis function, $\mathbf{f}_m$. The chosen points are the triangle centroids, $\mathbf{r}_m^1$ and $\mathbf{r}_m^2$. Using instead the 3-point formula gives,

$$\begin{aligned} b_m &= \int_{S_m^{\mathrm{d}}} \mathrm{d}^2\mathbf{r}\, \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}) \\ &\approx \sum_{t=1}^{2} A(T_m^t)\frac{1}{3}\sum_{p=1}^{3} \mathbf{f}_m(\mathbf{r}_{m,p}^t) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}_{m,p}^t), \end{aligned} \tag{3.19}$$

where $\mathbf{r}_{m,p}^t$ again refers to the p'th evaluation point of element t in the m'th discretization pair.

After having found these matrix elements the linear system of Eq. (3.15) must be solved. This is done by using the *LAPACK* library, see Ref. [8]. The rest of the solution is identical to the first case study solution. When the current density, $\mathbf{J}$ is estimated it may be inserted into the integral of Eq. (3.14) which now may be solved numerically giving the electric field, $\mathbf{E}_1(\mathbf{r})$.

3.6 Implementation

A program for calculating the EM field of the scattering problem is implemented in FORTRAN code and shown in appendix G. The implementation is divided into the following parts:

1. The discretization algorithm creating N_{el} discrete elements based on the boundary surface, S. The open source library *stripack* and a recursive algorithm implemented from scratch were used to create triangular discretizations, see Ref. [9] and [10] for details.

2. The FORTRAN module *disc_mod.f90* containing routines related to the discretization algorithm, e.g. routines for running a discretization algorithm and then transforming the output into the custom made data structure. The module is included in appendix G.1.

3. The FORTRAN module *calc_mod_omp.f90* containing routines related to the calculation part of the program. The module is included in appendix G.2.

4. The FORTRAN file *J_calc.f90* containing the first part of the main program. It includes setting the parameters of the problem and the calculation of the surface current density in all discretization elements. The program writes the resulting surface current density to the file *J_data*. The program code is included in appendix G.3.

5. The FORTRAN file *E_calc_omp.f90* containing the second part of the main program. The program gets the current density data from the file *J_data* and uses it to calculate the electric field amplitude by Eq. (3.14) at the observation points set in the beginning of the file. One may choose between calculating the total and the scattered electric field amplitude. The program code is included in appendix G.4.

6. The shell script *run_prog.sh* containing the commands for compiling the above mentioned group of files. The script is included in appendix G.5.

The most computationally demanding parts of the program was parallelized by using the *OPENMP* library, see Ref. [11]. More specifically this includes the calculation of the matrix elements of Eq. (3.12) in the file *E_calc_omp.f90* and Eqs. (3.17) and (3.19) in the file *calc_mod_omp.f90*, all equations applying the 3-point Gaussian quadrature formula.

3.6.1 Discretization

There are many discretization algorithms available as open source code on the internet. The output data of such algorithms may come in many different variations concerning number of discretization elements, corners per element etc. Also when designing a program from scratch decisions of this kind may be changed during the implementation process. Therefore it may be very practical to implement a custom made data structure being as general as possible. In this way the only modification to the code needed when using a new discretization algorithm is the interface which fills the custom made data structure with the output of the discretization algorithm. In addition a custom data structure may always be modified to be compatible with new features wanted in the program which may appear later in the process. The custom made data structure created for the purpose of solving the case studies is presented in Table 3.1 and it is implemented in the file *disc_mod.f90* given in appendix G.1.

Derived type	Type	Subvariable Name	Dimension [(dim1,dim2,...)]
Point	real	point	(3)
Element	integer	nr_of_corners	(1)
	integer	corners	(nr_of_corners)
Pair	integer	corners	(2)
	integer	elements	(2)
Structure	Element	elements	(:)
	Pair	neighbors	(:)
	Point	points	(:)
	Point	midpoints	(:)
	Point	quadpoints	(:)

Table 3.1: Table showing the data structure used to store the discretization. The main derived type is *Structure* containing three arrays of the type *Point*, one array of the type *Element* and one array of the type *Pair*.

As shown by Table 3.1 the whole discretization is stored in a derived type called *Structure*. This includes five arrays, three of which contain variables of the derived type *Point*, and one array for each of the types *Element* and *Pair*. A *Point* simply represents a point in 3D space, an *Element* represents an element of the discretization and a *Pair* represents a pair of elements sharing one edge. The *points* array, the *midpoints* array and the *quadpoints* array contains all discretization corner points, all element centroids and all Gaussian quadrature evaluation points respectively. The *elements* array contain all discretization elements. Each element is identified by its number of corners, *nr_of_corners*, and its corner points, *corners*, containing the indexes of the corner points stored in the array *points* in the *Structure* type. The *neighbours* array contain all element pairs sharing one edge. A *pair* is identified by the two element indexes, *elements*, referring to the array *elements* of the *Structure* type and a pair of point indexes, *corners*, referring to the *points* array of the *Structure* type.

The *disc_mod* module furthermore contains the following routines/functions:

- *stripack_convert*, generating a discretization using the *stripack* library and converting the output into the custom made data structure.
- *recursive_triangulation*, generating a discretization using the recursive method presented in Appendix F.

- *recursive_split*, splits each element into four smaller triangles.
- *point_in_list*, checks whether a list of integers contains a specific number.
- *area*, calculating the area of a single element of the discretization.
- *find_midpoints*, finding the centroids of all elements in the discretization.
- *find_neighbours*, finding all pairs of elements in the discretization sharing one edge.
- *common_2*, checking whether two arrays containing integers share two common elements.
- *normalize*, normalizes a real vector.
- *area_dist*, calculates the distribution of areas for a given discretization.

When implementing the discretization for the surface of the scatterer in the case studies the open source library, *STRIPACK*, from Ref. [9] and the recursive algorithm from Ref. [10], was applied. The *STRIPACK* library includes the routine *trplot* for displaying the discretized sphere. The result when the number of discretization points was set to 1000 is shown in Fig. 3.3 below. The source code in the file *stripack_prb.f90*, from Ref. [9], was used for the specific purpose of generating the figure. Figure 3.4 from Ref. [10] shows the corresponding result for the recursive discretization algorithm when the number of points is set to 1026. A clear difference is lower spread in shape and area of the triangles in the recursive discretization.

Discretization mesh fineness

An important consideration when creating the discretization is the total number of elements or equivalently the total number of discretization points. Too many points will result in too high computational time without a significant increase in accuracy, but too few points will give poor accuracy or ultimately a completely wrong solution. In the first case study the wavelength, λ, of the incoming wave must be small compared to the radius of the sphere in order for the KA to be valid. At the same time the wavelength must be large compared to the size of the elements in order to keep the approximation of constant integrand made in in Eq. (3.10) valid. Both requirements pushes the maximum size of the elements down and thus the minimum number of elements up. The following analysis gives a rough estimate of the minimum required number of elements in order for the computation to be accurate.

Fig. 3.3: Discretized Sphere:

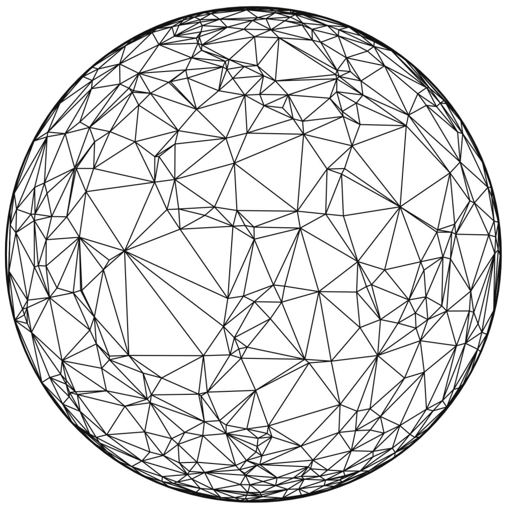

Figure 3.3: The result of discretizing a sphere using the Delaunay triangulation algorithm of the *STRIPACK* library from Ref. [9]. The number of discretization points is set to 1000.

Fig. 3.4: Discretized Sphere:

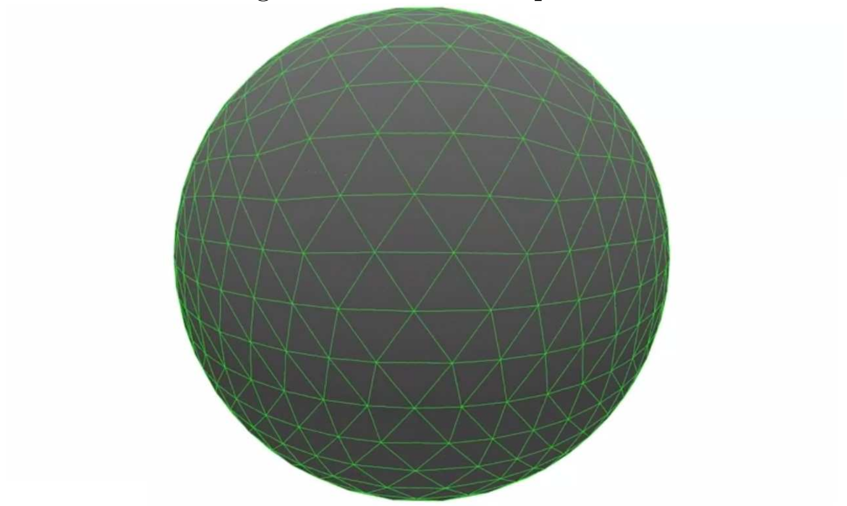

Figure 3.4: The result of discretizing a sphere using the recursive discretization algorithm. The source of the figure is Ref. [10]. The number of discretization points is 1026.

When the number of elements, N_{el}, is large the following approximation is valid for the area, A_Δ, of a triangular element,

$$A_\Delta \approx \frac{4\pi\rho^2}{N_\text{el}}, \tag{3.20}$$

where ρ is the radius of the sphere. The dominant factor regarding how fast the integrands of Eqs. (3.11), (3.12) and (3.16)-(3.19) are oscillating is the wavelength, λ, of the incoming EM wave. Let L denote the average length of an edge of a triangular element in the discretization. If the number of elements is large the area, A_Δ, of a triangle is approximately $L^2/2$ on average. We may for simplicity introduce the radius-to-wavelength ratio, a, and lambda-to-edge length ratio, b,

$$a = \frac{\rho}{\lambda}, \tag{3.21}$$

$$b = \frac{\lambda}{L}. \tag{3.22}$$

Inserting the above relations into Eq.(3.20) gives the following approximation,

$$A_\Delta \approx \frac{4\pi(a\lambda)^2}{N_\text{el}} \approx \frac{L^2}{2} = \frac{\lambda^2}{2b^2}. \tag{3.23}$$

Simplifying Eq. (3.23) and accounting for $n_\text{ep} \geq 1$ uniformly distributed evaluation points per element results in the following estimate for the number of elements given by the ratios, a and b,

$$N_\text{el} \approx \frac{8}{n_\text{ep}}\pi(ab)^2. \tag{3.24}$$

Kirchhoff's approximation and the integration method introduces minimum thresholds for a and b respectively in order to be valid; $a \geq a_\text{min}$ and $b \geq b_\text{min}$. This results in the following rough threshold approximation for the minimum required number of elements, N_el,

$$N_\text{el} \gtrsim \frac{8}{n_\text{ep}}\pi(a_\text{min}b_\text{min})^2. \tag{3.25}$$

It is normal to use the values $a_\text{min} = 10$ and $b_\text{min} = 8$ respectively for the KA and the 1-point Gaussian quadrature formula. If the 3-point Gaussian quadrature formula is applied instead the number of elements, N_el, may assumingly be reduced by a factor of $n_\text{ep} = 3$ giving the equivalent accuracy. When applying the MWR there is no longer a minimum limit concerning the ratio, a. It is important however to keep in mind that a is a problem parameter, while b is a solution parameter. Thus for a given problem only b

may be adjusted to meet the requirements concerning accuracy. Also important to underline is the assumption of large N_{el}. If the ratio, a, is very small giving too few elements, N_{el}, Eq. (3.25) obviously breaks down because the discretization mesh does no longer look like a sphere.

3.6.2 Calculation Module, Main Program and Shell Script

Some functions useful for the calculation part of the program is collected in the module *calculation_mod_omp.f90*. It includes the following routines/functions:

- *dot_prod_c*, calculating the dot product of two complex vectors.
- *dot_prod_r*, calculating the dot product of two real vectors.
- *G*, calculating the value of the Green's function.
- *fn*, calculating the function value of the n'th RWG basis function at a given point.
- *norm_vec_c*, calculating the complex normal vector (zero imaginary part) of a discretization element pointing outwards from the origin.
- *cross_c*, calculating the cross product of two complex vectors.
- *cross_r*, calculating the cross product of two real vectors.
- *J_gen*, calculating the current densities by the MWR using 1-point Gaussian quadrature integration.
- *J_gen_quad*, calculating the current densities by the MWR using parallelized 3-point Gaussian quadrature integration.

The main program is split in two and contained in the files *J_calc.f90* and *E_calc.f90*. The former file is where to control most of the parameters of the problem and the solution. This is the part of the program which finds an approximation for the current density on the discretized surface and prints the data to file. Furthermore the *J_calc.f90* file contains code for printing information about the generated discretization to file or screen (controlled by the shell script). The following list shows all the parameters which is set in the file *J_calc.f90*:

- *n*, the number of discretization points.

- *disc_number*, the discretization method used. The possible values are 1 and 2 representing respectively the Delaunay discretization method and the recursive discretization method.
- *KA*, the solution method applied. The possible values are 1 and 0 referring respectively to the KA and the MWR.
- *quad_order*, the order of the Gaussian quadrature method. The possible values are 1 and 3 representing respectively the 1-point and the 3-point quadrature formula.
- *lambda*, the wavelength of the incoming EM wave units of R. The radius of the sphere, ρ, is equal to 1.
- *E_inc_hat*, the polarization direction of the incoming EM wave.
- *k_hat*, the traveling direction of the incoming EM wave.

The file *E_calc.f90* is where the set of observation points, which is stored in the array *r_list*, are chosen and the calculation of the resulting electric field is performed. Also whether the total or the scattered field should be calculated is controlled in this part of the program. This separation of the main program is practical because finding the electric field in two or more different sets of observation points but with the same set of problem parameters only demand finding the current density once. Thus when the first part is run once the current density data written to file may be used by the second program to find the solution in different sets of observation points without needing to find the current densities for every new set of observation points.

All the source files are wrapped together by the shell script *run_prog.sh* which prints the output of the first main program to the file *output_J* (or possibly to the screen). Parts of this output file is shown in appendix G.6 for one specific set of problem and solution parameters.

3.7 Testing the Implementation

It is highly recommended to test each new component of the program immediately after it is written in order to minimize the amount of time spent on debugging. Before proceeding to the next step of solving a more general case of the EM scattering problem, where the KA is not valid, it was important to make sure that the code produced satisfactory results for the simpler case where the KA was valid. This section covers some of the most important testing procedures performed and presents the most important test results.

3.7.1 Testing the Discretization

The routines written in connection with the discretization is collected in the *disc_mod* module. Two discretization algorithms are used by the program. One is created by a library imported from the external source given in Ref. [9] as described in Section 3.6.1. The source was evaluated as reliable and nothing indicated that the library contained errors. It was however checked that the output of the program was correct. The *stripack* library includes several useful routines in this respect. The routine *trlprt* prints a list of the triangles with coordinates of the corners of each triangle in the discretization and *trplot* makes a PostScript image of the discretized sphere. Therefore by varying the number of points in the discretization and using these routines to investigate the output one may check if the program creates a discretization satisfying the needs.

Another testing procedure for the discretization part of the program is checking the correctness of the conversion of the output data from the stripack library routines into the custom made data structure. A series of *write* and *print* statements was written in the first part of the main program (see appendix G.3) in order to check if the conversion was done correctly. These routines prints the most vital information regarding the discretization and is also excellent for testing the recursive discretization, which is implemented from scratch, filling the custom made data structure directly.

3.7.2 Testing the Calculation Module Subroutines

Most of the code contained in the *calc_mod_omp* module (see appendix G.2) are simple functions trivially tested by creating a series of test cases, for example the dot and cross product functions. Two functions demanding somewhat more sophisticated testing was the *area* function and the *norm_vec_c* function. The first one was tested by summing the areas of all elements. The fact that this sum converged towards the theoretical limit, $A_{\text{sphere}} = 4\pi\rho^2$, for increasing number of discretization points strongly indicated that the function was correct. The latter was tested by comparing with the centroids of the triangles which shall be close to the normal vectors when the elements are small and the scattering object is a unit sphere. The tests showed good correspondence.

More complex was testing the implementation of the Green's function contained in the function called G. This function was taken through a thorough testing procedure in order to be as certain as possible of its correctness. First a script was written using an oscillating electric dipole in vacuum as testing case. The geometry is shown in Fig. 3.5. The dipole moment, $\mathbf{p}$, is located at the origin and oriented along the z-axis.

Fig. 3.5: Coordinate system for a single oscillating dipole:

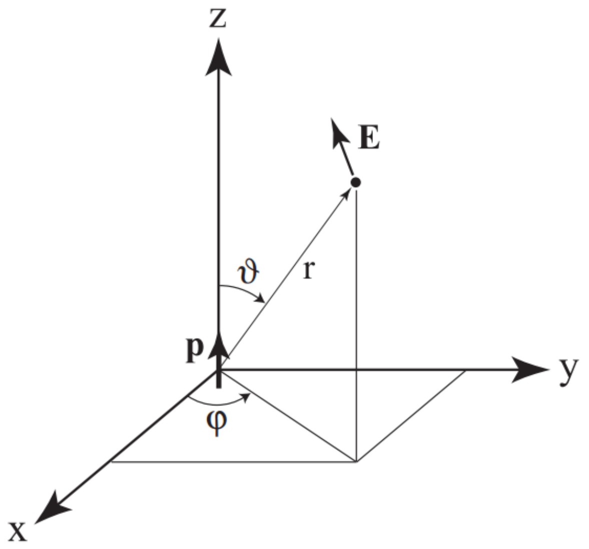

Figure 3.5: Geometry of the oscillating dipole radiation testing case. Spherical coordinates are used with the oscillating dipole located at $r = 0$ and aligned in the positive z-direction. (From Ref. [12]).

Now, the resulting electric field components are given by the following equations from Ref. [12],

$$E_r = \frac{|\mathbf{p}|\cos\theta}{4\pi\epsilon}\frac{\exp(ikr)}{r}k^2\left[\frac{2}{k^2r^2} - \frac{2i}{kr}\right], \tag{3.26}$$

$$E_\theta = \frac{|\mathbf{p}|\sin\theta}{4\pi\epsilon}\frac{\exp(ikr)}{r}k^2\left[\frac{1}{k^2r^2} - \frac{i}{kr} - 1\right], \tag{3.27}$$

$$E_\phi = 0, \tag{3.28}$$

where $\mathbf{p}$, k and ϵ denote respectively the dipole moment of the oscillating dipole, the wavenumber for EM waves in the medium and the permittivity of the medium. Furthermore, r, θ and ϕ denote the spherical coordinates of the observation point as shown in Fig. 3.5. The electric field is also given in the Green's function formalism by the following equation from Ref. [12],

$$\mathbf{E}(\mathbf{r}) = \omega^2\mu\bar{\mathbf{G}}(\mathbf{r},\mathbf{r}')\cdot\mathbf{p}, \tag{3.29}$$

where ω, μ and $\bar{\mathbf{G}}(\mathbf{r},\mathbf{r}')$ denote respectively the dipole oscillation frequency, the permeability of the medium and the Green's tensor. It is clear that each column of the Green's tensor specifies the electric field of a dipole oriented along each of the three axis. Therefore when the dipole is oriented along the z-axis only the elements of the third column contributes to the resulting electric field and are thus the only elements which may be tested with this geometry.

In order to test the elements in the first or second columns the dipole must be aligned respectively with the x-axis and the y-axis. This have the consequence that Eqs. (3.26)-(3.28) must be adapted because of the changed direction of the dipole moment, $\mathbf{p}$. This is simply a matter of transformation of the coordinates. This is simplest accomplished through Cartesian coordinates. For instance, say we want to calculate the electric field in the position $\mathbf{r} = a\hat{\mathbf{x}}$, where a is an arbitrary real number, and the dipole moment is oriented in the x-direction, $\mathbf{p} = p\hat{\mathbf{x}}$. The solution to this problem is identical to the solution of the problem of finding the electric field at the position, $\mathbf{r} = a\hat{\mathbf{z}}$, when the dipole orientation is along the z-axis, $\mathbf{p} = p\hat{\mathbf{z}}$. Now, the solution to this problem is given by Eqs. (3.26)-(3.28). Generally if $\mathbf{E}^z(x, y, z)$ is the electric field as a function of the Cartesian coordinates when the dipole moment is aligned with the z-axis, the transformation formulas are given by the following equations,

$$\mathbf{E}^x(x, y, z) = \mathbf{E}^z(-z, y, x) \tag{3.30}$$

$$\mathbf{E}^y(x, y, z) = \mathbf{E}^z(x, -z, y) \tag{3.31}$$

where $\mathbf{E}^i(x, y, z)$ denotes the electric field in position (x,y,z) when the dipole is aligned with the i-axis and $i \in \{x, y, z\}$.

Now, the main point is that if the two ways of calculating the electric field gives comparable results for a range of different observation points using all three alignment setups for the dipole moment it strongly indicates that Eqs. (3.4)-(3.7) and their numerical implementation through the function G in *calc_mod_omp* is correct. The following is an example output from a test where the observation point was set to $\mathbf{r}$=(1.0,2.0,3.0) and the dipole moment, $\mathbf{p}$, points in the z-direction. The lines 29-34 shows very good correspondence between the Cartesian components of the electric field calculated by the two different methods. This result was representative for all the tests done with this method.

```
1  dipole direction:   0.0000000       0.0000000       1.0000000
2  phi:   1.1071488     theta:  0.64052230      r:   3.7416575
3  obs:  0.99999988       2.0000000       3.0000002
4
5  Greens Dyad:
6             i           j          Re(G_ij)        Im(G_ij)
7             1           1 (-1.27916243E-02,-1.41639151E-02)
```

```
          1           2 ( 5.94545971E-04, 3.35851870E-03)
          1           3 ( 8.91818898E-04, 5.03777806E-03)
          2           1 ( 5.94545971E-04, 3.35851870E-03)
          2           2 (-1.18998038E-02,-9.12613608E-03)
          2           3 ( 1.78363826E-03, 1.00755580E-02)
          3           1 ( 8.91818898E-04, 5.03777806E-03)
          3           2 ( 1.78363826E-03, 1.00755580E-02)
          3           3 (-1.04134390E-02,-7.29835127E-04)

E_r (-1.61900710E+09, 1.39038438E+09)
E_t ( 1.76932070E+09, 2.14163533E+09)

E_r2 (-1.61900698E+09, 1.39038362E+09)
E_t2 ( 1.76932006E+09, 2.14163520E+09)
E_p2 (  48.000000    ,  192.00000    )

Comparing the absolute values:
E_abs1:  3.50306304E+09
E_abs2:  3.50306227E+09

Comparing the complex cartesian field components:
E_x1 ( 2.01724992E+08, 1.13951898E+09)
E_x2 ( 2.01724688E+08, 1.13951859E+09)
E_y1 ( 4.03450048E+08, 2.27903795E+09)
E_y2 ( 4.03449504E+08, 2.27903770E+09)
E_z1 (-2.35546496E+09,-1.65084288E+08)
E_z2 (-2.35546445E+09,-1.65084832E+08)
```

Another test regarding the implementation of the Green's function which was performed was the attempt to reproduce Fig. 8.3 in Chapter 8.3 of Ref. [12] by the Green's function formalism of Eq. (3.29). The results are presented in Fig. 3.6. It shows good correspondence between the two methods and further supports the claim of a correct implementation of the Green's function.

3.7.3 Testing the Implementation of the MWR

Even though the main motivation for the testing procedures was to make sure that the program parts involved in solving the first case study was working the tests regarding the routines written for the MWR implementation is included in this section for completeness. The main components of the MWR are the function, *fn*, calculating the value of a basis function at a specific point and the function, *J_gen*, calculating the integrals of Eqs. (3.16) and (3.18), solving the linear system of Eq. (3.15) and calculating the current density by inserting the result into Eq. (2.40). The former function may be quickly tested by checking if the function output is normal to the normal vector of the element. A more conclusive test is to choose a set of random pair of

Fig. 3.6: Asymptotic behaviour of the electric field by the two methods:

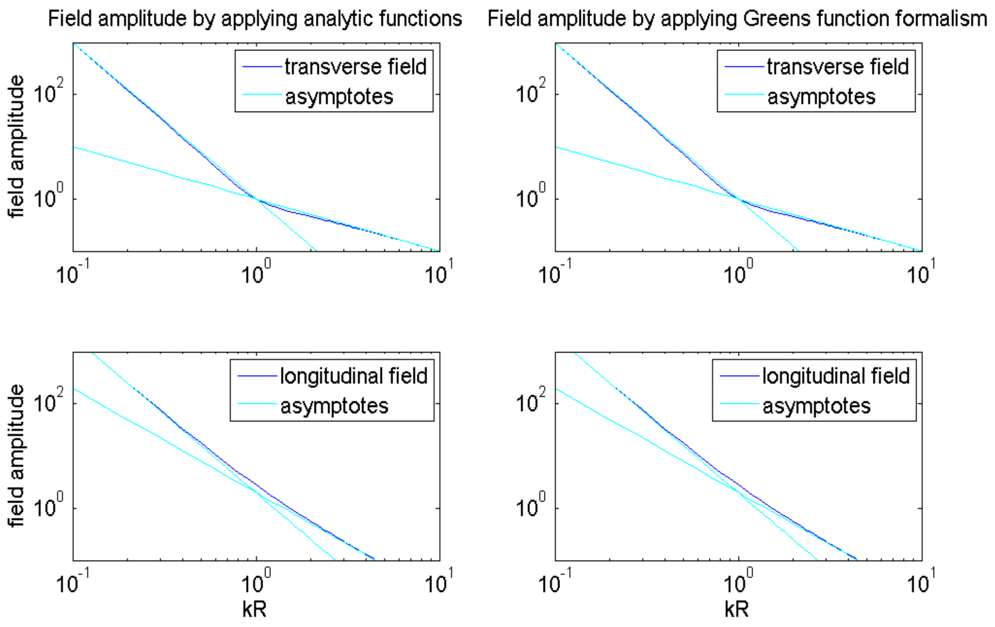

Figure 3.6: Comparison of the asymptotic bahavior of the transverse and longitudinal electric field components calculated using Eqs. (3.26) and (3.27) (left column) and by the Green's function formalism of Eq. (3.29) (right column). As can be seen from the analytic formulas the asymptotes are $(kr)^{-1}$ and $(kr)^{-3}$ for the transverse field and $(kr)^{-2}$ and $(kr)^{-3}$ for the longitudinal field. This is a reproduction of Fig. 8.3 in Ref. [12] for both methods.

elements and go through the result after each line of code to check if the algorithm behaves correctly. A similar approach is possible for the double integral algorithm of the *J_gen* and *J_gen_quad* functions, namely picking out random elements of the matrix and checking the calculation line by line. The linear system solver from the *LAPACK* library may be tested by calculating the average absolute difference,

$$D = \frac{1}{N_{\text{ed}}} \sum_{m=1}^{N_{\text{ed}}} \frac{\left| \sum_{n=1}^{N_{\text{ed}}} D^{1}_{mn} \alpha_n - b_m \right|}{|b_m|}. \tag{3.32}$$

The result was typically in the order of magnitude, $D \sim 10^{-5} - 10^{-6}$, which is around the expected with the applied single precision and clearly indicates a correct solution of the linear system.

3.7.4 Testing the Scattering Calculation

The last part is the main program split between the *J_calc.f90* file and the *E_calc_omp* file. The vital components to check in this part of the code are the constants used, the observation point list and the integration algorithm. The first two is simply tested by printing out the data being used by the program. The latter may first be tested by simple test cases, for example setting all current densities to zero and checking if the resulting electric field in all observation points is equal to the incoming field. A more rigorous line by line approach as described above is a natural follow up.

3.7.5 Testing the Complete Program and the Conclusion to the Tests

After testing all the different parts of the code separately it is possible to start testing the complete program as a whole. This is done by selecting the parameters involved and choosing a set of observation points before running the full program. The results of these simulations are presented and discussed in Section 4 applying the testing criteria presented in Section 3.8. One of the main motivations for the testing procedures presented in the current section was to assure that the program parts involved in solving the first case study was working well before implementing the solution of the more complex second case study. The test results and the final electric field amplitude plots combined led to the conclusion that the program was working correctly and was ready to be extended to solving the more general second case study.

3.8 Evaluating the Final Results

This section presents a set of criteria which may be used to evaluate the correctness of the final results for the EM field. One of the reasons for choosing a sphere as the shape of the scatterer was because theoretical results are available for the scattering problem with a scatterer of this shape. It however proved to be difficult to find theoretical results directly comparable with the output of the program written. Therefore other testing criteria with the ability to shed light on the correctness of the numerical results was found instead. It is however important to note that these criteria are necessary, but not sufficient to positively conclude regarding the correctness of the results. That is, they may disprove but not prove the correctness. They may however give a strong indication whether the results are good or not.

3.8.1 Interference Pattern

One of the most fundamental facts when it comes to EM waves is that they follow the superposition principle because of the linearity of Maxwell's equations. This means that when an observation point receives EM waves from different sources simultaneously the net effect is a superposition of all individual source contributions. This leads at some positions to constructive interference, where the net effect is a greater amplitude than each of the individual contributions. At other positions destructive interference is the net result meaning the contributions cancel each other out. This is indeed exploited by the Green's function formalism by splitting the scatterer into a finite number of imaginary surface current density sources all contributing to the resulting field at the observation point. Therefore it is expected that the resulting field around the scatterer forms an interference pattern.

The resulting field will be presented in the shape of contour plots of the electric field amplitude. This naturally doubles the spatial frequency as the negative peaks are turned into positive ones. When observing the EM field in a plane the observed spatial frequency can never exceed the highest spatial frequency component of the individual contributions. Basic trigonometric identities reveal that a superposition of sinusoidal waves can never have higher frequency than the highest frequency among the components. Therefore it is expected that the highest spatial frequency of the calculated electric field amplitude is maximum twice the spatial frequency of the incoming EM wave due to the absolute value operation. Thus if the incoming wave has a wavelength of, λ, the smallest peak-to-peak distance in the interference pattern shown by the field amplitude is expected to be $d^{\lambda}_{\text{min}} = \lambda/2$.

Figure 3.7 shows a typical solution of the EM scattering problem. It pictures a contour plot of the EM wave intensity scattered by a metal cylinder in vacuum. The intensity, I, of an EM wave in vacuum is proportional to the square of the electric field amplitude, see Ref. [13],

$$I \equiv \frac{1}{2} c\epsilon_0 E_0^2, \tag{3.33}$$

where E_0 denote the amplitude of the electric field. Thus the above discussion regarding interference and spatial frequency is still valid. Figure 3.7 shows exactly as predicted an interference pattern with no spatial frequency components higher than twice the incoming spatial frequency. The region where the highest spatial frequency components appear is directly in front the cylinder. Here, one may count roughly ten peaks in the intensity over a range equal to the cylinder radius, R_c, resulting in a peak-to-peak distance,

$$d \approx \frac{R_c}{10} = \frac{5\lambda}{10} = \frac{\lambda}{2} = d^{\lambda}_{\text{min}} \tag{3.34}$$

which is just as expected for the highest spatial frequency component of the intensity plot. Even though Fig. 3.7 involves a cylinder as scattering object the situation is very similar to what is studied in this thesis. It is expected that the interference pattern should closely resemble the presented plot and the highest spatial frequency components of the amplitude plot should never exceed the discussed limit of twice the spatial frequency of the incoming wave.

Fig. 3.7: Contour plot of scattered EM wave intensity from cylinder:

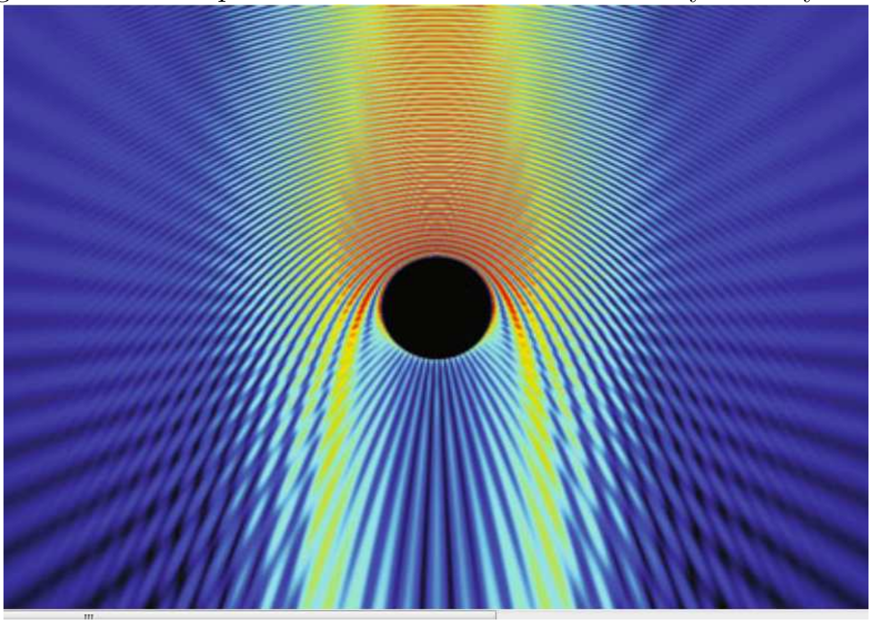

Figure 3.7: Contour plot from Ref. [14] of the intensity of the EM wave scattered by a metal $[\epsilon(\omega) = -17]$ cylinder placed in vacuum. The incoming wave approaches the cylinder from the top of the figure. The width of the incoming beam is $w = 15\lambda$ and the radius of the cylinder is $R_c = 5\lambda$, when λ denote the wavelength of the incoming EM wave.

3.8.2 Symmetries

The scattering problem investigated through the two case studies has got some symmetries which gives information about what to expect from the solution. The fact that the incoming wave is a linearly polarized planar EM wave and the scatterer has got a spherical shape implies some testable symmetries in the solution plots.

Figure 3.8 shows the three dimensional geometry of the problem when having a spherical scatterer. The center of the sphere is located at the origin and the cross section planes through the center with normal vectors aligned with each of the three basis vectors of the space is shown. The wave vector, $\mathbf{k}$, is chosen to point in the positive x-direction. Now the polarization vector of the incoming EM wave must be perpendicular to the wave vector and must therefore have zero x-component. Two possible alternatives are shown in Fig. 3.8, namely $\mathbf{E}^{\text{inc}} = \mathbf{E}_a^{\text{inc}}$ along the positive z-direction and $\mathbf{E}^{\text{inc}} = \mathbf{E}_b^{\text{inc}}$ along the negative y-direction.

Fig. 3.8: Scattering Geometry when having Spherical Scatterer:

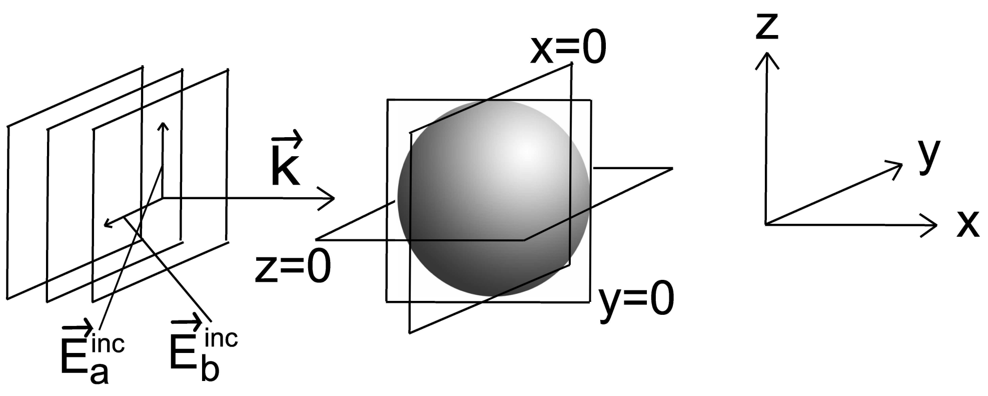

Figure 3.8: Geometry of the scattering problem with spherical scatterer and planar incoming EM wave. Two possible polarization vectors, $\mathbf{E}_a^{\text{inc}}$ and $\mathbf{E}_b^{\text{inc}}$, are shown.

If either of these alternatives are chosen for the polarization vector there arise a couple of symmetries in the solution. As the incoming EM wave *and* the scattering object in both these alternatives has mirror symmetry with the planes $z = 0$ and $y = 0$ as symmetry axes, the solution must also have both these symmetries in both the mentioned setups. Note however that even though the scatterer has mirror symmetry with the plane $x = 0$ as symmetry axis, the incoming EM wave do not because the wave vector, $\mathbf{k}$, points in the positive x-direction. Therefore the solution does not need to have this symmetry.

Another testable symmetry in the solution involves calculating the solution in the plane $z = 0$ for the case of $\mathbf{E}^{\text{inc}} = \mathbf{E}_a^{\text{inc}}$ and then calculating the solution in the plane $y = 0$ for the case of $\mathbf{E}^{\text{inc}} = \mathbf{E}_b^{\text{inc}}$. These two solutions should be equal because the scatterer has rotational symmetry with the direction of the wave vector as symmetry axis. Testing this symmetry will effectively be a test of the discretization applied. It checks if the discretization mesh "looks" equal upon rotation when using a given wavelength. This is naturally wavelength dependent as a smaller wavelength require more discretization elements in order to get a sphere which "looks" symmetrical upon rotation.

3.8.3 Energy Conservation

Energy can neither be created nor disappear only converted into other forms. The energy of an isolated system must therefore remain constant. This is one of the fundamental conservation laws in physics. The total amount of energy is conserved in all known physical processes.

Let our system be contained inside an imaginary boundary surface surrounding the scatterer. The system is not isolated due to the continuous transportation of energy by the EM waves moving through the system. However, since the scatterer is perfectly conducting no energy is absorbed inside the system. This means that all energy entering the system must also exit. Furthermore, since the incoming EM wave has a constant amplitude and direction the energy flux entering through the system surface must also be constant when averaged in time. Since no energy is neither created nor absorbed inside the system this must also be the case regarding the total amount of energy inside the system. Therefore the time averaged energy flux leaving the system must equal the time averaged energy flux entering the system. This may also be expressed through Poynting's theorem, see Ref. [13],

$$\frac{dW}{dt} = -\frac{dU_{\mathrm{em}}}{dt} - \oint_{S_b} \mathbf{S} \cdot \hat{\mathbf{n}} \, \mathrm{d}^2\mathbf{r}, \tag{3.35}$$

where $\hat{\mathbf{n}}$ is a unit normal vector on the boundary surface. The first term of Eq. (3.35) represents the amount of work done by the EM field per time unit acting on the charges inside the boundary surface, S_b. The second term represents the change per time unit of the amount of energy stored in the EM field inside the boundary surface. The last term represents the energy flow across the boundary surface per time unit, where the vector quantity, $\mathbf{S}$, called Poynting's vector is given by,

$$\mathbf{S} = \frac{1}{\mu_0}(\mathbf{E} \times \mathbf{B}). \tag{3.36}$$

By using Faraday's law, Eq. (2.2), and electric and magnetic fields, $\mathbf{E} = \mathbf{E}_0 \cos(\mathbf{k} \cdot \mathbf{r} - \omega t)$ and $\mathbf{B} = \mathbf{B}_0 \cos(\mathbf{k} \cdot \mathbf{r} - \omega t)$, it is simple to show the following relation between electric and magnetic field amplitudes in a plane wave moving through vacuum,

$$\mathbf{B}_0 = \frac{k}{\omega}(\hat{\mathbf{k}} \times \mathbf{E}_0) = \frac{\hat{\mathbf{k}} \times \mathbf{E}_0}{c} \tag{3.37}$$

where $\hat{\mathbf{k}}$ denote a unit vector in the traveling direction of the EM wave. Inserting this relation into Eq. (3.36) leads to,

$$\begin{aligned} \mathbf{S} &= \frac{1}{\mu_0}(\mathbf{E}_0 \times \frac{\hat{\mathbf{k}} \times \mathbf{E}_0}{c}) \cos^2(\mathbf{k}\cdot\mathbf{r} - \omega t) \\ &= \frac{1}{\mu_0 c}|\mathbf{E}_0|^2 \cos^2(\mathbf{k}\cdot\mathbf{r} - \omega t)\hat{\mathbf{k}} \end{aligned} . \tag{3.38}$$

This expression shows that the energy flux per time unit fluctuates with time, however time averaging this quantity, remembering that a squared cosine averages to $\frac{1}{2}$ and $\epsilon_0\mu_0 = c^2$, gives the following constant expression,

$$\langle \mathbf{S} \rangle_t = \frac{1}{2\mu_0 c}|\mathbf{E}_0|^2 = \frac{1}{2}c\epsilon_0|\mathbf{E}_0|^2\hat{\mathbf{k}} = I\hat{\mathbf{k}}, \tag{3.39}$$

which identifies the intensity, I, of the EM wave from Eq. (3.33) as the amplitude of the time averaged Poynting vector.

Now, the first two terms of Eq. (3.35) time averages to zero because no work is done by the fields and no energy is absorbed inside the boundary surface. This leads to the following simplification,

$$\oint_{S_b} \langle \mathbf{S} \rangle_t \cdot \hat{\mathbf{n}}\, \mathrm{d}^2\mathbf{r} = 0, \tag{3.40}$$

which states the above mentioned equivalence in the average incoming and outgoing energy flux per time unit.

This is very convenient, because the average energy flux entering the system per time unit may be easily calculated, since we control the parameters of the incoming wave. This must as we now know equal the average amount of energy carried by the scattered EM field continuously leaving through the boundary surface of the system per time unit. Equations (3.39) and (3.40) shows that this amount may be calculated by evaluating the scattered electric field amplitude at all points at the boundary surface of the system. This gives therefore a direct way of testing whether the calculated solution of the scattering problem satisfies the law of energy conservation.

The simplest way of performing this test is by choosing a spherical boundary surface centered at the center of the scatterer with radius ρ_S larger that the radius, ρ, of the scatterer. The situation is illustrated in Fig. 3.9 below.

Fig. 3.9: Geometry of Energy Conservation test:

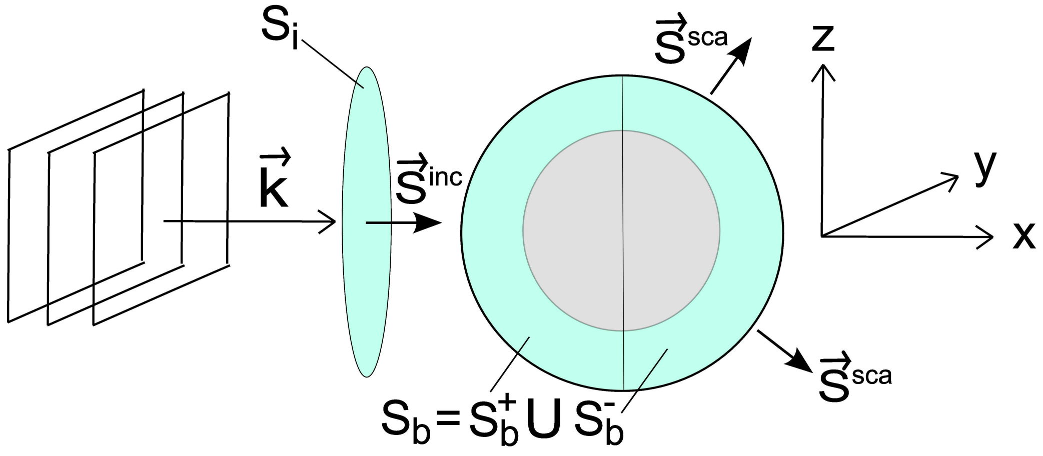

Figure 3.9: Illustration of the geometry of the test regarding energy conservation.

It is convenient to separate between Poynting's vector for the incoming EM wave, $\mathbf{S}^{\mathrm{inc}}$, and Poynting's vector for the scattered EM wave, $\mathbf{S}^{\mathrm{sca}}$. The time averaged incoming amount of energy per unit time, denoted F_{in}, is now given by,

$$F_{\mathrm{in}} = \int_{S_b^+} \langle \mathbf{S}^{\mathrm{inc}} \rangle \cdot \hat{\mathbf{n}} \, \mathrm{d}^2\mathbf{r} = \int_{S_i} \langle \mathbf{S}^{\mathrm{inc}} \rangle \cdot \hat{\mathbf{n}} \, \mathrm{d}^2\mathbf{r} = \int_{S_i} I^{\mathrm{inc}} \, \mathrm{d}^2\mathbf{r}, \tag{3.41}$$

where S_b^+ is the hemisphere of the system boundary surface which is hit from the outside by the incoming plane wave and I^{inc} is the intensity of the incoming EM wave, also known as the *irradiance*. The integration domain, S_i, of the right-hand side integral is a circle in front of the boundary surface of the system with the same radius, ρ_S. This projection makes the unit wave vector, $\hat{\mathbf{k}}$, parallel to the new surface normal, $\hat{\mathbf{n}}$. The simplification is possible because the dot product operation corresponds to projecting the integration domain, S_b^+, on to a plane normal to the wave vector, $\mathbf{k}$, which gives exactly the surface, S_i.

The average amount of energy leaving the system per time unit, denoted F_{out}, may similarly be expressed in the following way,

$$F_{\mathrm{out}} = \oint_{S_b} \langle \mathbf{S}^{\mathrm{sca}} \rangle \cdot \hat{\mathbf{n}} \, \mathrm{d}^2\mathbf{r} = \oint_{S_b} I^{\mathrm{sca}} \, \hat{\mathbf{k}}^{\mathrm{sca}} \cdot \hat{\mathbf{n}} \, \mathrm{d}^2\mathbf{r}, \tag{3.42}$$

where I^{sca} and $\hat{\mathbf{k}}^{\mathrm{sca}}$ denote respectively the intensity and wave vector direction of the scattered EM wave. There are two possible approaches of calculating this integral. One alternative is to calculated both the scattered electric field and the scattered magnetic field at evaluation points throughout the boundary surface S_b and using Eq. (3.37) to calculate the traveling direction of the scattered EM field, $\hat{\mathbf{k}}^{\text{sca}}$. Another alternative is to extend the radius of the boundary surface, S_b, and using the approximation, $\hat{\mathbf{k}}^{\text{sca}} \approx \hat{\mathbf{r}}$. All the non-radial terms of the wave vector of the scattered EM wave must vanish far from the scatterer because all the scattered waves move undisturbed in a straight line through the vacuum of space after leaving the scatterer surface. Far from the sphere the scatterer looks like a point and the wave vector must approach the limit of being parallel with a line originating in the center of this "point". This line and thus the asymptotic direction of the wave vector is aligned with the radial unit vector $\hat{\mathbf{r}}$.

After having calculated both the average energy transported into and out of the system per time unit, F_{in} and F_{out}, the test is simply to check if they are equal, satisfying the requirement set by the law of energy conservation.

3.8.4 Rayleigh Scattering

If the wavelength of the incoming EM wave is much larger than the radius of the sphere, $\lambda \gg \rho$, an approximation called *Rayleigh Scattering* is valid. This approximation simplifies the equations leading to the following wavelength, distance and scattering angle dependence of the scattered intensity, from Eq. (11) in Ref. [15],

$$I^{\text{sca}} \sim \frac{I_r^{\text{inc}} + I_l^{\text{inc}} \cos^2\theta}{|\mathbf{r}|^2 \lambda^4}, \tag{3.43}$$

where I_r^{inc} and I_l^{inc} denote the incoming intensity of the field component polarized perpendicular and parallel to the scattering plane respectively, where the scattering plane contains the incoming and observed scattered EM waves. Furthermore, λ denotes the wavelength of the incoming EM wave, $|\mathbf{r}|$ denotes the distance to the scatterer and θ denotes the angle with the incoming wave vector.

Equation (3.43) implies a fast decreasing scattered intensity with increasing wavelength. This is the explanation for the red sunset and blue sky mentioned in the introduction. The short wavelengths, observed as blue light, are scattered away in a significantly greater extent than the longer wavelengths, observed as red light. Thus the light rays from the sun having to travel through longer distances of atmosphere are observed as red and the sky primarily consisting of scattered sunlight is observed as blue. (Purple light consists of even smaller wavelengths, but the sunlight contain less energy at these wavelengths and the human eye is less sensitive for light in this part of the spectrum. The sky therefore looks blue instead of purple.)

A second implication of Eq. (3.43) is the introduction of a new symmetry plane for the scattered field. The expression is symmetrical around $\pi/2$ and $3\pi/2$, thus implying that there should be no difference between the forward and backward scattering patterns. The plane $x = 0$ in Fig. 3.8 is thus the third case of mirror symmetry in the solution plot of the scattered field.

3.8.5 Error Functions

When comparing different sets of data a function is needed to quantify the difference. The following function is a much used alternative,

$$\mathrm{Err}^1_{x,y} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[\frac{x(i)-y(i)}{x(i)}\right]^2}, \tag{3.44}$$

which is called the Root Mean Square (RMS) value of the relative difference, where x and y are sets of N data points. If the data set, x, approaches or equals zero for some i the function is numerically unstable or not defined at all, so a second alternative is needed. A possible alternative numerically stable for data sets, x and y, where the data set x has a nonzero mean is the following,

$$\mathrm{Err}^2_{x,y} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left[x(i)-y(i)\right]^2}}{\frac{1}{N}\sum_{i=1}^{N}x(i)}. \tag{3.45}$$

Chapter 4

Results and Discussion

This section presents the results of the numerical calculations of the electric field amplitude for a set of three separate problems. The three problems have the following wavelength to scatterer radius relations, $\lambda \ll \rho$, $\lambda \approx \rho$ and $\lambda \gg \rho$. All three are examples of the second case study, but only the first one may be regarded an example of the first case study and is thus the only problem solved by the method involving the KA. The other two problems are solved by the MWR. The results for each of the three problems are presented in separate sections and evaluated according to the evaluation criteria presented in Section 3.8. Evaluation regarding conservation of energy was left out because of limited time.

First is a section included discussing the two discretization methods and an important difference between them found during the production of calculation results. The input parameters for each calculation is listed in Section 3.6.2. Common for all figures generated (except for the figures of Section 4.1) is *quad_order* $= 3$, *disc_type* $= 2$ and *k_hat* $= [1\ 0\ 0] = \hat{\mathbf{x}}$. This implies the use of the 3-point Gaussian integration formula, the recursive triangulation method and an incoming wave heading in the positive x-direction. The remaining input parameters, *n*, *KA*, *lambda*, and *E_inc_hat* are denoted respectively, N_p, KA, λ and $\hat{\mathbf{E}}^{\mathrm{inc}}$.

4.1 The Discretization Methods

Early in the process of producing calculation results the *STRIPACK* library was used for generating the discretization. The numerical calculations produced some reasonable results with the KA for the problem involving small wavelength as shown in Fig. 4.1.

Fig. 4.1: Contour plot using *STRIPACK* discretization:

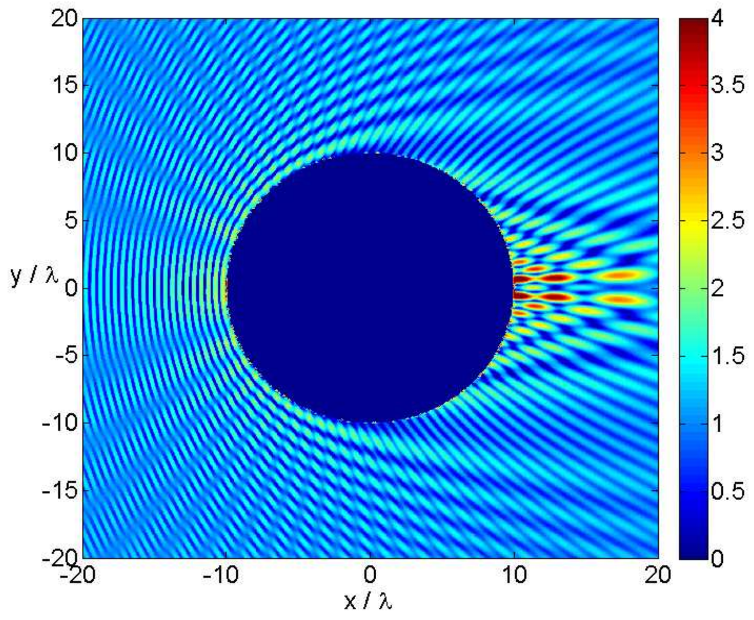

Figure 4.1: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\rm inc}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\rm p} = 80500$, $KA = $ true, $\lambda = 0.1\rho$, $\hat{\mathbf{E}}^{\rm inc} = \hat{\mathbf{z}}$, *k_hat* $= [1\ 0\ 0]$, *quad_order* $= 1$, *disc_type* $= 1$.

Without going into details the plots generated in the KA domain by using the *STRIPACK* discretization did not seem flawed with respect to symmetry and interference pattern considerations. However, this was completely different when applying it to the MWR, which involves considerably less discretization points. One example is shown in Fig. 4.2.

It shows no symmetry at all around the plane $x = 0$, which it definitely *should* according to the argument given in Section 3.8.2. The number of elements was above the critical limit from Eq. (3.25), which gives,

$$N_{\rm el} \gtrsim 8\pi(ab_{\rm min})^2 = 8\pi 8^2 \approx 1609, \tag{4.1}$$

where the ratio, $a = \dfrac{\rho}{\lambda} = 1$, was inserted. The actual number of points used, $N_{\rm el} = 9996$ ($= 2(N_{\rm p} - 2)$ found empirically), was definitely above the critical limit. So there had to be another explanation. After having gone through all the testing procedures presented in Section 3.7 the problem was most probably not caused by a programming error.

Fig. 4.2: Contour plot using *STRIPACK* discretization:

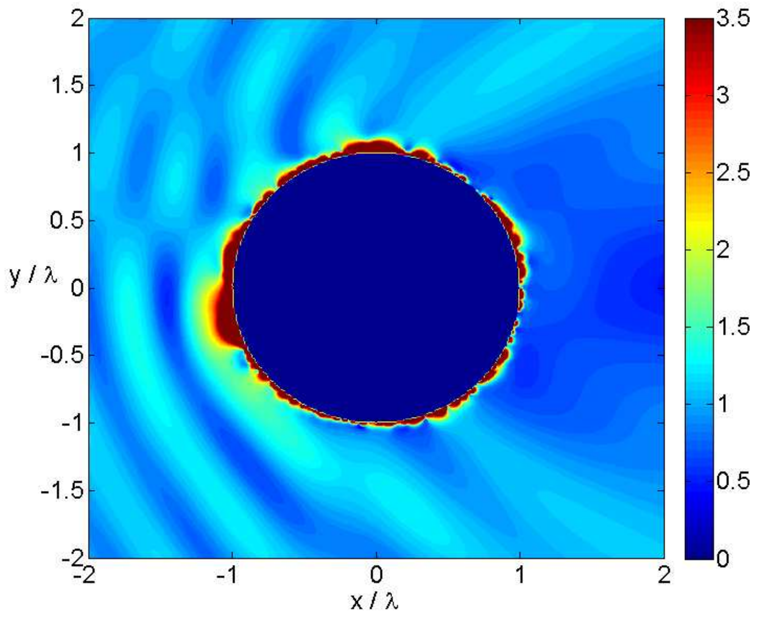

Figure 4.2: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 5000$, $KA =$ false, $\lambda = 1.0\rho$, $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$, $k\_hat = [1\ 0\ 0]$, $quad\_order = 1$, $disc\_type = 1$.

After contemplating some possible causes the discretization came up as a possible source of the trouble. In fact, just by looking at Fig. 3.3 it is clear that the shape and size of each element is varying considerably. A deeper analysis revealed exactly how much the element areas varied for the *STRIPACK* discretization method compared to the recursive triangulation method. The area distributions for $N_{\mathrm{p}} = 4098$ is shown in Fig. 4.3. The mean, standard deviation and maximum of the two distributions are shown in Table 4.1. The shape and area of the recursive discretization is clearly varying much less.

The maximum area of the *STRIPACK* discretization is actually more than ten times larger than the mean area. This means that the spatial sampling rate for the numerical integrals no longer fulfills the requirements of Eq. (3.25) on the whole surface when having more than a tenth of the critical number of elements. For the recursive discretization the maximum area is less than twice the size of the mean area. The number of required elements in order for the numerical integration to be correct is therefore

Fig. 4.3: Area distribution of the two discretization methods:

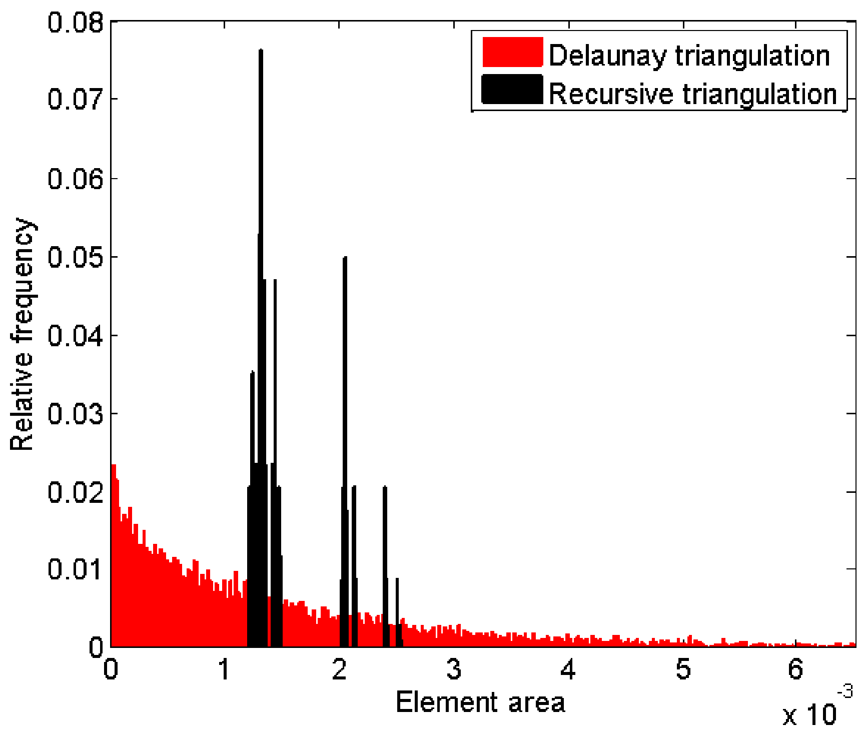

Figure 4.3: Normalized area distribution for the Delaunay method from the *STRIPACK* discretization library and the recursive method for number of discretization points, $N_\mathrm{p} = 4098$, corresponding to the number of elements, $N_\mathrm{el} = 8192$.

Area data	Recursive	Stripack
mean	$1.5328 \cdot 10^{-3}$	$1.5313 \cdot 10^{-3}$
standard deviation	$3.7164 \cdot 10^{-4}$	$1.6568 \cdot 10^{-3}$
maximum	$2.5322 \cdot 10^{-3}$	$1.6900 \cdot 10^{-2}$

Table 4.1: Table of area data for the elements of the two discretization methods for $N_\mathrm{p} = 4098$ and $\rho = 1$. The expected element area for N_el equally sized elements on the surface is $A_\mathrm{exp} = \dfrac{4\pi}{8192} = 1.53398$.

significantly higher for the *STRIPACK* discretization than for the recursive discretization method. So when instead applying the recursive discretization method the predicted symmetries reappeared in the results even when using less discretization points as shown by Section 4.3. For this reason the recursive discretization method was applied in the generation of all the numerical results presented in the following text.

4.2 Results for $\lambda \ll \rho$

The first problem studied involved the wavelength set to, $\lambda = 0.1\rho$. The KA is valid in this case, therefore all results in this section is generated with the input parameter, $KA =$ true, following the right-most route in Fig. 3.1.

Fig. 4.4: Contour plot of relative electric field amplitude:

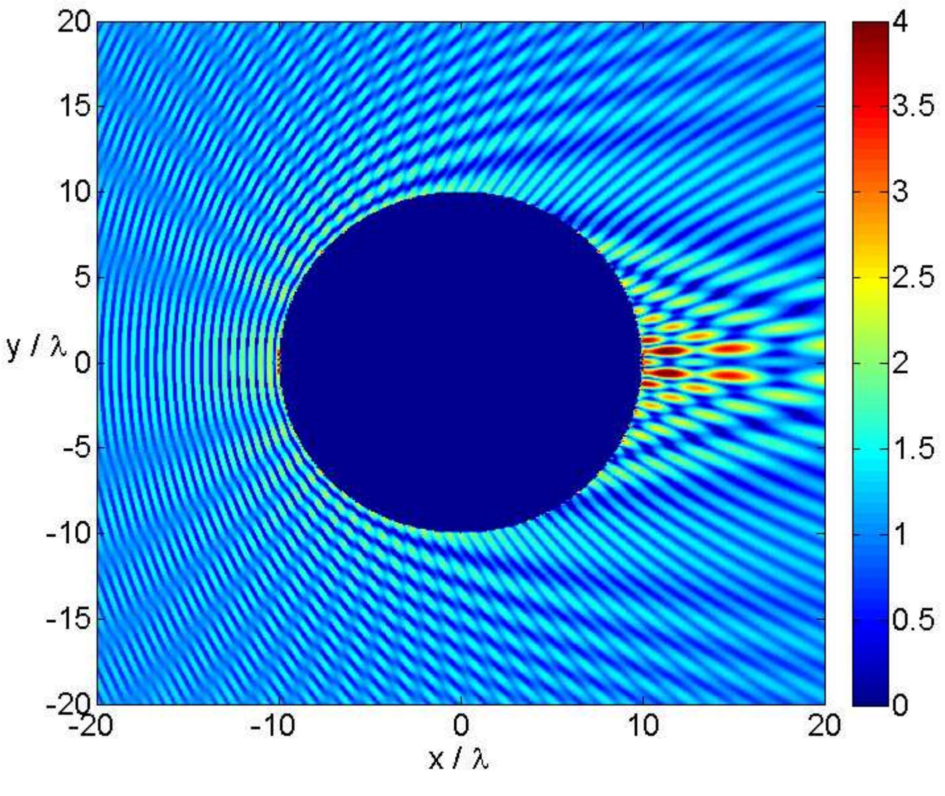

Figure 4.4: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 65538$, $KA =$ true, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

Figure 4.4 and 4.5 shows as expected an interference pattern around the scatterer very similar to the one present in Fig. 3.7. The biggest difference is probably in the shadow region behind the scatterer where the sphere produces

Fig. 4.5: Contour plot of relative electric field amplitude:

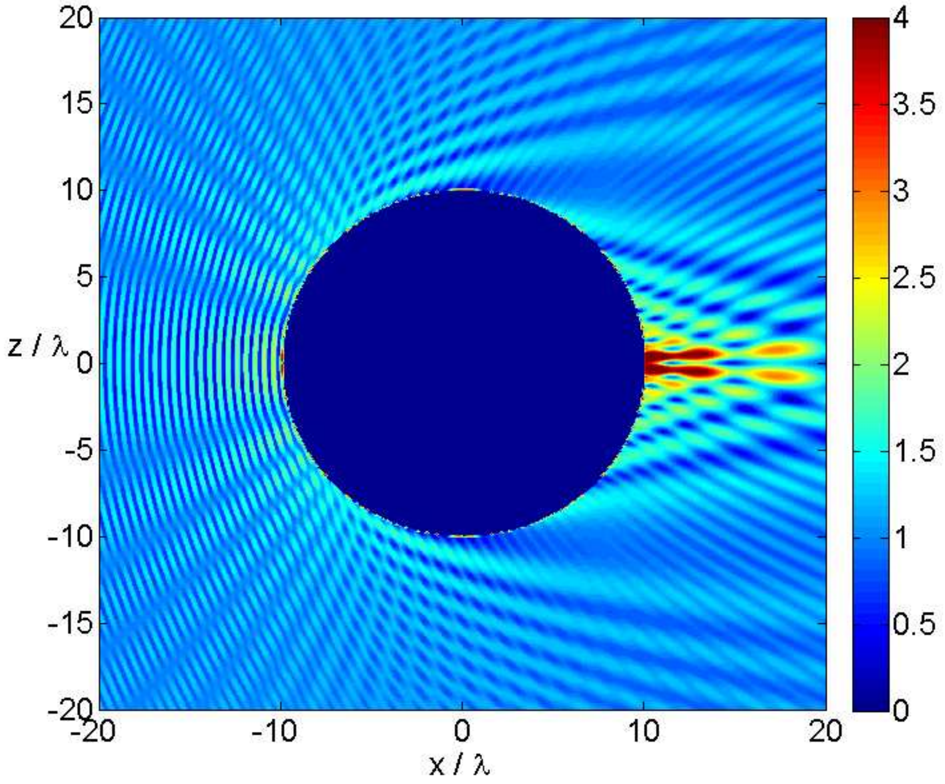

Figure 4.5: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, in the plane $y = 0$. The input parameters were set to the following values, $N_{\text{p}} = 65538$, $KA = \text{true}$, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

a strong focusing effect. This is at least not in conflict with the intuitive expectation when remembering that the cylinder is infinitely long in one of the dimensions, while the spherical shape allows the EM wave to bend around the scatterer in both the directions perpendicular to $\mathbf{k}$. Whether or not this effect is supported by theory or, even better, by physical experiments needs further investigation.

The shortest peak-to-peak ratios are seen to be located in the region just in front of the sphere. Here, it is possible in both figures to count twenty peaks in a distance corresponding to ten times the wavelength, λ. This gives a peak-to-peak distance of,

$$d = \frac{10\lambda}{20} = \frac{\lambda}{2} = d^{\lambda}_{\text{min}}, \tag{4.2}$$

which is exactly the expected minimum theoretical peak-to-peak distance.

Figures 4.4 and 4.5 in addition to Fig 4.6 indicates the expected mirror symmetries presented in Section 3.8.2 with the planes, $y = 0$ and $z = 0$, as symmetry planes. These symmetries are more accurately displayed by the Figs. 4.7 and 4.8. The mirror symmetry around $\theta = \pi$ in the figures translates to having the expected mirror symmetries with $y = 0$ and $z = 0$ as symmetry planes.

Fig. 4.6: Contour plot of relative electric field amplitude:

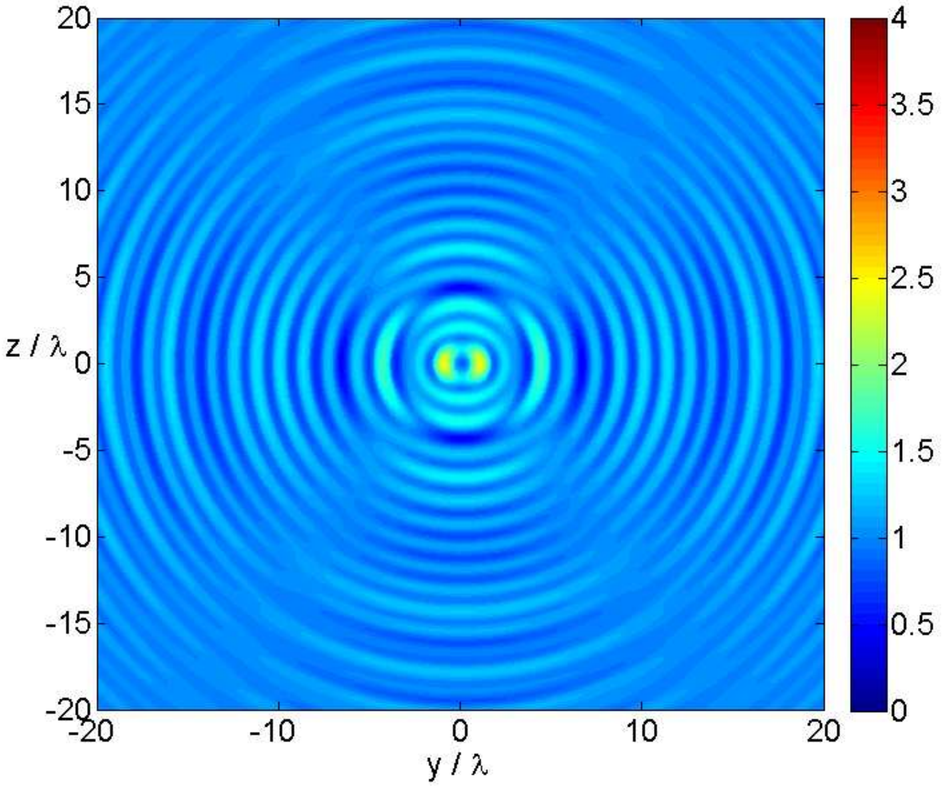

Figure 4.6: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $x = 2$. The input parameters were set to the following values, $N_{\mathrm{p}} = 65538$, $KA = \mathrm{true}$, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

Fig. 4.7: Mirror symmetry about the plane $y = 0$:

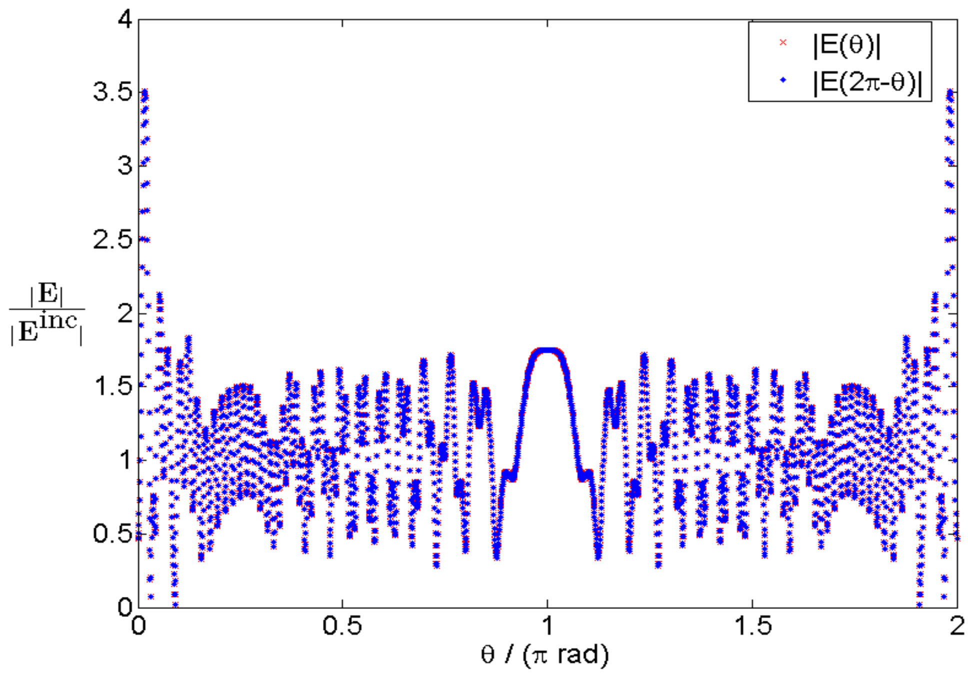

Figure 4.7: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, around a circle with radius, $\rho_{\mathrm{circle}} = 1.5\rho$, in the plane $z = 0$ centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\mathrm{Err}^1 = 2.80 \cdot 10^{-5}$. The input parameters were set to the following values, $N_{\mathrm{p}} = 65538$, $KA = \mathrm{true}$, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

Fig. 4.8: Mirror symmetry about the plane $z = 0$:

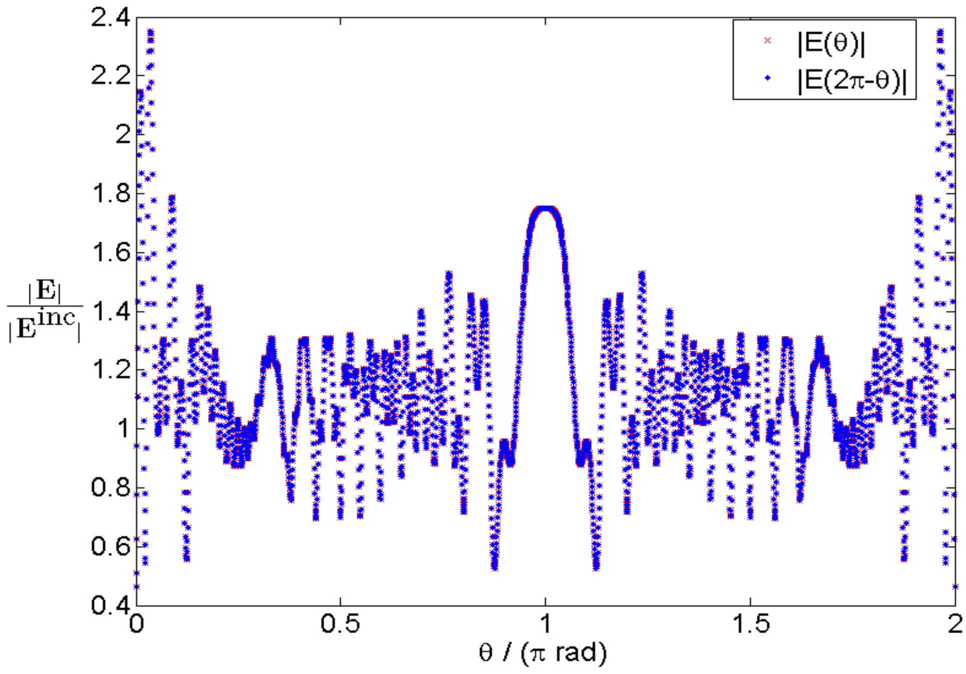

Figure 4.8: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 1.5\rho$, in the plane, $y = 0$, centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\text{Err}^1 = 1.03 \cdot 10^{-5}$. The input parameters were set to the following values, $N_{\text{p}} = 65538$, $KA = \text{true}$, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

In order to test the quality of the discretization the polarization direction and the plane containing the observation points was rotated $\pi/2$ radians without rotating the discretization mesh. This tests if the discretized sphere "looks" different upon rotation when the sensing wavelength is set to, $\lambda = 0.1\rho$. The contour plot of the relative electric field amplitude in the plane $y = 0$ with polarization direction, $\hat{\mathbf{E}} = -\hat{\mathbf{y}}$ is shown in Fig. 4.9. Qualitatively it looks identical to the contour plot of Fig. 4.4, as expected. The values on a circle in the plane, $z = 0$, when having a $\hat{\mathbf{z}}$ polarized incoming wave is quantitatively compared with the values on a circle in the plane, $y = 0$, having a $(-\hat{\mathbf{y}})$ polarized incoming wave in Fig.4.10

Fig. 4.9: Contour plot of relative electric field amplitude:

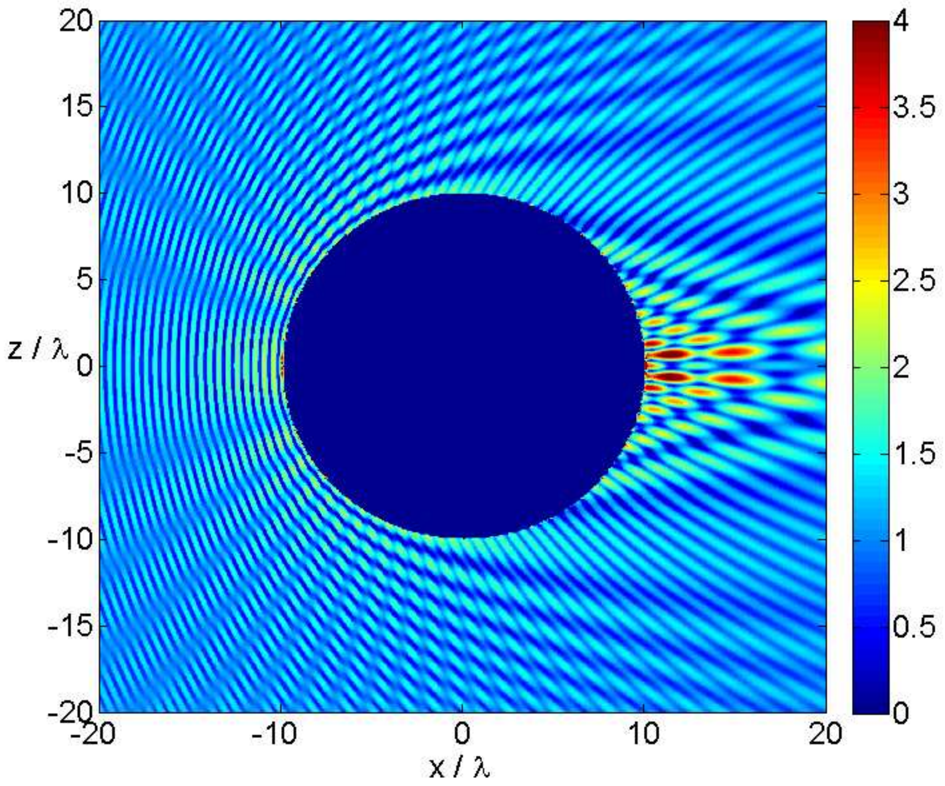

Figure 4.9: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $y = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 65538$, $KA = \text{true}$, $\lambda = 0.1\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = -\hat{\mathbf{y}}$.

Fig. 4.10: Symmetry upon rotation around incoming wave vector axis:

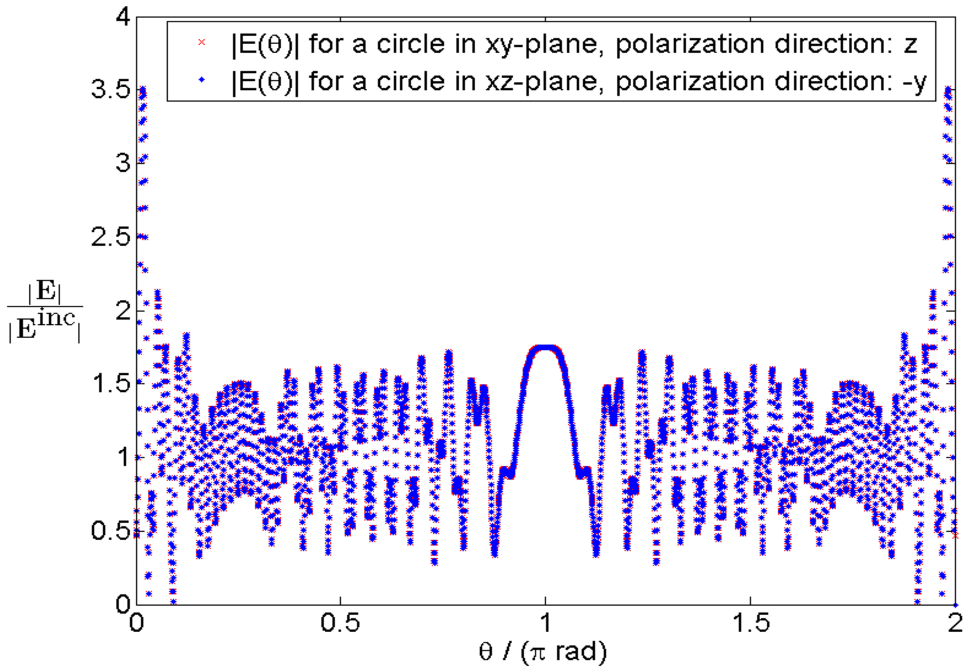

Figure 4.10: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 1.5\rho$, in the planes $z = 0$ and $y = 0$, centered at the origin. The starting point, $\theta = 0$, represents for both circles the point $[x,\ y,\ z] = [1,\ 0,\ 0]$.The polarization of the incoming EM wave was set to respectively $\hat{\mathbf{z}}$ and $-\hat{\mathbf{y}}$. The other input parameters were set to the following values, $N_{\text{p}} = 65538$, $KA = \text{true}$ and $\lambda = 0.1\rho$. The RMS of the relative error was found to be, $\text{Err}^1 = 9.95 \cdot 10^{-5}$.

4.3 Results for $\lambda = \rho$

The second problem studied involved the wavelength set equal to the radius of the scatterer, $\lambda = 1.0\rho$. The KA is not valid in this case, therefore all results in this section is generated with the input parameter, KA = false, following the left-most route in Fig. 3.1 involving the MWR.

Fig. 4.11: Contour plot of relative electric field amplitude:

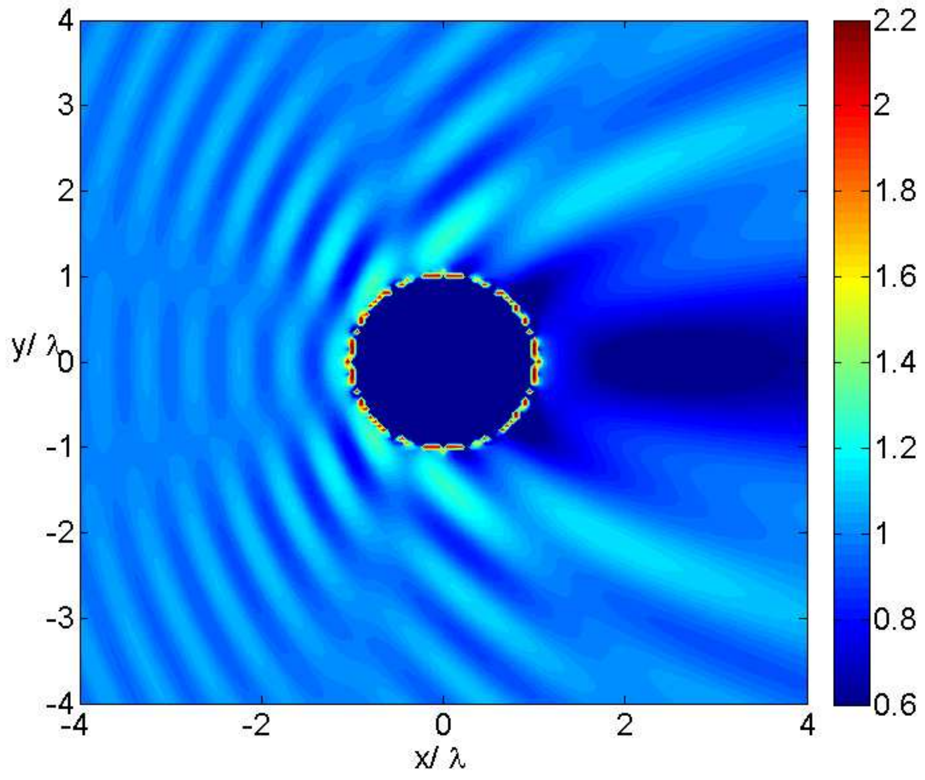

Figure 4.11: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, KA = false, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Figure 4.11 and 4.12 shows, as the contour plots of the previous section, an interference pattern around the scatterer similar to the one present in Fig. 3.7. The focusing effect seen in the previous section is however not present in these figures, which also needs to be checked with established theory and/or experimental results.

As in the previous section it is possible by visual inspection of Figs. 4.4 and 4.5 to find that the shortest peak-to-peak distances are again located

Fig. 4.12: Contour plot of relative electric field amplitude:

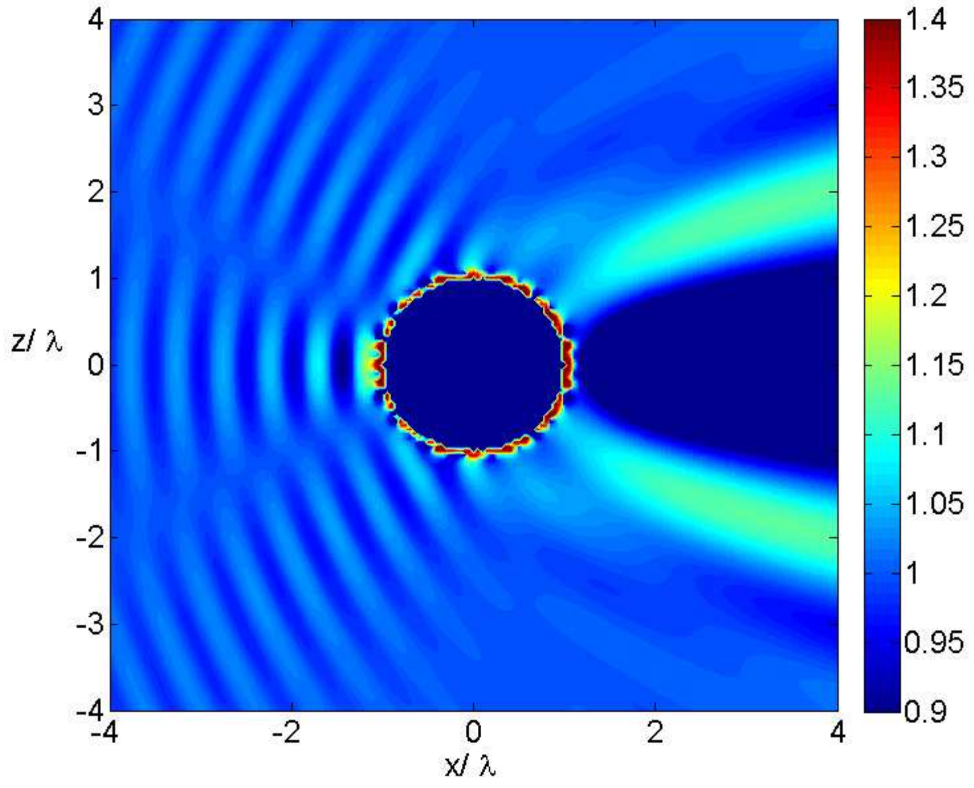

Figure 4.12: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $y = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $KA =$ false, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

directly in front of the scatterer. Here, it is possible in both figures to count six peaks in a distance corresponding to three times the wavelength, λ. This gives a peak-to-peak distance of,

$$d = \frac{3\lambda}{6} = \frac{\lambda}{2} = d^{\lambda}_{\mathrm{min}}, \tag{4.3}$$

which again is exactly the expected minimum theoretical peak-to-peak distance.

Figures 4.11, 4.12 and 4.13 indicates as in the previous section the expected mirror symmetries presented in Section 3.8.2 with the planes, $y = 0$ and $z = 0$, as symmetry planes. These symmetries are more accurately displayed by the Figs. 4.14 and 4.15. The mirror symmetry around $\theta = \pi$ in the figures translates to having the mentioned symmetries.

Fig. 4.13: Contour plot of relative electric field amplitude:

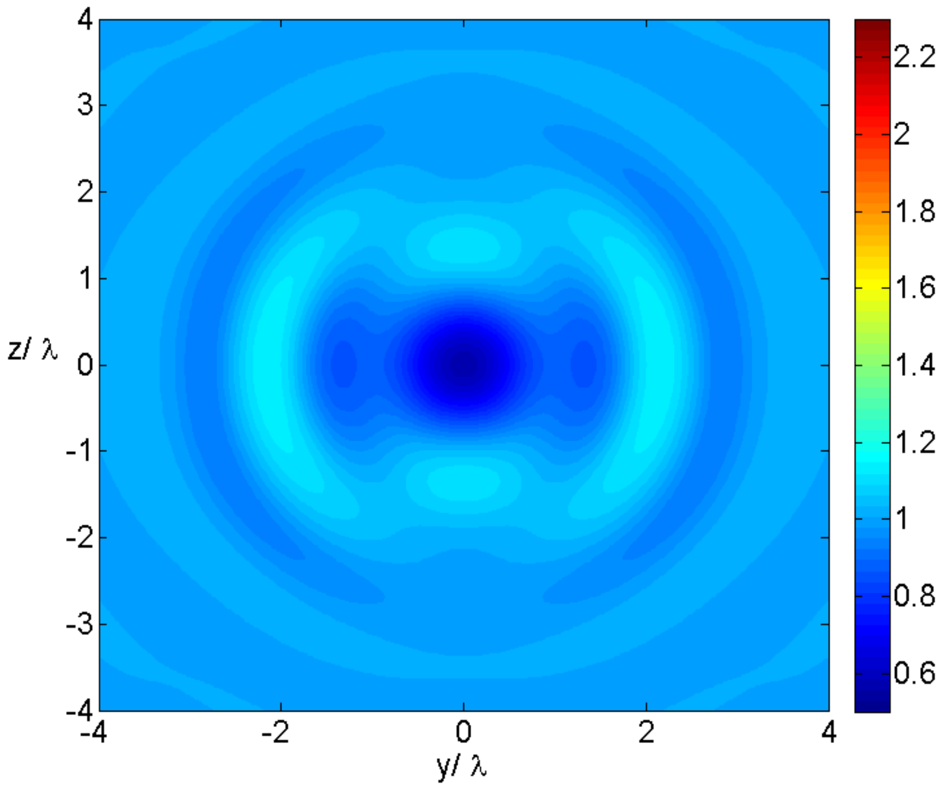

Figure 4.13: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, in the plane $x = 2$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

The rotational symmetry test which checks if the sphere "looks" the same upon rotation when the sensing wavelength is set to, $\lambda = 1.0\rho$ and the number of discretization points is set to $N_{\text{p}} = 4098$ is included below. Figures 4.16 and 4.17 corresponds to Figs. 4.9 and 4.10 of the previous section. The symmetry displayed by the figures are maybe not that surprising when the symmetry was present in the case of $\lambda = 0.1\rho$ and $N_{\text{p}} = 65538$. After all, Eq. (3.25) states that decreasing the ratio $a = \rho/\lambda$ by a factor, $f = 10$, allows for a decrease in N_{el} and thus N_{p} by a factor, $f = 10^2 = 100$. And the applied decrease factor was only $f = 65538/4098 \approx 16$.

Fig. 4.14: Mirror symmetry about the plane $y = 0$:

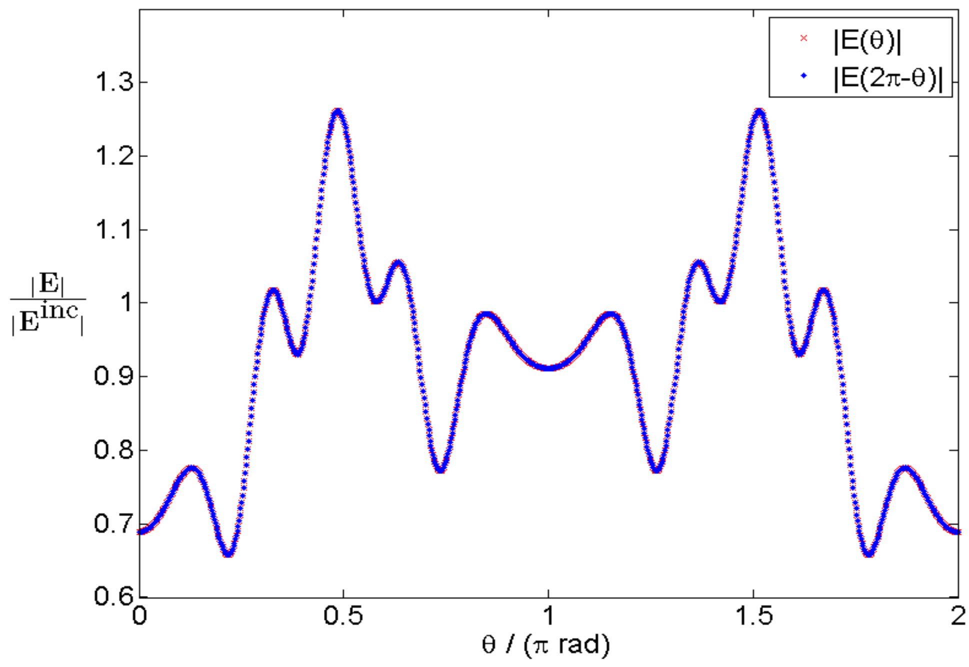

Figure 4.14: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 1.5\rho$, in the plane $z = 0$ centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\text{Err}^1 = 1.02 \cdot 10^{-5}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.15: Mirror symmetry about the plane $z = 0$:

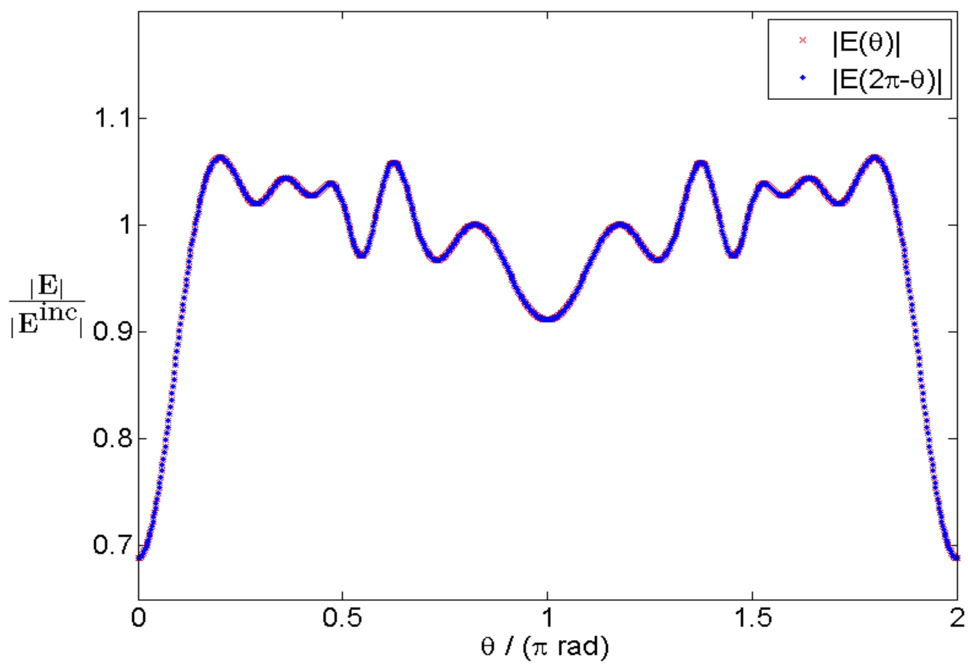

Figure 4.15: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 1.5\rho$, in the plane, $y = 0$, centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\text{Err}^1 = 9.49 \cdot 10^{-6}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.16: Contour plot of relative electric field amplitude:

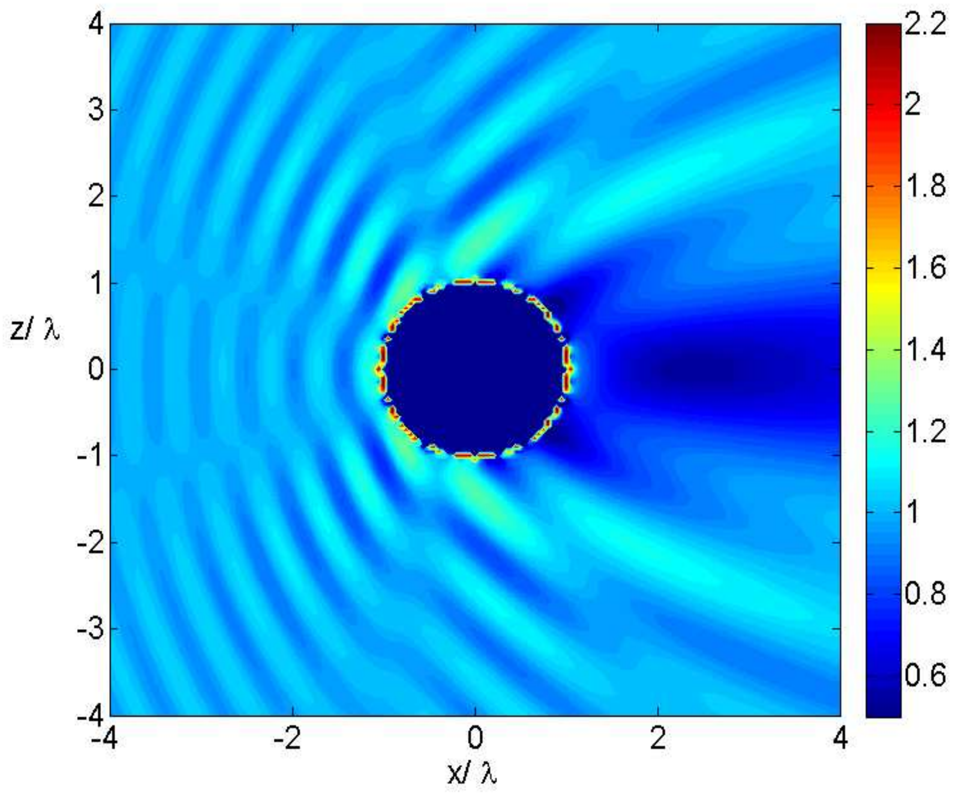

Figure 4.16: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $y = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $KA = \text{false}$, $\lambda = 1.0\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = -\hat{\mathbf{y}}$.

Fig. 4.17: Symmetry upon rotation around incoming wave vector axis:

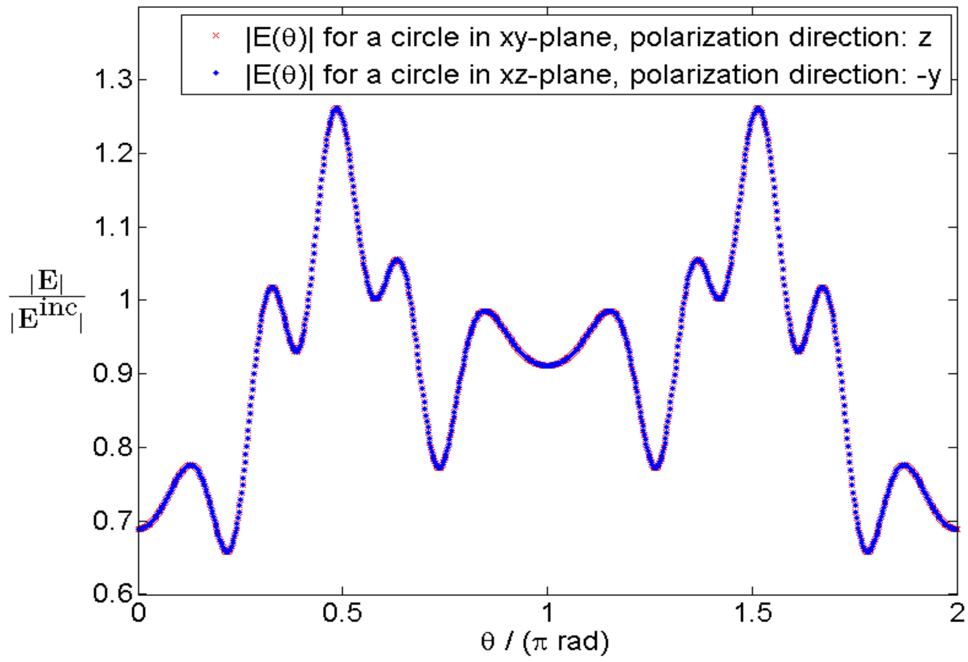

Figure 4.17: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 1.5\rho$, in the planes $z = 0$ and $y = 0$, centered at the origin. The starting point, $\theta = 0$, represents for both circles the point $[x,\ y,\ z] = [1,\ 0,\ 0]$.The polarization of the incoming EM wave was set to respectively $\hat{\mathbf{z}}$ and $-\hat{\mathbf{y}}$. The other input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$ and $\lambda = 1.0\rho$. The RMS of the relative error was found to be, $\text{Err}^1 = 1.04 \cdot 10^{-4}$.

4.4 Results for $\lambda \gg \rho$

The third problem studied involved the wavelength set equal to forty times the radius of the scatterer, $\lambda = 40.0\rho$. The KA is not valid in this case, therefore all results in this section are generated with the input parameter, $KA =$ false, following the left-most route in Fig. 3.1, involving the MWR. This is a case where the numerical results are, in addition to the symmetry and interference pattern considerations, comparable with the theory of the Rayleigh approximation presented in Section 3.8.4.

In the contour plots of this section the scatterer is located in the center of the figure, but only one of the points inside the sphere (having the value of zero) is included, namely at the origin, because of the resolution distance of $2\rho = \lambda/20$ between each point in the plot. This is sufficient for showing the field amplitude pattern around the sphere on a scale comparable with the wavelength but insufficient for showing the pattern close to the sphere, that is the near field behaviour.

Figure 4.18 shows the interference pattern around the sphere for $\lambda = 40.0\rho$. It is not meaningful to compare the pattern of this figure to the Fig. 3.7, because of the huge difference in radius-to-wavelength ratio. However the prediction concerning minimum peak-to-peak distance should still hold. And indeed by counting peaks on a range of four wavelengths directly in front of the sphere the number adds up to eight peaks. This gives a peak-to-peak distance of,

$$d = \frac{4\lambda}{8} = \frac{\lambda}{2} = d^{\lambda}_{\mathrm{min}}, \tag{4.4}$$

which again is exactly the expected minimum theoretical peak-to-peak distance.

The mirror symmetries around $y = 0$ and $z = 0$ in the total field amplitude are displayed by the Figs. 4.19 and 4.20. The mirror symmetry around $\theta = \pi$ in the figures translates as in the previous sections to having the mentioned mirror symmetry planes.

Fig. 4.18: Contour plot of relative electric field amplitude:

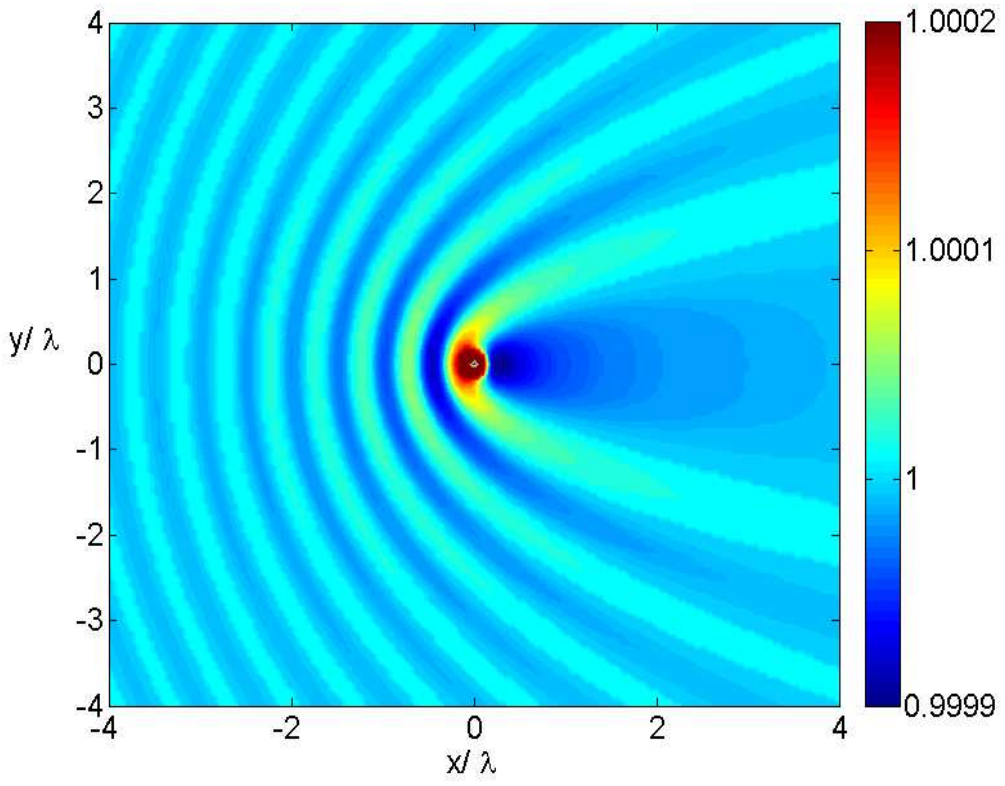

Figure 4.18: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $KA =$ false, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

Fig. 4.19: Mirror symmetry about the plane $y = 0$:

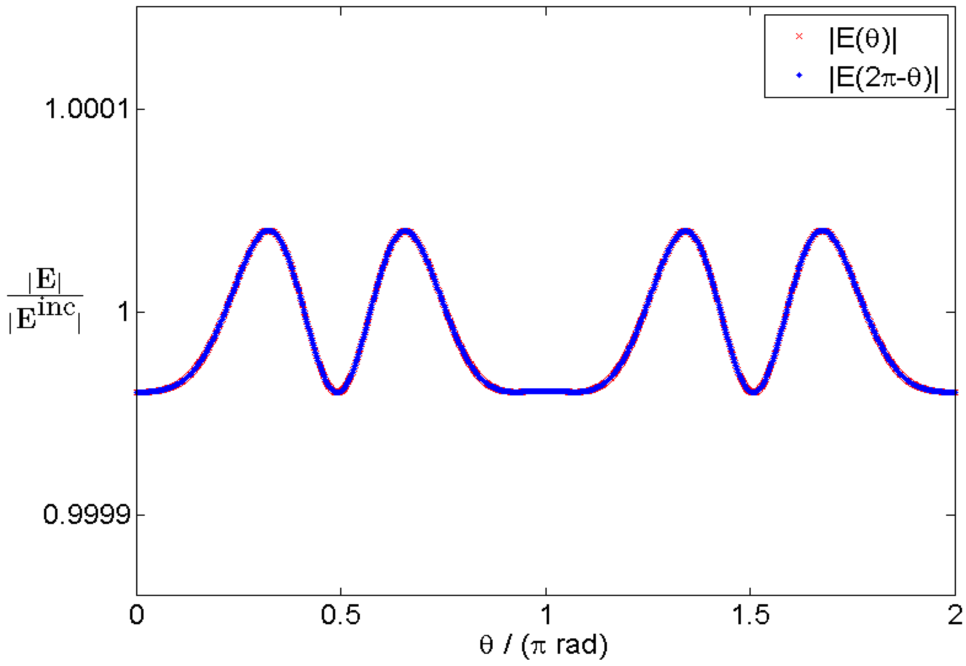

Figure 4.19: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 40.0\rho = \lambda$, in the plane $z = 0$ centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\text{Err}^1 = 1.88 \cdot 10^{-7}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, KA = false, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.20: Mirror symmetry about the plane $z = 0$:

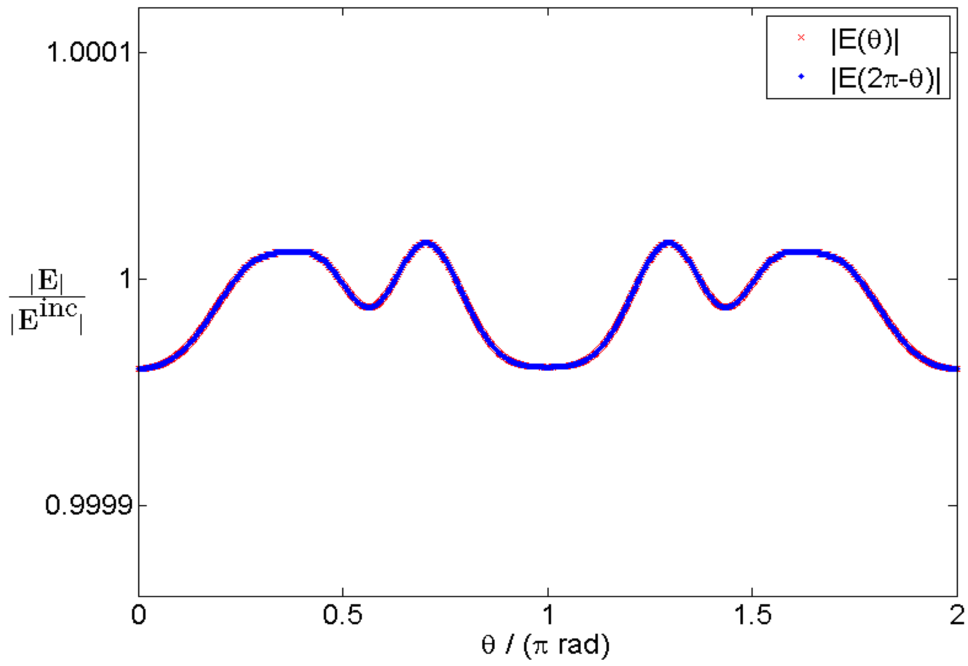

Figure 4.20: Plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, around a circle with radius, $\rho_{\text{circle}} = 40.0\rho = \lambda$, in the plane, $y = 0$, centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The RMS of the relative error was found to be, $\text{Err}^1 = 1.68{\cdot}10^{-7}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, KA = false, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Figures 4.21 and 4.22 shows the scattered electric far field amplitude relative to the incoming electric field amplitude. They correspond well with the theory of Section 3.8.4 with circles of equal value in the plane $z = 0$ and clearly reduced amplitude in the directions normal to the incoming wave vector, $\mathbf{k}$ in the plane $y = 0$. This is quantitatively displayed by Figs. 4.23 and 4.24. They show good correspondence with Eq. 3.43. In Fig. 4.23 the intensity component, I_l^{inc}, is zero leaving only the constant term, while in Fig. 4.24 the intensity component, I_r^{inc}, is zero leaving only the $\cos^2\theta$ term.

Figure 4.25 shows the calculated wavelength dependence of the scattered intensity. Regression analysis gave $a_{\text{fit}} = 0.01578$ and $b_{\text{fit}} = -4.037$ as the best fit to the graph, $a_{\text{fit}}\lambda^{b_{\text{fit}}}$. This differs from the theoretical exponent by,

$$\frac{b_{\text{theory}} - b_{\text{fit}}}{b_{\text{theory}}} = \frac{-4 - (-4.037)}{4.0} = 0.925\%. \tag{4.5}$$

Figure 4.26 shows how the scattered intensity depends on the distance from the scatterer and compares the numerical result with the Rayleigh scattering theory. The calculated intensity plot was off the theoretical curve by 0.34%, when applying the error formula of Eq. (3.44).

Fig. 4.21: Contour plot of relative electric field amplitude:

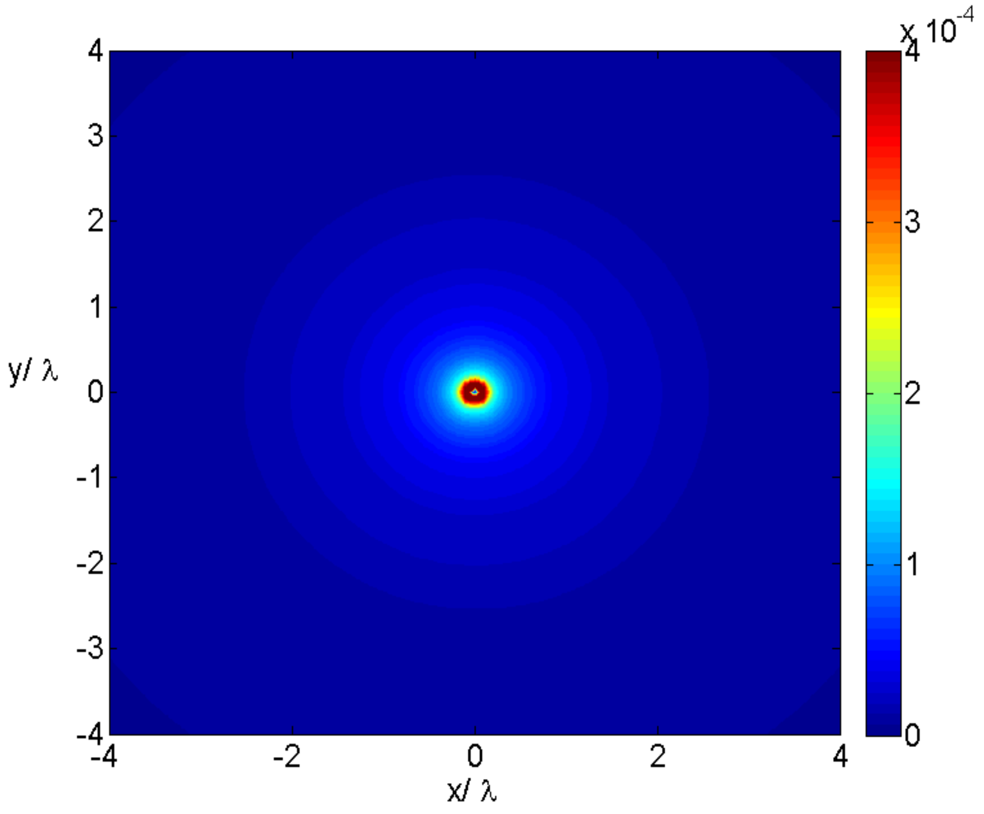

Figure 4.21: Contour plot of the relative scattered electric field amplitude, $|\mathbf{E}^{\text{sca}}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\text{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.22: Contour plot of relative electric field amplitude:

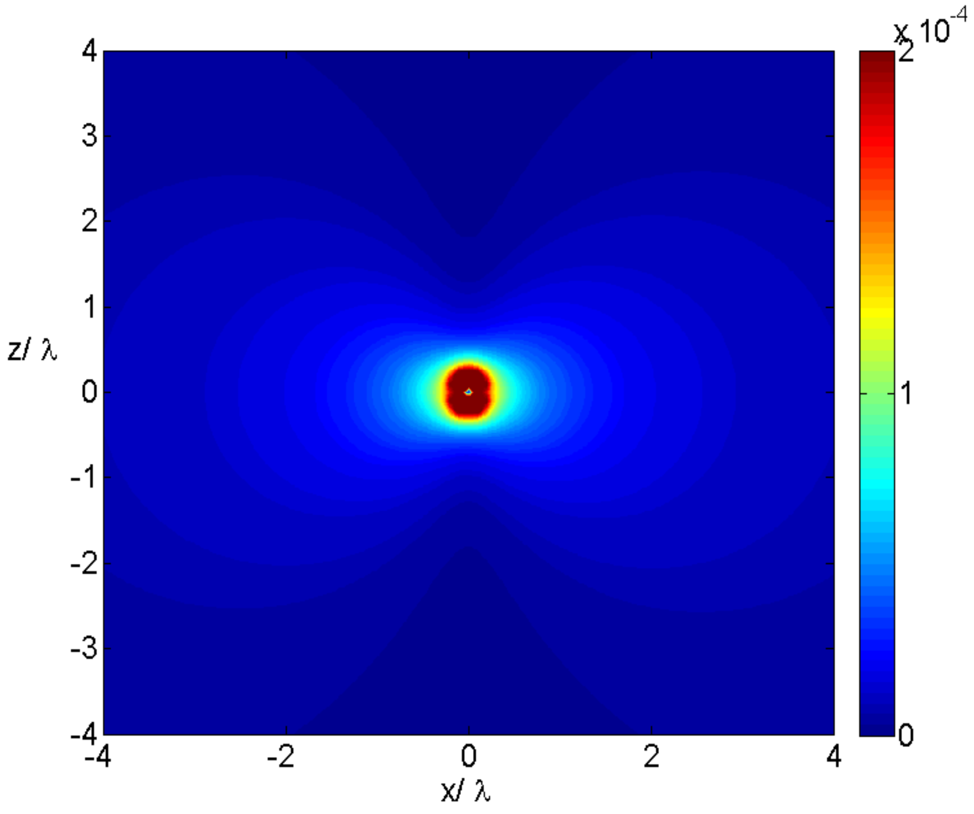

Figure 4.22: Contour plot of the relative scattered electric field amplitude, $|\mathbf{E}^{\mathrm{sca}}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $y = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $KA = $ false, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

Fig. 4.23: Angular dependence of scattered intensity at $z = 0$:

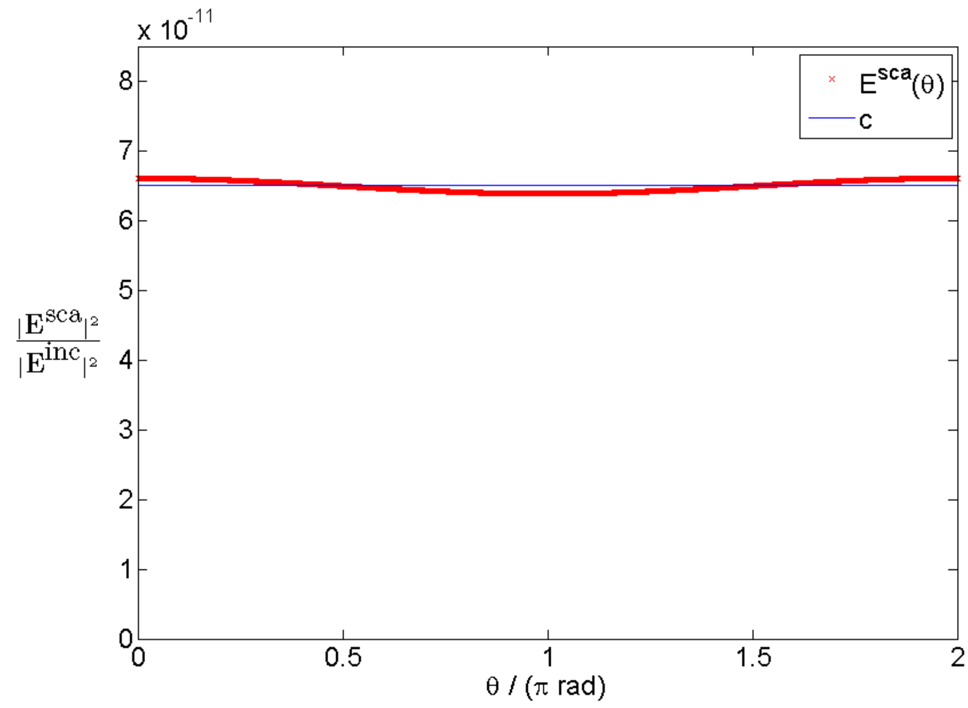

Figure 4.23: Plot of the squared relative scattered electric field amplitude , $|\mathbf{E}^{\text{sca}}(\mathbf{r})|^2/|\mathbf{E}(\mathbf{r})^{\text{inc}}|^2$, around a circle with radius, $\rho_{\text{circle}} = 200.0\rho = 5\lambda$, in the plane, $y = 0$, centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The theoretical reference from Eq. (3.43) is plotted with $c = 6.511 \cdot 10^{-11}$. The relative error was found to be, $\text{Err}^1 = 1.23 \cdot 10^{-2}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.24: Angular dependence of scattered intensity at $y = 0$:

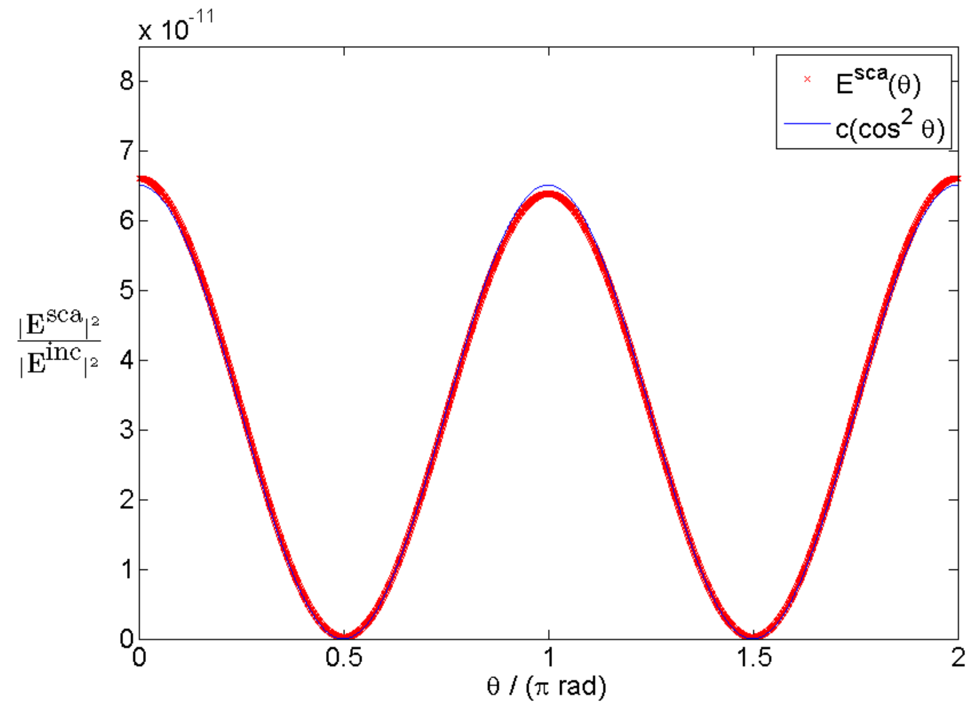

Figure 4.24: Plot of the squared relative scattered electric field amplitude, $|\mathbf{E}^{\text{sca}}(\mathbf{r})|^2/|\mathbf{E}(\mathbf{r})^{\text{inc}}|^2$, around a circle with radius, $\rho_{\text{circle}} = 200.0\rho = 5\lambda$, in the plane, $y = 0$, centered at the origin, where $\theta = 0$ represents the point $[x,\ y,\ z] = [1,\ 0,\ 0]$. The theoretical reference from Eq. (3.43) is plotted with $c = 6.511 \cdot 10^{-11}$. The relative error was found to be, $\text{Err}^2 = 2.10 \cdot 10^{-2}$. The input parameters were set to the following values, $N_{\text{p}} = 4098$, $KA = \text{false}$, $\lambda = 40.0\rho$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.25: Wavelength dependence of the intensity:

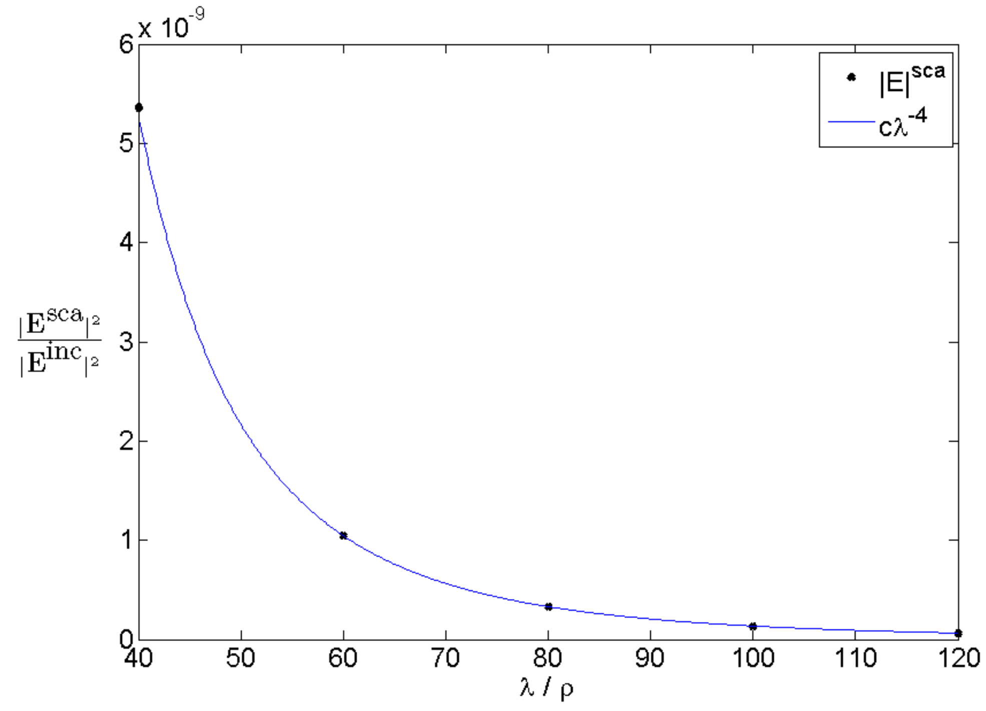

Figure 4.25: Plot of the squared relative scattered electric field amplitude, $|\mathbf{E}^{\text{sca}}(\mathbf{r})|^2/|\mathbf{E}(\mathbf{r})^{\text{inc}}|^2$, in the point, $[x, y, z] = [-100.0\rho,\ 0.0,\ 0.0]$ for a range of different wavelengths, λ. A graph proportional to the theoretical λ^{-4} dependence and proportionality constant, $c = 0.0135621216$, is also included. The relative error was found to be, $\text{Err}^1 = 1.78 \cdot 10^{-2}$. The input parameters were set to the following values, $N_{\text{p}} = 258$, $KA = \text{false}$ and $\hat{\mathbf{E}}^{\text{inc}} = \hat{\mathbf{z}}$.

Fig. 4.26: Wavelength dependence of the intensity:

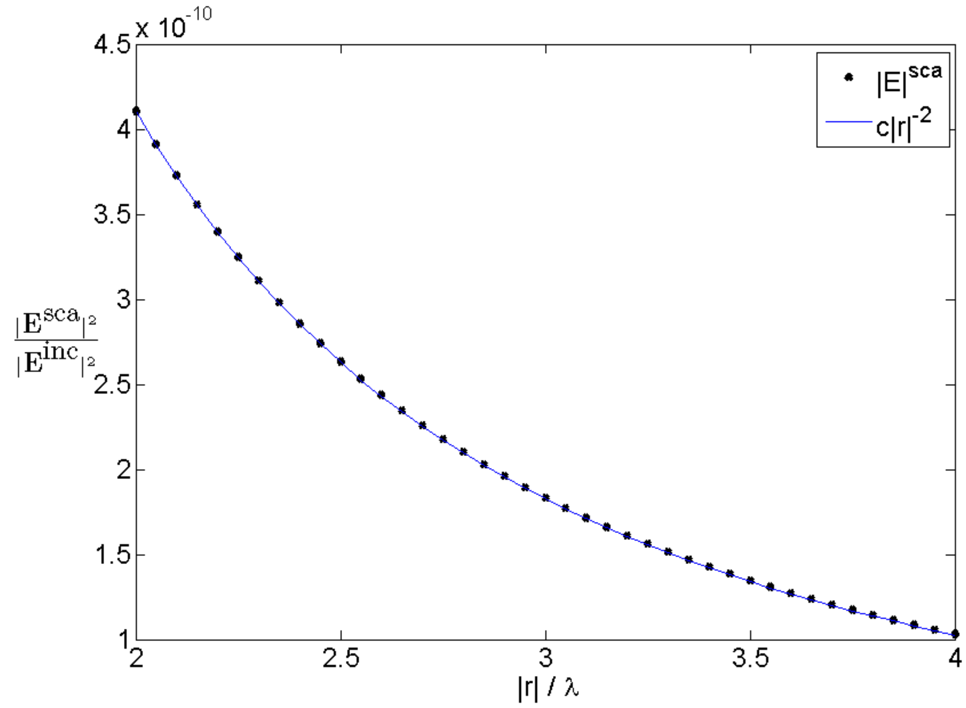

Figure 4.26: Plot of the squared relative scattered electric field amplitude, $|\mathbf{E}^{\mathrm{sca}}(\mathbf{r})|^2/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|^2$, for a range of different distances from the scatterer, $|\mathbf{r}|$. A graph proportional to the theoretical $|\mathbf{r}|^{-2}$ dependence and proportionality constant, $c = 1.643 \cdot 10^{-9}$, is also included. The relative error was found to be, $\mathrm{Err}^1 = 3.4 \cdot 10^{-3}$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $\lambda = 40.0\rho$, $KA =$ false and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

4.5 The MWR for Smaller Wavelengths

Ideally the results when using the MWR and the KA should be equal when applied to the same scattering problem and the wavelength is much smaller than the radius. This is however not easy to test thoroughly without access to the necessary equipment, namely a computer with enough memory. The problem is that the test must be performed in the domain of the KA requiring a lot of discretization elements which implies a huge matrix to be solved in the MWR taking up large amounts of memory. One way of solving this memory problem is to implement a Gaussian quadrature formula with more evaluation points per discretization element. This reduces the required number of elements and thus the number of matrix elements and the amount memory.

With the 3-point formula and the recursive discretization algorithm the case of $\lambda = \rho/3$ is close to the wavelength limit solvable by the implemented MWR with the equipment at hand. This gives the contour plot shown in Fig. 4.27 of the relative electric field. A positive sign is the observable focusing effect in the forward scattered field, which is a distinctive feature of the contour plots generated by the KA in Section 4.2.

This subject was not pursued further due to limited time.

Fig. 4.27: Contour plot of electric field amplitude using the MWR at a case of small wavelength:

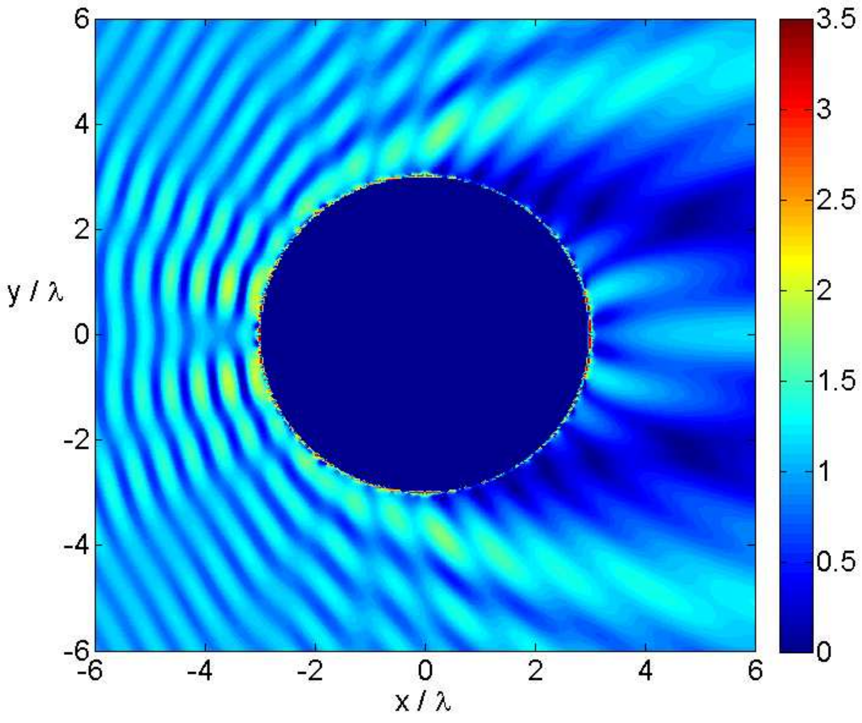

Figure 4.27: Contour plot of the relative electric field amplitude, $|\mathbf{E}(\mathbf{r})|/|\mathbf{E}(\mathbf{r})^{\mathrm{inc}}|$, in the plane $z = 0$. The input parameters were set to the following values, $N_{\mathrm{p}} = 4098$, $KA = \text{false}$, $\lambda = \rho/3$ and $\hat{\mathbf{E}}^{\mathrm{inc}} = \hat{\mathbf{z}}$.

4.6 Further Work

Both the implementation and the results produced are thoroughly tested through the testing procedures and evaluation of the results. But even though these tests show good agreement with the necessary requirements they are not conclusive. There are a series of other testable criteria. Testing the requirement of energy conservation as presented in Section 3.8.3 should certainly be of high priority. Moreover, testing how well the results fit with Mie theory would also be highly desirable, for instance by calculating the scattering cross section and comparing it with the theoretical predictions.

After testing the program against theoretical criteria there are many ways of improving the capabilities of the program. A simple, but powerful way of increasing the scope of the program mentioned in the previous section is to implement higher orders of the Gaussian quadrature method. This would reduce the required amount of memory allowing cases of shorter wavelengths. To increase the speed the recursive discretization method could be parallelized and also writing/reading the discretization to/from file could save some time.

A topic briefly touch upon in the thesis is the possibility of calculating the EM field close to the scatterer. Implementing the trick of splitting the numerical integrals into numerically solvable nonsingular terms and analytically solvable terms would increase the range of possible observation points to arbitrarily close to the scatterer. Another improvement would be to make the program capable of solving the scattering problem without the assumption of a perfectly conducting scatterer. This would involve having $\mathbf{M} \neq \mathbf{0}$, doubling the number of unknowns. Making the program capable of reproducing LSPR effects could be a future goal.

Chapter 5

Conclusion

Accurate numerical methods for calculating how an EM wave is scattered by an object has got important applications for instance in particle detection and material fabrication within the area of nanotechnology. Therefore there are many different available methods today and methods are continuously being developed and improved upon to meet the need for increased accuracy and flexibility. The main advantages of the SIE method studied in this thesis are having relatively low demands in computational time and memory compared with volume methods and the possibility of calculating the solution arbitrarily close to the scatterer. It involves the exploitation of a dyadic Green's function.

The method transforms the scattering problem into finding the imaginary electric and magnetic current densities, **J** and **M**, which induces the same physical effect as the scattering material. When these current densities are identified, by for instance the MWR or the KA, they are inserted into a definite integral. This integral over the boundary surface of the scatterer must be calculated numerically.

A FORTRAN program capable of solving the EM scattering problem for the special case of a perfectly conducting sphere is presented. It was designed as flexible as possible involving a custom data structure storing the discretization. This proved handy when extending the program to handle Gaussian quadrature formulas of multiple evaluation points and when the need for a new discretization algorithm appeared. The most time consuming parts of the program was parallelized using the OpenMP library significantly reducing the time consumption of the program.

The program was tested on the EM scattering problem involving a perfectly conducting spherical scatterer in vacuum. Both the KA and the MWR are implemented allowing a wide range of radius-to-wavelength ratios. Results for the cases of $\lambda \ll \rho$, $\lambda = \rho$ and $\lambda \gg \rho$ was produced and analyzed according to interference pattern and symmetry considerations and in the latter case the results were compared with Rayleigh theory. All criteria was met with small relative errors. The symmetry considerations gave relative errors in the order of magnitude $10^{-4} - 10^{-7}$. Some of the results were not significantly higher than expected only as a consequence of applying single precision. The results for the case of having long wavelength also corresponded well with the Rayleigh theory giving relative errors of 2.1% and less. A finer discretization mesh and observation points further away from the scatterer would probably result in even smaller errors.

The testing procedures and evaluation criteria analyzed strongly indicates that the program is capable of producing good results with both implemented methods for the tested cases of the EM scattering problem. There are however other testable criteria, as conservation of energy and Mie theory predictions, which could further strengthen or falsify this hypothesis. Improvements in the code, as higher Gaussian quadrature order, solving the case of a non-perfectly conducting scatterer involving $\mathbf{M} \neq \mathbf{0}$ and analytical solution of the singular integral terms in the Taylor expansion of the integrand, could further increase the capabilities of the program.

Appendix A

Fourier Transform

A.1 Definition

When a function $\mathbf{U}(\mathbf{r}, t)$ is piecewise continuous on every finite interval and absolutely integrable on the t-axis the following transformation, called the Fourier transform in the time domain, is allowed,

$$\tilde{\mathbf{U}}(\mathbf{r}, \omega) = \mathcal{F}\{\mathbf{U}(\mathbf{r}, t)\} = \int_{-\infty}^{\infty} dt \mathbf{U}(\mathbf{r}, t) e^{i\omega t}. \tag{A-1}$$

The inverse transformation is then given by,

$$\mathbf{U}(\mathbf{r}, t) = \mathcal{F}^{-1}\{\tilde{\mathbf{U}}(\mathbf{r}, \omega)\} = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} \tilde{\mathbf{U}}(\mathbf{r}, \omega) e^{-i\omega t}. \tag{A-2}$$

A.2 Fourier Transform of Time Derivative

Definition (A-2) gives,

$$\frac{\partial}{\partial t}\mathbf{U}(\mathbf{r}, t) = \frac{\partial}{\partial t}\int_{-\infty}^{\infty} \frac{d\omega}{2\pi} \tilde{\mathbf{U}}(\mathbf{r}, \omega) e^{-i\omega t} = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi}(-i\omega) \tilde{\mathbf{U}}(\mathbf{r}, \omega) e^{-i\omega t}, \tag{A-3}$$

which can be written as

$$\frac{\partial}{\partial t}\mathbf{U}(\mathbf{r}, t) = \mathcal{F}^{-1}\{(-i\omega)\tilde{\mathbf{U}}(\mathbf{r}, \omega)\}. \tag{A-4}$$

Performing the Fourier transform, (A-1), on both sides of Eq. (A-4) now gives the following expression for the Fourier transform of the time derivative of a function,

$$\mathcal{F}\{\frac{\partial}{\partial t}\mathbf{U}(\mathbf{r}, t)\} = \mathcal{F}\{\mathcal{F}^{-1}\{(-i\omega)\tilde{\mathbf{U}}(\mathbf{r}, \omega)\}\} = -i\omega\tilde{\mathbf{U}}(\mathbf{r}, \omega). \tag{A-5}$$

Appendix B

Dyadic Tensor

This section introduces the dyadic tensor, its most relevant properties for the current thesis and a useful simplified notation convention.

B.1 Definition

A dyadic tensor, $\bar{\mathbf{T}}$, is a tensor of order two, which is defined by the sum

$$\bar{\mathbf{T}} = \sum_{i=1}^{N}\sum_{j=1}^{N} T_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j), \tag{B-1}$$

where N is the dimension of the space, $\hat{\mathbf{x}}_i$ is the unit basis vector in the i'th direction and $\otimes$, denote the tensor product. By representing the tensor product, $\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j$, by a basis matrix where the only non-zero element is in row i and column j with the value 1, the dyad $\bar{\mathbf{T}}$ with dimension, N=3, may be displayed in the following way,

$$\begin{aligned}\bar{\mathbf{T}} =& T_{11}\begin{bmatrix}1&0&0\\0&0&0\\0&0&0\end{bmatrix} + T_{12}\begin{bmatrix}0&1&0\\0&0&0\\0&0&0\end{bmatrix} + ... + T_{33}\begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix}\\ =& \begin{bmatrix}T_{11}&T_{13}&T_{13}\\T_{21}&T_{22}&T_{23}\\T_{31}&T_{32}&T_{33}\end{bmatrix}.\end{aligned} \tag{B-2}$$

B.2 Notation

The sum (B-1) is often written in index form as

$$\bar{\mathbf{T}} = T_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j), \tag{B-3}$$

omitting the summation symbols.

B.3 Properties

This section presents important dyadic tensor properties.

B.3.1 Basic Identities

One may define operators acting on two arguments, where one argument is a vector and the other is a dyadic tensor. The two most important of these operators are analogous with the vector cross product, $\times$, and the vector dot product, $\cdot$, denoted with the same symbols, however acting on different type of arguments. These operators are defined in the following way by the well known vector operators (see Ref. [16]),

$$\hat{\mathbf{x}}_i \cdot (\hat{\mathbf{x}}_j \otimes \hat{\mathbf{x}}_k) = (\hat{\mathbf{x}}_i \cdot \hat{\mathbf{x}}_j)\hat{\mathbf{x}}_k \tag{B-4}$$

$$(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) \cdot \hat{\mathbf{x}}_k = \hat{\mathbf{x}}_i(\hat{\mathbf{x}}_j \cdot \hat{\mathbf{x}}_k) \tag{B-5}$$

$$\hat{\mathbf{x}}_i \times (\hat{\mathbf{x}}_j \otimes \hat{\mathbf{x}}_k) = (\hat{\mathbf{x}}_i \times \hat{\mathbf{x}}_j) \otimes \hat{\mathbf{x}}_k \tag{B-6}$$

$$(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) \times \hat{\mathbf{x}}_k = \hat{\mathbf{x}}_i \otimes (\hat{\mathbf{x}}_j \times \hat{\mathbf{x}}_k) \tag{B-7}$$

$$(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) \cdot (\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) = (\hat{\mathbf{x}}_j \cdot \hat{\mathbf{x}}_k)(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_l), \tag{B-8}$$

where $\hat{\mathbf{x}}_i$, $\hat{\mathbf{x}}_j$, $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_l$ are arbitrary unit vectors in three dimensions.

B.3.2 Derivations of Vector-Dyad Operation Results

When the identities of the previous section are established for arbitrary unit vectors it is possible to derive expressions for the elements of $\mathbf{c} = \mathbf{b} \cdot \bar{\mathbf{A}}$, $\mathbf{d} = \bar{\mathbf{A}} \cdot \mathbf{b}$, $\bar{\mathbf{C}} = \mathbf{b} \times \bar{\mathbf{A}}$, $\bar{\mathbf{D}} = \bar{\mathbf{A}} \times \mathbf{b}$, and $\bar{\mathbf{E}} = \bar{\mathbf{A}} \cdot \bar{\mathbf{B}}$, where $\bar{\mathbf{A}}$, $\bar{\mathbf{B}}$, $\bar{\mathbf{C}}$, $\bar{\mathbf{D}}$ and $\bar{\mathbf{E}}$ are dyads and $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ are vectors.

Starting with $\mathbf{c} = \mathbf{b} \cdot \bar{\mathbf{A}}$, we have in index notation,

$$\mathbf{c} = c_i\hat{\mathbf{x}}_i = b_j\hat{\mathbf{x}}_j \cdot A_{kl}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l), \tag{B-9}$$

where $\hat{\mathbf{x}}_i$, $\hat{\mathbf{x}}_j$, $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_l$ are unit basis vectors in three dimensions. Collecting the scalars, b_j and A_{kl}, using property (B-4) and recognizing the relation, $(\hat{\mathbf{x}}_j \cdot \hat{\mathbf{x}}_k) = \delta_{jk}$, when $\mathbf{x}_j$ and $\hat{\mathbf{x}}_k$ are unit basis vectors gives,

$$\begin{aligned} \mathbf{c} &= A_{kl}b_j\hat{\mathbf{x}}_j \cdot (\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) = A_{kl}b_j(\hat{\mathbf{x}}_j \cdot \hat{\mathbf{x}}_k)\hat{\mathbf{x}}_l \\ &= A_{kl}b_j(\delta_{jk})\hat{\mathbf{x}}_l = A_{jl}b_j\hat{\mathbf{x}}_l = c_i\hat{\mathbf{x}}_i. \end{aligned} \tag{B-10}$$

The last equality implies $l = i$, leading to

$$\mathbf{c} = c_i\hat{\mathbf{x}}_i = A_{ji}b_j\hat{\mathbf{x}}_i, \tag{B-11}$$

showing that the result of taking the dot product analogue of a vector, $\mathbf{b}$, with a tensor, $\bar{\mathbf{A}}$ from the left, is a new vector, $\mathbf{c}$, where the i'th component is the dot product of a vector consisting of the elements of the i'th column of the tensor $\bar{\mathbf{A}}$ with the vector, $\mathbf{b}$.

Performing the corresponding analysis for the operation of multiplying a vector, $\mathbf{b}$, with a dyad, $\bar{\mathbf{A}}$ from the right, $\mathbf{d} = \bar{\mathbf{A}} \cdot \mathbf{b}$, by the dot product analogue gives

$$\begin{aligned} \mathbf{d} = d_i\hat{\mathbf{x}}_i = \bar{\mathbf{A}} \cdot \mathbf{b} &= A_{kl}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \cdot b_j\hat{\mathbf{x}}_j \\ &= A_{kl}b_j(\hat{\mathbf{x}}_l \cdot \hat{\mathbf{x}}_j)\hat{\mathbf{x}}_k = A_{kl}b_j\delta_{lj}\hat{\mathbf{x}}_k \\ &= A_{ij}b_j\hat{\mathbf{x}}_i, \end{aligned} \tag{B-12}$$

where property (B-5) was used in the derivation. The result shows that taking the dot product analogue of a vector, $\mathbf{b}$, with a dyad, $\bar{\mathbf{A}}$ from the right, results in a new vector, $\mathbf{d}$, where the i'th component is the dot product of a vector consisting of the elements of the i'th row of the tensor $\bar{\mathbf{A}}$ with the vector, $\mathbf{b}$.

The derivation for the cross product analogue, $\bar{\mathbf{C}} = \mathbf{b} \times \bar{\mathbf{A}}$, is similar. Using property (B-6) gives,

$$\begin{aligned} \bar{\mathbf{C}} = C_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) &= b_k\hat{\mathbf{x}}_k \times A_{lm}(\hat{\mathbf{x}}_l \otimes \hat{\mathbf{x}}_m) \\ &= A_{lm}b_k\hat{\mathbf{x}}_k \times (\hat{\mathbf{x}}_l \otimes \hat{\mathbf{x}}_m) = A_{lm}b_k(\hat{\mathbf{x}}_k \times \hat{\mathbf{x}}_l) \otimes \hat{\mathbf{x}}_m \\ &= A_{lm}b_k\varepsilon_{nkl}(\hat{\mathbf{x}}_n \otimes \hat{\mathbf{x}}_m), \end{aligned} \tag{B-13}$$

where ε_{nkl} is the Levi-Civita tensor of order three, which contains the following values,

$$\varepsilon_{nkl} = \begin{cases} +1, & \text{if } \{n,k,l\} = \{1,2,3\}, \{3,1,2\} \text{ or } \{2,3,1\} \\ -1, & \text{if } \{n,k,l\} = \{3,2,1\}, \{2,1,3\} \text{ or } \{1,3,2\} \\ 0, & \text{otherwise} \end{cases} . \tag{B-14}$$

Again, comparing the first and last line of equation (B-13) shows that $n = i$ and $m = j$ giving

$$\bar{\mathbf{C}} = C_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) = \varepsilon_{ikl}b_kA_{lj}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j). \tag{B-15}$$

A vector, $\mathbf{v} = v_i\hat{\mathbf{x}}_i$, where the vector elements, v_i, are given by $v_i = \varepsilon_{ikl}a_kb_l$, represents the cross product of the vector, $\mathbf{a} = a_i\hat{\mathbf{x}}_i$, with the vector, $\mathbf{b} = b_i\hat{\mathbf{x}}_i$, $\mathbf{v} = \mathbf{a} \times \mathbf{b}$. With this in mind, equation (B-15) clearly shows that the cross product of a vector, $\mathbf{b}$, with a dyad, $\bar{\mathbf{A}}$, from the left results in a new dyad, $\bar{\mathbf{C}}$, where the elements of the j'th column in $\bar{\mathbf{C}}$ are identical to the elements of the cross product of the vector, $\mathbf{b}$ with a vector containing the same elements as the jth column of the dyad $\bar{\mathbf{A}}$.

Furthermore, the cross product analogue of a vector, $\mathbf{b}$, with a dyad, $\bar{\mathbf{A}}$, from the right results in a new dyad $\bar{\mathbf{D}} = \bar{\mathbf{A}} \times \mathbf{b}$, given by,

$$\begin{aligned}\bar{\mathbf{D}} &= D_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) = A_{lm}(\hat{\mathbf{x}}_l \otimes \hat{\mathbf{x}}_m) \times b_k\hat{\mathbf{x}}_k \\ &= A_{lm}b_k(\hat{\mathbf{x}}_l \otimes \hat{\mathbf{x}}_m) \times \hat{\mathbf{x}}_k = A_{lm}b_k\hat{\mathbf{x}}_l \otimes (\hat{\mathbf{x}}_m \times \hat{\mathbf{x}}_k) \\ &= A_{lm}b_k\varepsilon_{nmk}(\hat{\mathbf{x}}_l \otimes \hat{\mathbf{x}}_n) = \varepsilon_{jmk}A_{im}b_k(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j),\end{aligned} \tag{B-16}$$

where Eq. (B-7) was used in the second line. The relations, $l = i$ and $n = j$, was found by comparing line 1 with the first expression of line 3. With the above mentioned element representation of the vector cross product in mind, Eq. (B-16) clearly shows that the cross product of a vector, $\mathbf{b}$, with a dyad $\bar{\mathbf{A}}$, from the right, results in a new dyad, $\bar{\mathbf{D}}$, where the elements of the i'th row in $\bar{\mathbf{D}}$ are identical to the elements of the cross product of a vector containing the same elements as the i'th row of the dyad $\bar{\mathbf{A}}$ with the vector, $\mathbf{b}$.

Finally, the dot product of a dyad, $\bar{\mathbf{A}}$, with another dyad, $\bar{\mathbf{B}}$, results in a new dyad $\bar{\mathbf{E}}$. Its elements may be expressed in the following way,

$$\begin{aligned}\bar{\mathbf{E}} &= E_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j) = A_{kl}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \cdot B_{mn}(\hat{\mathbf{x}}_m \otimes \hat{\mathbf{x}}_n) \\ &= A_{kl}B_{mn}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \cdot (\hat{\mathbf{x}}_m \otimes \hat{\mathbf{x}}_n) = A_{kl}B_{mn}(\hat{\mathbf{x}}_l \cdot \hat{\mathbf{x}}_m)(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_n) \\ &= A_{kl}B_{mn}\delta_{lm}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_n) = A_{kl}B_{ln}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_n) \\ &= A_{il}B_{lj}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j).\end{aligned} \tag{B-17}$$

Element E_{ij} of the resulting tensor is therefore identical to the result of a dot product of a vector, $\mathbf{a}$, containing the same elements as the i'th row of the dyad $\bar{\mathbf{A}}$, with a vector $\mathbf{b}$, containing the same elements as the j'th column of the dyad, $\bar{\mathbf{B}}$. In matrix representation, the result therefore corresponds to regular matrix multiplication.

B.3.3 The Identity Dyad

The identity dyad, denoted as $\bar{\mathbf{1}}$, is given by

$$\bar{\mathbf{1}} = \delta_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j), \tag{B-18}$$

where δ_{ij} is the Kroenecker-delta. It satisfies the following properties

$$\bar{\mathbf{1}} \cdot \bar{\mathbf{T}} = \bar{\mathbf{T}}, \tag{B-19}$$

$$\bar{\mathbf{T}} \cdot \bar{\mathbf{1}} = \bar{\mathbf{T}}, \tag{B-20}$$

$$\bar{\mathbf{1}} \cdot \mathbf{a} = \mathbf{a}, \tag{B-21}$$

$$\mathbf{a} \cdot \bar{\mathbf{1}} = \mathbf{a}, \tag{B-22}$$

where $\bar{\mathbf{T}}$ is any dyad and $\mathbf{a}$ is any vector. These properties are easily verified by inserting the definition (B-18) into each of the four properties listed and using the results from the previous section.

B.3.4 A Useful Relationship

This section seeks to prove the following relationship,

$$(\mathbf{a} \times \mathbf{b}) \cdot \bar{\mathbf{C}} = \mathbf{a} \cdot (\mathbf{b} \times \bar{\mathbf{C}}) = -\mathbf{b} \cdot (\mathbf{a} \times \bar{\mathbf{C}}), \tag{B-23}$$

where $\mathbf{a}$ and $\mathbf{b}$ are vectors and $\bar{\mathbf{C}}$ is a dyadic tensor.

The first term of equation (B-23) may be written out in index notation as

$$\begin{aligned}
(\mathbf{a} \times \mathbf{b}) \cdot \bar{\mathbf{C}} &= (a_i\hat{\mathbf{x}}_i \times b_j\hat{\mathbf{x}}_j) \cdot C_{kl}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \\
&= a_i b_j C_{kl}(\hat{\mathbf{x}}_i \times \hat{\mathbf{x}}_j) \cdot (\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \\
&= a_i b_j C_{kl}\varepsilon_{mij}\hat{\mathbf{x}}_m \cdot (\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l) \\
&= a_i b_j C_{kl}\varepsilon_{mij}(\hat{\mathbf{x}}_m \cdot \hat{\mathbf{x}}_k)\hat{\mathbf{x}}_l \\
&= a_i b_j C_{kl}\varepsilon_{mij}\delta_{mk}\hat{\mathbf{x}}_l \\
&= a_i b_j C_{kl}\varepsilon_{kij}\hat{\mathbf{x}}_l,
\end{aligned} \tag{B-24}$$

where equation (B-4) and the orthogonality of the basis vectors were used. The next term can similarly be written as

$$\begin{aligned}
\mathbf{a} \cdot (\mathbf{b} \times \bar{\mathbf{C}}) &= a_i\hat{\mathbf{x}}_i \cdot (b_j\hat{\mathbf{x}}_j \times C_{kl}(\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l)) \\
&= a_i b_j C_{kl}\hat{\mathbf{x}}_i \cdot (\hat{\mathbf{x}}_j \times (\hat{\mathbf{x}}_k \otimes \hat{\mathbf{x}}_l)) \\
&= a_i b_j C_{kl}\hat{\mathbf{x}}_i \cdot ((\hat{\mathbf{x}}_j \times \hat{\mathbf{x}}_k) \otimes \hat{\mathbf{x}}_l) \\
&= a_i b_j C_{kl}\varepsilon_{mjk}\hat{\mathbf{x}}_i \cdot (\hat{\mathbf{x}}_m \otimes \hat{\mathbf{x}}_l) \\
&= a_i b_j C_{kl}\varepsilon_{mjk}(\hat{\mathbf{x}}_i \cdot \hat{\mathbf{x}}_m)\hat{\mathbf{x}}_l \\
&= a_i b_j C_{kl}\varepsilon_{mjk}\delta_{im}\hat{\mathbf{x}}_l \\
&= a_i b_j C_{kl}\varepsilon_{ijk}\hat{\mathbf{x}}_l,
\end{aligned} \tag{B-25}$$

where the equations (B-4) and (B-6) and the orthogonality of the basis vectors were used. Finally, the last term may be written as

$$\begin{aligned}
-\mathbf{b}\cdot(\mathbf{a}\times\bar{\mathbf{C}}) &= -b_j\hat{\mathbf{x}}_j\cdot(a_i\hat{\mathbf{x}}_i\times C_{kl}(\hat{\mathbf{x}}_k\otimes\hat{\mathbf{x}}_l))\\
&= -a_ib_jC_{kl}\hat{\mathbf{x}}_j\cdot(\hat{\mathbf{x}}_i\times(\hat{\mathbf{x}}_k\otimes\hat{\mathbf{x}}_l))\\
&= -a_ib_jC_{kl}\hat{\mathbf{x}}_j\cdot((\hat{\mathbf{x}}_i\times\hat{\mathbf{x}}_k)\otimes\hat{\mathbf{x}}_l)\\
&= -a_ib_jC_{kl}\varepsilon_{mik}\hat{\mathbf{x}}_j\cdot(\hat{\mathbf{x}}_m\otimes\hat{\mathbf{x}}_l)\\
&= -a_ib_jC_{kl}\varepsilon_{mik}(\hat{\mathbf{x}}_j\cdot\hat{\mathbf{x}}_m)\hat{\mathbf{x}}_l\\
&= -a_ib_jC_{kl}\varepsilon_{mik}\delta_{jm}\hat{\mathbf{x}}_l\\
&= -a_ib_jC_{kl}\varepsilon_{jik}\hat{\mathbf{x}}_l,
\end{aligned} \tag{B-26}$$

where the same equations as above were used. From the definition of the Levi-Civita symbol in appendix B.3.2 it is clear that the following property is satisfied,

$$\varepsilon_{ijk}=\varepsilon_{jki}=\varepsilon_{kij}=-\varepsilon_{ikj}=-\varepsilon_{kji}=-\varepsilon_{jik}. \tag{B-27}$$

The fact that the first, third and sixth term of this equation are equal proves equation (B-23). Q.E.D.

B.3.5 A Second Useful Relationship

In this section the following relation regarding a dyad, $\bar{\mathbf{B}}$, and two vectors, **a** and **c**, is shown to be correct,

$$(\mathbf{a}\times\bar{\mathbf{B}})\cdot\mathbf{c}=\mathbf{a}\times(\bar{\mathbf{B}}\cdot\mathbf{c}). \tag{B-28}$$

The expression on the left-hand side may be written in index notation in the following way,

$$\begin{aligned}
(\mathbf{a}\times\bar{\mathbf{B}})\cdot\mathbf{c} &= (a_i\hat{\mathbf{x}}_i\times B_{jk}(\hat{\mathbf{x}}_j\otimes\hat{\mathbf{x}}_k))\cdot c_l\hat{\mathbf{x}}_l\\
&= a_iB_{jk}c_l(\hat{\mathbf{x}}_i\times(\hat{\mathbf{x}}_j\otimes\hat{\mathbf{x}}_k))\cdot\hat{\mathbf{x}}_l\\
&= a_iB_{jk}c_l((\hat{\mathbf{x}}_i\times\hat{\mathbf{x}}_j)\otimes\hat{\mathbf{x}}_k)\cdot\hat{\mathbf{x}}_l\\
&= a_iB_{jk}c_l\varepsilon_{nij}(\hat{\mathbf{x}}_n\otimes\hat{\mathbf{x}}_k)\cdot\hat{\mathbf{x}}_l\\
&= a_iB_{jk}c_l\varepsilon_{nij}\delta_{kl}\hat{\mathbf{x}}_n\\
&= a_iB_{jk}c_k\varepsilon_{nij}\hat{\mathbf{x}}_n.
\end{aligned} \tag{B-29}$$

In the same way the expression on the right-hand side may be written in the following way,

$$\begin{aligned}\mathbf{a}\times(\bar{\mathbf{B}}\cdot\mathbf{c}) &= a_i\hat{\mathbf{x}}_i\times\left(B_{jk}(\hat{\mathbf{x}}_j\otimes\hat{\mathbf{x}}_k)\cdot c_l\hat{\mathbf{x}}_l\right)\\ &= a_iB_{jk}c_l\hat{\mathbf{x}}_i\times\left((\hat{\mathbf{x}}_j\otimes\hat{\mathbf{x}}_k)\cdot\hat{\mathbf{x}}_l\right)\\ &= a_iB_{jk}c_l\hat{\mathbf{x}}_i\times\left((\hat{\mathbf{x}}_j(\hat{\mathbf{x}}_k\cdot\hat{\mathbf{x}}_l)\right)\\ &= a_iB_{jk}c_l\hat{\mathbf{x}}_i\times\left(\hat{\mathbf{x}}_j\delta_{kl}\right)\\ &= a_iB_{jk}c_k\hat{\mathbf{x}}_i\times\hat{\mathbf{x}}_j\\ &= a_iB_{jk}c_k\varepsilon_{nij}\hat{\mathbf{x}}_n.\end{aligned} \tag{B-30}$$

In both derivations the basic identities of appendix B.3.1 were used. The final expressions are identical thus proving Eq. (B-28).

B.3.6 The Transpose of a Dyadic Tensor

The transpose, $\bar{\mathbf{A}}^T$, of a dyad, $\bar{\mathbf{A}} = A_{ij}(\hat{\mathbf{x}}_i\otimes\hat{\mathbf{x}}_j)$, is defined as

$$\bar{\mathbf{A}}^T = [A_{ij}(\hat{\mathbf{x}}_i\otimes\hat{\mathbf{x}}_j)]^T = A_{ji}(\hat{\mathbf{x}}_i\otimes\hat{\mathbf{x}}_j). \tag{B-31}$$

Multiplying a vector, $\mathbf{a}$, with the transpose of the dyad, $\bar{\mathbf{A}}$, from the left, gives by using equation (B-11)

$$\mathbf{a}\cdot\bar{\mathbf{A}}^T = a_i\hat{\mathbf{x}}_i\cdot A_{jk}(\hat{\mathbf{x}}_k\otimes\hat{\mathbf{x}}_j) = A_{ij}b_j\hat{\mathbf{x}}_i. \tag{B-32}$$

Comparing this result with equation (B-12) makes it clear that the following equation is satisfied,

$$\mathbf{a}\cdot\bar{\mathbf{A}}^T = \bar{\mathbf{A}}\cdot\mathbf{a}. \tag{B-33}$$

Appendix C

Green's Function

This section covers the general idea behind the Green's function and the properties of the specific Green's function relevant for the main text.

C.1 General Idea of the Green's Function

When having a linear differential equation given by

$$L\phi(\mathbf{r}) = f(\mathbf{r}), \tag{C-1}$$

where L is a linear differential operator in three dimensions, one may define a Green's function, $G(\mathbf{r}, \mathbf{r}')$, satisfying

$$LG(\mathbf{r}, \mathbf{r}') = \delta(\mathbf{r} - \mathbf{r}'), \tag{C-2}$$

where $\delta(\mathbf{r} - \mathbf{r}')$ is the three-dimensional Dirac delta function with non-zero contribution only at the point, $\mathbf{r} = \mathbf{r}'$. The solution of the differential equation (C-1) is then given by

$$\phi(\mathbf{r}) = \int G(\mathbf{r}, \mathbf{r}') f(\mathbf{r}') \, \mathrm{d}^3\mathbf{r}', \tag{C-3}$$

which is easily verified as a solution by inserting the expression into equation (C-1),

$$L\phi(\mathbf{r}) = L \int G(\mathbf{r}, \mathbf{r}') f(\mathbf{r}') \, \mathrm{d}^3\mathbf{r}'. \tag{C-4}$$

The operator, L, may be moved inside the integral, where it operates only on unmarked coordinates, $\mathbf{r}$. Using the definition of the Green's function (C-2) then gives

$$L\phi(\mathbf{r}) = \int \delta(\mathbf{r} - \mathbf{r}') f(\mathbf{r}') \, \mathrm{d}^3\mathbf{r}' = f(\mathbf{r}) \qquad Q.E.D. \tag{C-5}$$

Knowing the Green's function therefore gives a method for solving the differential equation (C-1). The idea is that the Green's function gives the response of the system at $\mathbf{r}$ from a source located at $\mathbf{r}'$, and summing up all contributions gives the net result of the unknown function, $\phi(\mathbf{r})$.

C.2 Properties of the Relevant Green's Function

The partial differential equation,

$$\nabla \times \nabla \times \mathbf{E}_i(\mathbf{r}) - k_i^2 \mathbf{E}_i(\mathbf{r}) = i\omega\mu_i \mathbf{j}(\mathbf{r}), \tag{C-6}$$

has a well known dyadic Green's function, $\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}')$, (from Ref. [3]),

$$\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}') = \left(\bar{\mathbf{1}} + \frac{\nabla\nabla}{k_i^2}\right) \frac{e^{ik_i|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|}, \tag{C-7}$$

where $\bar{\mathbf{1}} = \delta_{ij}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j)$ is the identity dyad and the operator, $\nabla\nabla = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}(\hat{\mathbf{x}}_i \otimes \hat{\mathbf{x}}_j)$, is a dyadic differential operator. It may be shown that the following two relations between the Green's function, $\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}')$, and its transposed is satisfied (from Ref. [3]),

$$\bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}')^T = \bar{\mathbf{G}}_i(\mathbf{r}', \mathbf{r}). \tag{C-8}$$

That is, transposing the Green's function corresponds to swapping the observation point with the source point. The next relation is the following,

$$[\nabla \times \bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}')]^T = -\nabla \times \bar{\mathbf{G}}_i(\mathbf{r}', \mathbf{r}). \tag{C-9}$$

Furthermore, Eq. (28) in Ref. [3] gives the following relation,

$$\nabla \times \bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}') = [\nabla \cdot \bar{\mathbf{G}}_i(\mathbf{r}, \mathbf{r}')] \times \bar{\mathbf{1}}. \tag{C-10}$$

The analogous cross product and dot product operations are defined in appendix B.3.2 and the transposing operator of a dyadic tensor is defined in appendix B.3.6.

Appendix D

Method of Weighted Residuals

The Method of Weighted Residuals is a numerical technique of solving a linear operator equation by transformation to a matrix equation. The main sources for this section were Ref. [17] and Ref. [18]. The basis of the method is having the following linear operator equation,

$$L\phi(\mathbf{x}) = l(\mathbf{x}),\ \mathbf{x} \in A \subset \mathbb{R}^n, \tag{D-1}$$

where ϕ is the unknown function which is to be determined, l is a known function, L is a linear operator, A is some domain of $\mathbf{x}$ where we want a solution to our problem and n is the number of dimensions $\mathbf{x}$ contains.

D.1 Linear Operator

The operator, L, in Eq. (D-1) may for instance be a differential or an integration operator. The linearity of the operator is expressed through the following equations (from Ref. [19]),

$$L\sum_i \xi_i(\mathbf{x}) = \sum_i L\xi_i(\mathbf{x}), \tag{D-2}$$

$$L[c\xi_i(\mathbf{x})] = cL\xi_i(\mathbf{x}) \tag{D-3}$$

where c is a constant and all the functions, $\xi_i(\mathbf{x})$, are arbitrary and real valued. Now, say we have the following two relations,

$$L\phi_R(\mathbf{x}) = l_R(\mathbf{x}), \tag{D-4}$$

$$L\phi_I(\mathbf{x}) = l_I(\mathbf{x}), \tag{D-5}$$

where ϕ_R, l_R, ϕ_I and l_I are all real valued functions. Multiplying Eq. (D-5) with the imaginary unit, i, and adding the two equations leads to,

$$L\phi_R(\mathbf{x}) + iL\phi_I(\mathbf{x}) = l_R(\mathbf{x}) + il_I(\mathbf{x}). \tag{D-6}$$

Using the linearity property of the operator, L, from Eq. (D-2) gives,

$$L\phi_Z(\mathbf{x}) = l_Z(\mathbf{x}), \tag{D-7}$$

where $\phi_Z(\mathbf{x}) = \phi_R(\mathbf{x}) + iL\phi_I(\mathbf{x})$ and $l_Z(\mathbf{x}) = l_R(\mathbf{x}) + il_I(\mathbf{x})$. This shows that the linearity property of the operator, L, allows the functions $\phi(\mathbf{x})$ and $l(\mathbf{x})$ in Eq. (D-1) to be complex valued. If we instead say we have N relations of the kind in Eq. (D-1) and we multiply each relation with a different unit basis vector, $\hat{\mathbf{x}}_i$, of the space $\mathbb{R}^N$ and then add all the relations we get the following equation,

$$L\phi_1(\mathbf{x})\hat{\mathbf{x}}_1 + L\phi_2(\mathbf{x})\hat{\mathbf{x}}_2 + ... + \phi_N(\mathbf{x})\hat{\mathbf{x}}_N = l_1(\mathbf{x})\hat{\mathbf{x}}_1 + l_2(\mathbf{x})\hat{\mathbf{x}}_2 + ... + l_N(\mathbf{x})\hat{\mathbf{x}}_N. \tag{D-8}$$

Again using the linearity property from Eq. (D-2) gives the following result,

$$L\boldsymbol{\phi}(\mathbf{x}) = \mathbf{l}(\mathbf{x}), \tag{D-9}$$

where $\boldsymbol{\phi}$ and $\mathbf{l}$ are vector functions in N dimensions. Thus even though real valued functions are used for ϕ and l in the following text the linearity property makes the results below valid also for complex vector functions.

D.2 Transformation to Matrix Equation

The MWR seeks to approximate the unknown function, $\phi(\mathbf{x})$, by a linear combinations of N linearly independent basis functions, $f_n(\mathbf{x})$,

$$\phi(\mathbf{x}) \approx \Phi(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n f_n(\mathbf{x}), \tag{D-10}$$

where α_n are constant expansion coefficients. The method defines a set of N *weighting functions*, $w_1(\mathbf{x}), w_2(\mathbf{x}), ..., w_m(\mathbf{x}), ..w_N(\mathbf{x})$. It is appropriate to define the residual, R, as the following,

$$R(\mathbf{x}) = L\Phi(\mathbf{x}) - l(\mathbf{x}). \tag{D-11}$$

The residual, R($\mathbf{x}$), is nonzero if the approximation, $\Phi(\mathbf{x})$, is not equal to the unknown function, $\phi(\mathbf{x})$. The MWR is based on forcing this residual to zero in some average using the weighting functions above,

$$\int_A w_m(\mathbf{x}) R(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 0, \quad m \in [1, N]. \tag{D-12}$$

There are several different ways of choosing these weighting functions, all giving rise to sub types of the MWR. One of these ways is presented in Appendix D.3. We may define the operation in Eq. (D-12) as the *inner product* of w_m and R and write it in the following equivalent form,

$$\int_A w_m(\mathbf{x})R(\mathbf{x})\,\mathrm{d}\mathbf{x} = [w_m, R] = 0, \quad m \in [1, N]. \tag{D-13}$$

By using Eqs. (D-10) and (D-13) Eq. (D-12) may be reformulated in the following way,

$$\int_A L\Phi(\mathbf{x})w_m(\mathbf{x}) - l(\mathbf{x})w_m(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0, \quad m \in [1, N],$$
$$\int_A \left(L\sum_{n=1}^{N} \alpha_n f_n(\mathbf{x}) \right) w_m(\mathbf{x}) - l(\mathbf{x})w_m(\mathbf{x})\,\mathrm{d}\mathbf{x} = 0, \quad m \in [1, N]. \tag{D-14}$$

The linearity property of Eq. (D-2) leads to the following equation,

$$\int_A \left(\sum_{n=1}^{N} \alpha_n L f_n(\mathbf{x}) \right) w_m(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_A l(\mathbf{x})w_m(\mathbf{x})\,\mathrm{d}\mathbf{x}, \quad m \in [1, N]. \tag{D-15}$$

The summation operation and thus the constant expansion coefficients, α_n, may be pulled outside the integral giving,

$$\sum_{n=1}^{N} \alpha_n \int_A w_m(\mathbf{x}) L f_n(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int_A l(\mathbf{x})w_m(\mathbf{x})\,\mathrm{d}\mathbf{x}, \quad m \in [1, N]. \tag{D-16}$$

Using the definition of the inner product from Eq. (D-13) leads to,

$$\sum_{n=1}^{N} \alpha_n [w_m, L f_n] = [l, w_m], \; m \in [1, N]. \tag{D-17}$$

This may be written as the following matrix equation,

$$A\mathbf{v} = \mathbf{b}, \tag{D-18}$$

where each element, v_n, of the unknown vector, $\mathbf{v}$ is equal to the coefficients, α_n, and we have the following known matrix elements, A_{mn}, and vector elements b_m,

$$A_{mn} = [w_m, L f_n], \tag{D-19}$$
$$b_m = [l, w_m]. \tag{D-20}$$

This concludes the derivation from the integral equation of Eq. (D-1) to the matrix equation of Eqs. (D-18)-(D-20).

D.3 Galerkin's Method of Weighted Residuals

Galerkin's Method of Weighted Residuals is one of the sub types of the MWR. In this method the weighting functions, w_n, are defined as,

$$w_n = \frac{\partial}{\partial \alpha_n} \Phi(\mathbf{x}). \tag{D-21}$$

With the definition in Eq. (D-10) this leads to the fact that the weighting functions are identical to the basis functions of the approximation, $\Phi(\mathbf{x})$,

$$w_n = f_n(\mathbf{x}). \tag{D-22}$$

Appendix E

Gaussian Quadrature Integration

Gaussian quadrature integration is a way of approximating the value of a definite integral numerically. The method approximates the integral by a weighted sum of function values evaluated at certain points within the integration domain. In one dimension the n-point weighted sum is exact for polynomial integrands of order $2n$-1.

E.1 General Gaussian Quadrature Formula

In one dimension the Gaussian quadrature method is given by the following equation,

$$\int_L f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} w_i f(x_i), \tag{E-1}$$

where n denote the number of points, x_i, where the integrand, f, is evaluated and w_i denotes the i'th weighting constant. When extended to two dimensional integrals the Gaussian quadrature method transforms to the following equation given in Ref. [7],

$$\int_S f(\mathbf{x})\,\mathrm{d}S \approx A\sum_{i=1}^{n} w_i f(\mathbf{x}_i), \tag{E-2}$$

where the domain area, A, is pulled out of the weighting constants, w_i and the variable, $\mathbf{x}$, denote a point in the domain S. The approximation involves, as in the one dimensional case, evaluating the function, f, at the n points denoted by $\mathbf{x}_i$.

E.2 Gaussian Quadrature Formulas for Triangles

In Ref. [7] there are given several sets of weighting functions, w_i, of different order, n, and degree of precision, d. The degree of precision is the maximum polynomial order of the integral for which the approximation is exact. The evaluation points are given in so called *area coordinates*. The advantage of this system is that the coordinates are independent of the shape of the triangle, which is very useful when dealing with triangles of arbitrary shape.

A point, $\mathbf{x}_i$, inside a triangle, $\triangle ABC$, is defined in area coordinates by the vectors from the origin to each of the corners of the triangle, $\mathbf{r}_A$, $\mathbf{r}_B$ and $\mathbf{r}_C$ as,

$$\mathbf{x}_i = \xi_i \mathbf{r}_A + \eta_i \mathbf{r}_B + \zeta_i \mathbf{r}_C, \tag{E-3}$$

where the coefficients (ξ_i, η_i, ζ_i) make out the area coordinates of the point, $\mathbf{x}_i$. The coordinates may also be defined by the masses needed in each of the triangle corners in order to make, $\mathbf{x}_i$, the center of mass. Note that this definition opens for an infinite amount of coordinates, $k(\xi_i, \eta_i, \zeta_i)$, where k is an arbitrary real number, all giving the same center of mass point, $\mathbf{x}_i$. The former definition however identifies a unique set of coordinates for each point on the triangle, where the sum always equals one,

$$\xi_i + \eta_i + \zeta_i = 1. \tag{E-4}$$

The 1-point Gaussian quadrature formula for a triangle is simply approximating the integrand as a constant. The constant value may be set to the function value of an arbitrary point inside the integration domain, but normal is choosing the centroid as evaluation point. This point is given in area coordinates as $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

Reference [7] gives formulas including three to thirteen evaluation points and degree of precision from two to seven. The first formula is given by the three evaluation points, $(\frac{1}{6}, \frac{1}{6}, \frac{2}{3})$, $(\frac{1}{6}, \frac{2}{3}, \frac{1}{6})$ and $(\frac{2}{3}, \frac{1}{6}, \frac{1}{6})$. An illustration of the latter is shown in Fig. E.1. The weighting constants are all equal, $w_1 = w_2 = w_3 = \frac{1}{3}$ and the formula is exact for polynomials of up to second degree.

Fig. E.1: Area coordinates example:

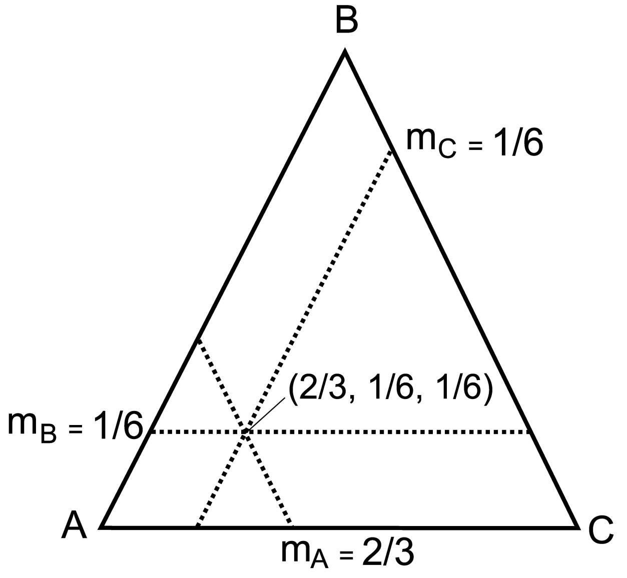

Figure E.1: Illustration of the point, $(m_A, m_B, m_C) = (\frac{2}{3}, \frac{1}{6}, \frac{1}{6})$, given in area coordinates.

Appendix F

The Recursive Discretization Algorithm

The implemented recursive discretization algorithm is a simple algorithm producing a discretization with small spread in element volume and shape. It basically starts off with six points on a unit sphere connected by twelve edges, making out eight equilateral triangles. The basic idea is then to recursively split each triangle into four smaller ones and normalizing the triangle corner points to the unit sphere. After a few iterations the sphere is divided into many small elements with corner points on the surface of the sphere.

F.1 Splitting the Triangular Elements

By managing the triangle splitting carefully the variance in shape and area may be kept low. Figure F.1 below shows how one triangular element is split into four smaller ones. Each new corner point is dividing the previous triangle edges in half. If the original triangles are equilateral this creates four new equilateral triangles, but the normalization operation placing the new points on the sphere surface spoils this symmetry making the center triangle slightly larger than the surrounding three. Thus the resulting elements are not exactly equal in shape and size, but the differences are small compared to for instance the *STRIPACK* triangulation algorithm from Ref. [9]. Another important difference is the possible number of elements in the final discretized sphere. The recursive algorithm allows for the number of elements to be 8, 32, 128, 512 etc., and no numbers in between these. The *STRIPACK* algorithm however allows for practically any number of corner points.

Fig. F.1: Triangle splitting:

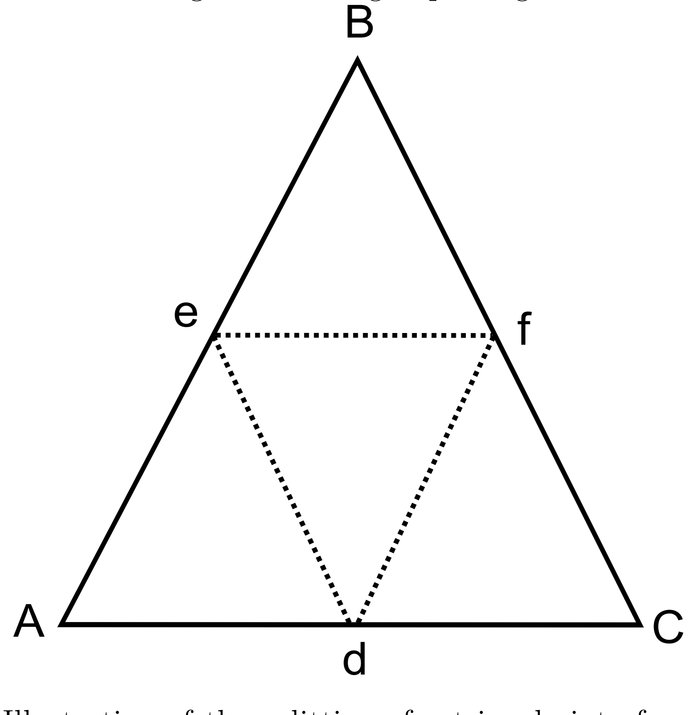

Figure F.1: Illustration of the splitting of a triangle into four new ones in the recursive discretization algorithm. The new corner points, d, e and f, are located at the midpoints of the edges, AB, BC and CA, respectively.

Appendix G

Fortran Code

This section includes the FORTRAN code for the numerical part of the thesis.

G.1 Discretization Module

The file *disc_mod.f90* contains the following FORTRAN code.

```
module disc_mod
!USE calculation_mod !, only : vec_len
! Module defining the discretization data structure and
! subroutines for filling the structure with data using
! different discretization algorithms.
implicit none

type :: Point
  real :: point(3)
end type Point

type :: Element
  integer :: nr_of_corners
  integer, dimension(:), allocatable :: corners
end type Element

type :: Pair
  integer :: corners(2)
  integer :: elements(2)
end type Pair

type Structure
  type(Point), dimension(:), allocatable   :: points
  type(Element), dimension(:), allocatable :: elements
  type(Pair), dimension(:), allocatable    :: neighbours
  type(Point), dimension(:), allocatable   :: midpoints
  type(Point), dimension(:,:), allocatable :: quadpoints
end type structure

contains

subroutine stripack_convert(struct, n)
! Generates a triangulation discretization using
! stripack and converts it to fit into the struct variable.
type(Structure) :: struct
integer :: n, i

! variables for stripack discretization
integer ( kind = 4 ), parameter :: nrow = 9
real    ( kind = 8 ) ds(n)
integer ( kind = 4 ) ier
real    ( kind = 8 ) x(n)
real    ( kind = 8 ) y(n)
real    ( kind = 8 ) z(n)
real    ( kind = 8 ) sc
integer ( kind = 4 ) list(6*n)
integer ( kind = 4 ) lptr(6*n)
integer ( kind = 4 ) lend(n)
integer ( kind = 4 ) iwk(2*n)
integer ( kind = 4 ) lnew
real    ( kind = 8 ) rlat(n)
real    ( kind = 8 ) rlon(n)
integer ( kind = 4 ) nt
integer ( kind = 4 ) ltri(nrow,2*n-4)

print *, 'Creating delaunay discretization with #points = ', n

call random_number ( harvest = rlat(1:n) )
call random_number ( harvest = rlon(1:n) )

rlat(1:n) = ( ( 1.0D+00 - rlat(1:n) ) * (  -90.0D+00 ) &
            +             rlat(1:n)   *     90.0D+00 )

rlon(1:n) = ( ( 1.0D+00 - rlon(1:n) ) * ( -180.0D+00 ) &
            +             rlon(1:n)   *    180.0D+00 )
sc = atan ( 1.0D+00 ) / 45.0D+00
```

```fortran
x(1:n) = sc * rlon(1:n)
y(1:n) = sc * rlat(1:n)
call trans ( n, y, x, x, y, z )
call trmesh ( n, x, y, z, list, lptr, lend, lnew, iwk, iwk(n+1),&
              ds, ier )
!do i=1,n
! write (*,"(F10.5,F10.5,E15.5)"),x(i), y(i), z(i)
!end do
!call trprnt ( n, x, y, z, 0, nrow, nt, ltri)
call trlist ( n, list, lptr, lend, nrow, nt, ltri, ier )
!call trlprt ( n, x, y, z, 0, nrow, nt, ltri)
!do i=1,nt
! write (*,"(4i4)"), i, ltri(1, i), ltri(2,i), ltri(3,i)
!end do

!Allocating and defining struct
allocate(struct%points(n))
do i=1,n
  struct%points(i)%point=(/x(i), y(i), z(i)/)
end do
allocate(struct%elements(nt))
do i=1,nt
  struct%elements(i)%nr_of_corners=3
  allocate(struct%elements(i)%corners(struct%elements(i)&
  %nr_of_corners))
  struct%elements(i)%corners=(/ltri(1:struct%elements(i)&
  %nr_of_corners, i)/)
end do

call find_neighbours(struct)
call find_midpoints(struct)
call find_quadpoints(struct)

end subroutine

subroutine recursive_triangulation(struct, n)
! Creating symmetric triangular discretization recursively
! n=6,18,58,258,1026, ...
type(Structure) :: struct
integer :: n, i, ne, np_curr, ne_curr, lim, split_num

print *, 'Creating discretization recursively with #points = ', n

```

```
allocate(struct%points(n))
struct%points(1)%point=(/ 1.0, 0.0, 0.0 /)
struct%points(2)%point=(/-1.0, 0.0, 0.0 /)
struct%points(3)%point=(/ 0.0, 1.0, 0.0 /)
struct%points(4)%point=(/ 0.0,-1.0, 0.0 /)
struct%points(5)%point=(/ 0.0, 0.0, 1.0 /)
struct%points(6)%point=(/ 0.0, 0.0,-1.0 /)

!ne=2*(n-2)
!split_num=1+int(log10((2+n)/8.0)/log10(4.0))
ne=2*(n-2) !int(8*4**(split_num-1))
!print *, split_num, ne
!read(*,*)

allocate(struct%elements(ne))
do i=1,ne
  struct%elements(i)%nr_of_corners=3
  allocate(struct%elements(i)%corners(3))
end do
struct%elements(1)%corners=(/ 1, 5, 3/)
struct%elements(2)%corners=(/ 3, 5, 2/)
struct%elements(3)%corners=(/ 2, 5, 4/)
struct%elements(4)%corners=(/ 4, 5, 1/)
struct%elements(5)%corners=(/ 1, 3, 6/)
struct%elements(6)%corners=(/ 3, 2, 6/)
struct%elements(7)%corners=(/ 2, 4, 6/)
struct%elements(8)%corners=(/ 4, 1, 6/)

np_curr=6
ne_curr=8

lim=int(log10(ne/6.0)/log10(4.0))
do i=1, lim
  call recursive_split(struct, np_curr, ne_curr)
  ne_curr=ne_curr*4
  np_curr=(ne_curr/2)+2
end do

call find_neighbours(struct)
call find_midpoints(struct)
call find_quadpoints(struct)

end subroutine
```

```

subroutine recursive_split(struct, n, ne)
integer                                   :: n, ne, i, counter, &
                                             ind_a, ind_b, ind_c
type(Structure)                           :: struct
type(Element), dimension(:), allocatable  :: el_list
real                                      :: a(3),b(3),c(3), &
                                             v0(3), v1(3), v2(3)
logical                                   :: lo(3)

allocate(el_list(4*ne))
counter=n+1

do i=1,ne
lo=(/ .false., .false., .false. /)
a=normalize(0.5*(struct%points(struct%elements(i)%corners(1))&
%point+struct%points(struct%elements(i)%corners(3))%point ))
b=normalize(0.5*(struct%points(struct%elements(i)%corners(1))&
%point+struct%points(struct%elements(i)%corners(2))%point ))
c=normalize(0.5*(struct%points(struct%elements(i)%corners(2))&
%point+struct%points(struct%elements(i)%corners(3))%point ))
ind_a=point_in_list(a,struct%points)
lo(1)=ind_a.eq. -1
if (lo(1))then ! a is not present in list
  struct%points(counter)%point=a
  ind_a=counter
  counter=counter+1
end if

ind_b=point_in_list(b,struct%points)
lo(1)=ind_b.eq. -1
if (lo(1))then ! a is not present in list
  struct%points(counter)%point=b
  ind_b=counter
  counter=counter+1
end if

ind_c=point_in_list(c,struct%points)
lo(1)=ind_c.eq. -1
if (lo(1))then ! a is not present in list
  struct%points(counter)%point=c
  ind_c=counter
  counter=counter+1
```

```
end if
struct%elements(ne+(i-1)*3+1)%corners=&
(/ struct%elements(i)%corners(1), ind_b, ind_a/)
struct%elements(ne+(i-1)*3+2)%corners=&
(/ ind_b,  struct%elements(i)%corners(2), ind_c/)
struct%elements(ne+(i-1)*3+3)%corners=&
(/ ind_a,  ind_c, struct%elements(i)%corners(3)/)
struct%elements(i)%corners=(/ ind_a, ind_b, ind_c /)
end do


end subroutine

function point_in_list(p, list)
! Function returning the index of the point p in list
! and returns -1 if it is not in the list
real
type(Point), dimension(:), allocatable        :: list
integer
integer

point_in_list=-1
do i=1,size(list)
temp=list(i)%point-p
temp(1)=sqrt(temp(1)**2+temp(2)**2+temp(3)**2)
if (temp(1)<1e-30)then
  point_in_list=i
  return
end if
end do
return
end function

subroutine find_quadpoints(struct)
! Subroutine finding 3 points in each element
! for gauss quad integration and saving the
! resulting points in struct%quadpoints
! Compatible with only triangular elements
type(Structure) :: struct
integer         :: m, ne

ne=size(struct%elements)
allocate(struct%quadpoints(ne, 3))
```

```
do m=1,ne
  struct%quadpoints(m,1)%point=(2.0/3.0) &
  *struct%points(struct%elements(m)%corners(1))%point &
  +(1.0/6.0)*struct%points(struct%elements(m)%corners(2))%point&
  +(1.0/6.0)*struct%points(struct%elements(m)%corners(3))%point
  struct%quadpoints(m,2)%point=(1.0/6.0) &
  *struct%points(struct%elements(m)%corners(1))%point &
  +(2.0/3.0)*struct%points(struct%elements(m)%corners(2))%point &
  +(1.0/6.0)*struct%points(struct%elements(m)%corners(3))%point
  struct%quadpoints(m,3)%point=(1.0/6.0) &
  *struct%points(struct%elements(m)%corners(1))%point &
  +(1.0/6.0)*struct%points(struct%elements(m)%corners(2))%point &
  +(2.0/3.0)*struct%points(struct%elements(m)%corners(3))%point
end do
end subroutine

real function area(point_list)
! Function calculating the area of an element consisting of the
! points contained by point_list.
! In present form only compatible with triangular elements,
! but may be extended.
type(Point), dimension(:) :: point_list
real , dimension(3) :: v1, v2

if (size(point_list) .eq. 3)then
  v1=point_list(2)%point-point_list(1)%point
  v2=point_list(3)%point-point_list(1)%point
  area=0.5*sqrt((v1(2)*v2(3)-v1(3)*v2(2))**2 &
  +(v1(3)*v2(1)-v1(1)*v2(3))**2+(v1(1)*v2(2)-v1(2)*v2(1))**2)
  return
else if (1 .eq. 0) then
  print *, 'false!'
  area=-999
  return
endif
end function

subroutine find_midpoints(struct)
! Subroutine finding all centroids of the elements in struct
! and saving the resulting points in struct%midpoints.
! Compatible with only triangular elements.
integer         :: ne, i, j, corners
type(Structure) :: struct
```

```
real            :: v1(3),v2(3),v3(3), mid(3)

ne=size(struct%elements)
if (allocated(struct%midpoints))then
  deallocate(struct%midpoints)
endif
allocate(struct%midpoints(ne))
do i=1,ne
  v1=struct%points(struct%elements(i)%corners(1))%point
  v2=struct%points(struct%elements(i)%corners(2))%point
  v3=struct%points(struct%elements(i)%corners(3))%point
  mid=(v1+v2+v3)/3.0
  struct%midpoints(i)%point=mid
end do
end subroutine

subroutine find_neighbours(struct)
! Subroutine finding all neighbours in struct and
! saving the result in struct%neighbours.
! Compatible with arbitrary number of corners.
integer                               :: ne, i, j, d1,d2, nn, &
                                         out_arr(2)
type(Pair), dimension(:), allocatable :: temp_neighbours
type(Structure)                       :: struct

nn=0
ne=size(struct%elements)
if (allocated(struct%neighbours))then
  deallocate(struct%neighbours)
endif
allocate(struct%neighbours(3*ne/2))

allocate(temp_neighbours(3*ne/2))
do i=1,ne
  do j=i+1,ne
  d1=size(struct%elements(i)%corners)
  d2=size(struct%elements(j)%corners)
  out_arr=common_2(struct%elements(i)%corners, &
  struct%elements(j)%corners, d1, d2 )
    if ( out_arr(2) /= -1 )then
      nn=nn+1
      struct%neighbours(nn)%elements=(/ i,j /)
      struct%neighbours(nn)%corners=(/out_arr(1:2)/)
    end if
  end do
end do

end subroutine

function common_2(arr1, arr2, dim1, dim2)
! function checking whether arr1 and arr2 have 2 common elements.
! If true, the function returns the two common elements
! If false, the function returns (-1, -1)
! Compatible with arbitrary number of dimensions
integer :: dim1, dim2, i, j, n_equal, common_2(2), n_different
integer :: arr1(dim1), arr2(dim2)
common_2=(/-1,-1/)

n_equal=0
do i=1,dim1
n_different=0
do j=1,dim2
  if (arr1(i) == arr2 (j))then
    n_equal=n_equal+1
    common_2(n_equal)=arr1(i)
  endif
end do
end do
return

end function common_2


function normalize(v)
real :: v(3), normalize(3), l

l = sqrt(v(1)**2+v(2)**2+v(3)**2)
normalize(1:3)=v(1:3)/l
return
end function normalize

subroutine area_dist(struct, disc_type, area_list)
! Subroutine generating the area distribution and
! printing the result to the file 'area_dist'.
real, parameter                :: pi=4.0*atan(1.0)
integer, parameter             :: out_unit=20
```

```
integer                           :: n, ne, disc_type, i, ints, &
                                     indx
real                              :: exp_avg, mean, std_dev, &
                                     area_list(:), delta, start,&
                                     max_area
real, dimension(:), allocatable :: hist
type(Structure)                   :: struct

n=size(struct%points)
ne=size(struct%elements)

!exp_avg=sqrt(8*pi/ne)
exp_avg=4*pi/ne

mean=0.0
max_area=0.0
do i=1,ne
mean=mean+area_list(i)
end do
mean=mean/ne
std_dev=0.0
do i=1, ne
std_dev=std_dev+(area_list(i)-mean)**2
end do
std_dev=sqrt(std_dev/ne)

start=mean-3*std_dev
ints=int(ne/20)+1

allocate(hist(ints))
do i=1, ints
  hist(i)=0.0
end do
delta=6*std_dev/ints

do i=1, ne
  indx=int((area_list(i)-start)/delta)+1
  if (indx<=ints .and. indx>0)then
    hist(indx)=hist(indx)+1.0/ne
  end if
  if (area_list(i)>max_area)then
    max_area=area_list(i)
  end if
end do

print *, 'mean:', mean
print *, 'exp_avg:', exp_avg
print *, 'std_dev:', std_dev
print *, 'max_area',max_area
print *, 'delta', delta
print *, 'intervals:', ints

! Printing the resulting area-distribution to area_dist file
open(unit=out_unit,file="area_dist",action="write", &
status="replace")
write (out_unit, *) 'ne', ne
write (out_unit, *) 'disc_type', disc_type
write (out_unit, *) 'mean', mean
write (out_unit, *) 'std_dev', std_dev
write (out_unit, *) 'max_area',max_area

do i=1,ints
if (start+delta*i>=0.0)then
  !write (*, "(e10.2,e10.2)") start+delta*i, hist(i)
  write (out_unit, "(e14.5,e14.5)") start+delta*i, hist(i)
end if
end do

close (out_unit)
end subroutine

end module
```

G.2 Calculation Module

The file *calculation_mod_omp.f90* contains the following FORTRAN code.

```
module calculation_mod_omp
USE omp_lib
USE disc_mod
! Module containing functions useful in the
! calculation part of the program
implicit none

type :: Vector_c
        complex :: vector(3)
```

```fortran
end type Vector_c

real, parameter :: pi=4.0*atan(1.0), c=299792458.0
complex, parameter :: im=(0.0, 1.0)

contains

function dot_prod_c(z1,z2)
! Function returning the dot product of two
! complex vectors z1 and z2
complex , dimension(:) :: z1,z2
complex :: dot_prod_c
integer :: i

if (size(z1)/=size(z2))then
        print *, 'Error - not equal sized vectors'
        return
else
        dot_prod_c=(0.0, 0.0)
        do i=1, size(z1)
                dot_prod_c=dot_prod_c+z1(i)*z2(i)
        end do
        return
end if

end function

function dot_prod_r(r1,r2)
! Function returning the dot product of
! two complex vectors z1 and z2
real , dimension(:) :: r1,r2
real :: dot_prod_r
integer :: i

if (size(r1)/=size(r2))then
        print *, 'Error - not equal sized vectors'
        return
else
        dot_prod_r=0.0
        do i=1, size(r1)
                dot_prod_r=dot_prod_r+r1(i)*r2(i)
        end do
        return
```

```
end if

end function

function G(p1, p2, i, j, k)
! Function returning the Greens tensor element i,j
! when p1 is the observation point, p2 is the reference point
! and k is the wavenumber

real, dimension(3) :: p1, p2, r_vec!, k_vec
real               :: k, R, factor
integer            :: i,j,m
complex            :: G, alpha, Z

do m=1,3
        r_vec(m)=p1(m)-p2(m)
end do
!k=sqrt(k_vec(1)**2+k_vec(2)**2+k_vec(3)**2)
R=sqrt( (p1(1)-p2(1))**2 + (p1(2)-p2(2))**2 + (p1(3)-p2(3))**2 )
alpha=( ( -k**2/(R**3) )-( 3*k*im/(R**4) )+( 3/(R**5) ) )

!print *, k*R/pi
!print *, cexp(im*dot_prod_r(k_vec, r_vec))
!print *, cexp(im*k*R)
!print *, k*R/dot_prod_r(k_vec, r_vec)

if (i .ne. j)then
        factor=((p1(i)-p2(i))*(p1(j)-p2(j))/((k**2)*4.0*pi))
        G=factor*exp(im*(k*R))*alpha
else
        !new method
        Z=(R/k**2)*(((p1(i)-p2(i))**2)*alpha+(im*k-(1.0/R))/(R**2))
        G=exp(im*(k*R))*(1/(4*pi*R))*(1.0+Z)
end if
return
end function

function fn(r, struct, pair_n, area, ot)
! Function returning the function value of the n'th
! RWG basis function at the point r
! struct is the discretization data structure,
! pair_n is the pair of neighbouring elements
! for the n'th basis function,
! area is the area of the element considered,
! ot is 1 or 2 dependent on which
! element in the pair in which the point r is located.
! Works of course only for triangular elements
real            :: r(3), fn(3), p(3), area, L, &
                   p1(3), p2(3), L_vec(3)
type(Pair)      :: pair_n
type(Structure) :: struct
integer         :: i, j, ot, pos_neg

! Finds the corner (p+ or p- dependent on value of ot)
! which is not common for the two elements in pair n
do i=1, 3
  if (struct%elements(pair_n%elements(ot))%corners(i) &
  /= pair_n%corners(1) &
  .and. struct%elements(pair_n%elements(ot))%corners(i) &
  /= pair_n%corners(2))then
    p(1:3)=struct%points(struct%elements(pair_n%elements(ot))&
    %corners(i))%point
  end if
end do
p1(1:3)=struct%points(pair_n%corners(1))%point
p2(1:3)=struct%points(pair_n%corners(2))%point
L_vec(1:3)=p1-p2
L=sqrt(L_vec(1)**2+L_vec(2)**2+L_vec(3)**2)

! creating pos_neg from ot: (1 or 2) -> (1 or -1)
pos_neg=-1*(((ot-1)*ot)-1)

!print *, 'r:', r, 'p:', p, r-p
fn=pos_neg*(r-p)*(L/(2*area))
return
end function

function norm_vec_c(p1, p2, p3, mid)
! Function returning the complex normal vector
! (zero imaginary part) of a discretization element
! pointing outwards from origo when p1,p2,p3 are
! corners of the element.
real    :: p1(3), p2(3), p3(3), v1(3), v2(3), norm_vec_r(3), &
           l, mid(3)
complex :: norm_vec_c(3)
integer :: i
```

```
logical :: correct_way

v2(1:3)=p3(1:3)-p1(1:3)
v1(1:3)=p2(1:3)-p1(1:3)
norm_vec_r=cross_r(v1,v2)
l=sqrt(norm_vec_r(1)**2+norm_vec_r(2)**2+norm_vec_r(3)**2)
norm_vec_c(1:3)=(/ cmplx(norm_vec_r(1)),cmplx(norm_vec_r(2)),&
cmplx(norm_vec_r(3)) /)
norm_vec_c=norm_vec_c/l

correct_way=.true.
do i=1,3
  if (mid(i)*norm_vec_r(i)<0)then
    if ((norm_vec_r(i)>0.1 .or. norm_vec_r(i)<-0.1) )then
      correct_way=.false.
      print *,'wrong way:', norm_vec_c, 'mid:', mid
    end if
  end if
end do
if (correct_way.eqv. .false.)then
                norm_vec_c=-1.0*norm_vec_c
end if

return
end function


function cross_c(v1,v2)
! Function returning the cross product of
! two complex vectors v1 and v2
complex, dimension(3) :: v1, v2, cross_c

cross_c(1)=v1(2)*v2(3)-v1(3)*v2(2)
cross_c(2)=v1(3)*v2(1)-v1(1)*v2(3)
cross_c(3)=v1(1)*v2(2)-v1(2)*v2(1)

return
end function

function cross_r(v1,v2)
! Function returning the cross product of
! two real vectors v1 and v2
real, dimension(3) :: v1, v2, cross_r
```

```

cross_r(1)=v1(2)*v2(3)-v1(3)*v2(2)
cross_r(2)=v1(3)*v2(1)-v1(1)*v2(3)
cross_r(3)=v1(1)*v2(2)-v1(2)*v2(1)

return
end function

function J_gen(k, k_hat, struct, area_list, my, E_in)
! Function calculating the surface current density, J,
! by the Method of Weighted Residuals
! k is the wave number, k_hat is the direction of the wave vector
! struct is the discretization data structure,
! area_list contains the area of all the elements,
! my is the permeability of region 1,
! E_inc contains the incoming E-field at r=/(0,0,0)/,
! E_inc=E_inc_hat*E_inc_0*exp(k dot r)
integer                               :: t,l,m,n,s,q,dir,&
                                         n_el, n_pairs, info
integer, dimension(:), allocatable    :: ipiv
complex                               :: G_tensor(3,3), temp_c, &
                                         f_c1(3), f_c2(3), &
                                         f_c3(3), sum1(3), sum2, &
                                         sum3
real                                  :: k, f_r(3), my, E_in(3),&
                                         k_hat(3), temp_r(3), &
                                         least
real, dimension(:), allocatable       :: area_list
complex, dimension(:,:), allocatable  :: D_mat, D_mat_pre
real, dimension(:,:,:), allocatable   :: f_mat
complex, dimension(:), allocatable    :: b_vec, rs_vec
type(Vector_c) , dimension(:), allocatable :: J_gen, E_inc_list
type(Structure)                           :: struct

n_pairs=size(struct%neighbours)
n_el=size(struct%elements)
allocate(E_inc_list(n_el))
allocate(J_gen(n_el))
allocate(D_mat(n_pairs,n_pairs))
allocate(D_mat_pre(n_pairs,n_pairs))
allocate(f_mat(n_pairs,2,3))
allocate(b_vec(n_pairs))
allocate(ipiv(n_pairs))

allocate(rs_vec(n_pairs))

do m=1,n_pairs
  do t=1,2
    f_mat(m,t,1:3)=fn(struct%midpoints(struct%neighbours(m)% &
    elements(t))%point, struct, struct%neighbours(m), &
    area_list(struct%neighbours(m)%elements(t)), t)
  end do
end do

! Calculating the incoming E-field, E_inc, at all centroids
do m=1,n_el
  E_inc_list(m)%vector=E_in*exp(im*dot_prod_r(k*k_hat, &
  struct%midpoints(m)%point))
end do

do m=1,n_pairs  ! for all rows of the linear system
  if (modulo(m,100).eq.0) then
    print *, 'pair nr', m, 'of total', n_pairs
  end if
  do n=1,n_pairs  ! for all columns of the linear system
    sum2=(0.0,0.0)
    do t=1,2  ! for both elements of the outer integral
      sum1=(/ (0.0,0.0),(0.0,0.0),(0.0,0.0) /)
      do l=1,2  ! for both elements of the inner integral
        if (struct%neighbours(m)%elements(t) &
        .ne.struct%neighbours(n)%elements(l))then
          do dir=1,3 ! for all three spatial directions
            do q=1,3
              G_tensor(dir,q)=G(struct%midpoints &
              (struct%neighbours(m)%elements(t))%point, &
              struct%midpoints(struct%neighbours(n)%elements(l))&
              %point, dir, q, k)
           end do
          end do
          f_c1(1:3)=cmplx(f_mat(n,l,1:3))
          do dir=1,3
            temp_c=dot_prod_c(f_c1,G_tensor(dir,1:3))
            sum1(dir)=sum1(dir)+area_list(struct%neighbours(n)&
            %elements(l))*temp_c
          end do
        end if
      end do
      f_c2(1:3)=cmplx(f_mat(m,t,1:3))!cmplx(f_r(1:3))
      sum2=sum2+dot_prod_c(f_c2, sum1) &
      *area_list(struct%neighbours(m)%elements(t))
    end do
    D_mat(m,n)=sum2
  end do
  sum3=(0.0,0.0)
  do t=1,2
    f_c3(1:3)=cmplx(f_mat(m,t,1:3))
    sum3=sum3+dot_prod_c(f_c3, E_inc_list(struct%neighbours(m)&
    %elements(t))%vector)*&
    area_list(struct%neighbours(m)%elements(t))
  end do
  b_vec(m)=sum3
  rs_vec(m)=sum3
end do

D_mat=D_mat*k*c*my/im
D_mat_pre=D_mat

D_mat(1,1:2)=(/ (2.0,0.0), (2.0,0.0)/)
D_mat(2,1:2)=(/ (3.0,0.0), (5.0,0.0)/)

D_mat_pre=D_mat

call CGESV(n_pairs, 1, D_mat, n_pairs, ipiv, b_vec, n_pairs, info)

temp_r(1)=0.0
temp_r(2)=0.0
do m=1, n_pairs
  temp_c=(0.0,0.0)
  do n=1, n_pairs
    temp_c=temp_c+D_mat_pre(m,n)*b_vec(n)
  end do
        !print *, temp_c, rs_vec(m), abs(temp_c-rs_vec(m))
        temp_r(1)=temp_r(1)+abs(temp_c-rs_vec(m))
        temp_r(2)=temp_r(2)+abs(rs_vec(m))
        !print *, abs(rs_vec(m))
end do

if (info .eq. 0)then
  print *, 'Linear system solved successfully!'
```

```
else
  print *, 'An error occured while solving the linear system'
  stop
end if

print *, 'avg error:', temp_r(1)/temp_r(2)
!read(*,*)

! Calculating the resulting current densities
! in all element centroids
do n=1, n_el
  J_gen(n)%vector=(/(0.0,0.0), (0.0,0.0), (0.0,0.0)/)
end do

do n=1,n_pairs
  do t=1,2
      J_gen(struct%neighbours(n)%elements(t))%vector= &
      J_gen(struct%neighbours(n)%elements(t))%vector &
      +b_vec(n)*f_mat(n,t,1:3)
  end do
end do
return
end function

function J_gen_quad(k, k_hat, struct, area_list, my, E_in)
! Function calculating the surface current density, J,
! by the Method of Weighted Residuals using
! the 3-point Gaussian quadrature formula
! The function is parallelized
! k is the wave number, k_hat is the direction of the wave vector
! struct is the discretization data structure,
! area_list contains the area of all the elements,
! my is the permeability of region 1,
! E_inc contains the incoming E-field at r=/(0,0,0)/,
! E_inc=E_inc_hat*E_inc_0*exp(k dot r)
integer                                :: t,l,m,n,s,p,q, &
                                          dir, n_el, n_pairs, &
                                          info, counter
integer, dimension(:), allocatable     :: ipiv
complex                                :: G_tensor(3,3), temp_c,&
                                          f_c1(3), f_c2(3), &
                                          f_c3(3), sum1(3), &
                                          sum2, sum3, sum_q(3),&
```

```
                                           sum_p, sum_q2
real                                    :: k, f_r(3), my, E_in(3),&
                                           k_hat(3), temp_r(3),&
                                           least,r_ob(3), r_ref(3)
real, dimension(:), allocatable         :: area_list
complex, dimension(:,:), allocatable    :: D_mat, D_mat_pre
real, dimension(:,:,:), allocatable     :: f_mat2
real, dimension(:,:,:,:), allocatable   :: f_mat
complex, dimension(:), allocatable      :: b_vec, rs_vec
type(Vector_c) , dimension(:,:), allocatable  :: E_inc_list
type(Vector_c) , dimension(:), allocatable    :: J_gen_quad
type(Structure)                               :: struct

n_pairs=size(struct%neighbours)
n_el=size(struct%elements)
allocate(E_inc_list(n_el,3))
allocate(J_gen_quad(n_el*3))
allocate(D_mat(n_pairs,n_pairs))
allocate(D_mat_pre(n_pairs,n_pairs))
allocate(f_mat(n_pairs,2,3,3))
allocate(f_mat2(n_pairs,2,3))
allocate(b_vec(n_pairs))
allocate(ipiv(n_pairs))

allocate(rs_vec(n_pairs))

do m=1,n_pairs
  do t=1,2
    f_mat2(m,t,1:3)=fn(struct%midpoints(struct%neighbours(m)&
    %elements(t))%point, struct, struct%neighbours(m), &
    area_list(struct%neighbours(m)%elements(t)), t)
          do q=1,3
      f_mat(m,t,q,1:3)=fn(struct%quadpoints(struct%neighbours(m)&
      %elements(t),q)%point, struct, struct%neighbours(m), &
      area_list(struct%neighbours(m)%elements(t)), t)
    end do
  end do
end do

! Calculating the incoming E-field, E_inc, at all quadrature points
do m=1,n_el
  do q=1,3
    E_inc_list(m,q)%vector= &
    E_in*exp(im*dot_prod_r(k*k_hat,struct%quadpoints(m,q)%point))
  end do
end do

counter=0
!$omp parallel private ( m,n, t,p,l, q, dir, s, G_tensor) &
!$omp private ( sum2, sum_p, sum1, sum_q, r_ob ) &
!$omp private ( r_ref, temp_c, f_c1, f_c2, f_c3, sum_q2, sum3) &
!$omp shared ( n_pairs, area_list, struct, b_vec, rs_vec, D_mat)
!$omp do
do m=1,n_pairs ! for all rows of the linear system
  counter=counter+1
  if (modulo(counter,100).eq.0) then
    print *, 'pair nr', counter, 'of total', n_pairs
  end if
  do n=1,n_pairs ! for all columns of the linear system
    sum2=(0.0,0.0)
    do t=1,2 ! for both elements of the outer integral
      sum_p=(0.0,0.0)
      do p=1,3
        sum1=(/ (0.0,0.0),(0.0,0.0),(0.0,0.0) /)
        do l=1,2 ! for both elements of the inner integral
          sum_q=(/ (0.0,0.0),(0.0,0.0),(0.0,0.0) /)
          do q=1,3
            r_ob=struct%quadpoints(struct%neighbours(m)&
            %elements(t),p)%point
            r_ref=struct%quadpoints(struct%neighbours(n)&
            %elements(l),q)%point
            if (vec_len(r_ob-r_ref)>1e-15)then
              do dir=1,3  ! for all three spatial directions
                do s=1,3
                  G_tensor(dir,s)=G(r_ob, r_ref, dir, s, k)
                end do
              end do
              f_c1(1:3)=cmplx(f_mat(n,l,q,1:3))
              do dir=1,3
                temp_c=dot_prod_c(f_c1,G_tensor(dir,1:3))
                sum_q(dir)=sum_q(dir)+temp_c
              end do
            end if
          end do !q
          sum1(1:3)=sum1(1:3)+ &
          area_list(struct%neighbours(n)%elements(l))*sum_q(1:3)
        end do !l
        f_c2(1:3)=cmplx(f_mat(m,t,p,1:3))
        sum_p=sum_p+dot_prod_c(sum1, f_c2)
      end do !p
      sum2=sum2+sum_p*(1.0/9.0)* &
      area_list(struct%neighbours(m)%elements(t))
    end do !t
    D_mat(m,n)=sum2
  end do
  sum3=(0.0,0.0)
  do t=1,2
    sum_q2=(0.0,0.0)
    do q=1,3
      f_c3(1:3)=cmplx(f_mat(m,t,q,1:3))
      sum_q2=sum_q2+dot_prod_c(f_c3, &
      E_inc_list(struct%neighbours(m)%elements(t),q)%vector)
    end do
    sum3=sum3+sum_q2*(1.0/3.0)* &
    area_list(struct%neighbours(m)%elements(t))
  end do
  b_vec(m)=sum3
  rs_vec(m)=sum3
end do
!$omp end do
!$omp end parallel

D_mat=D_mat*k*c*my/im
D_mat_pre=D_mat

call CGESV(n_pairs, 1, D_mat, n_pairs, ipiv, b_vec, n_pairs, info)

temp_r(1)=0.0
temp_r(2)=0.0
do m=1, n_pairs
  temp_c=(0.0,0.0)
  do n=1, n_pairs
    temp_c=temp_c+D_mat_pre(m,n)*b_vec(n)
  end do
  temp_r(1)=temp_r(1)+abs(temp_c-rs_vec(m))
  temp_r(2)=temp_r(2)+abs(rs_vec(m))
end do

if (info .eq. 0)then
```

```
  print *, 'Linear system solved successfully!'
else
  print *, 'An error occured while solving the linear system'
  stop
end if

print *, 'avg rel error:', temp_r(1)/temp_r(2)
!read(*,*)

! Calculating the resulting current densities in all element points
do n=1, n_el
  do q=1,3
    J_gen_quad((n-1)*3+q)%vector &
    =(/(0.0,0.0), (0.0,0.0), (0.0,0.0)/)
  end do
end do

print *, 'J_gen initialized to 0'

do n=1,n_pairs
  do t=1,2
    do q=1,3
      J_gen_quad((struct%neighbours(n)%elements(t)-1)*3+q)%vector &
      =J_gen_quad((struct%neighbours(n)%elements(t)-1)*3+q)%vector &
      +b_vec(n)*f_mat(n,t,q,1:3)
    end do
  end do
end do

print *, 'J_gen_quad calculated'
return
end function

function vec_len(v)
  real               :: v(3), vec_len
  vec_len=sqrt(v(1)**2+v(2)**2+v(3)**2)
  return
end function

end module calculation_mod_omp
```

G.3 Main Program Part 1

The file *J_calc.f90* contains the following FORTRAN code.

```
program J_calc
! Part 1 of the main program calculating the current density J
USE disc_mod
USE calculation_mod_omp
implicit none

! Declaring program variables

! Variables for the output data
integer, parameter :: out_unit=20, in_unit=2

! Variables for the discretization
integer :: n, ne, i, nc, j, l, q, np, nx,ny,nz
type(Point) ,dimension(:), allocatable :: point_list
type(Structure) :: struct
real area_tot

! Variables for the calculation
type(Point), dimension(:), allocatable     :: r_list
type(Vector_c) , dimension(:), allocatable :: H_inc, J_list, &
                                              E_res, E_inc
type(Vector_c) , dimension(3)    :: G_tensor
complex                          :: temp_c, integral
complex, dimension(3)            :: temp_c_vec
complex, dimension(3,3)          :: Greens
real                             :: k, lambda, omega, H_inc_0, &
                                    E_inc_0, temp_r
real , dimension(3)              :: k_vec, H_inc_hat, E_inc_hat, &
                                    k_hat, temp_r_vec
real, dimension(:), allocatable  :: Area_list
integer                          :: M, temp_i, quad_order, &
                                    disc_number
! Constants:
real                             :: my_0, eps_0, y_min, y_max, &
                                    z_min, z_max, x_min, x_max, de
logical                          :: KA

!---------------------------------------
!--------Input Parameters-(begin)-------
!---------------------------------------
```

```

!print *, 'Choose number of discretization points:'
n=258 !(rec:6, 18, 66, 258, 1026, 4098, 16386, 65538, 262146, ...)

!print *,'Choose wavelength in units of sphere radius R:'
lambda=40.0

! Print Choose polarization of incoming EM wave
E_inc_hat=(/ 0.0, 0.0, 1.0 /)

! Choose direction of incoming wave
k_hat=(/ 1.0, 0.0, 0.0/)

! Set quadrature order (1 or 3)
quad_order=3

! Use KA
KA = .false.

! Choose the discretization algorithm which fills struct with data
disc_number=2

!---------------------------------------
!--------Input Parameters-(end)-------
!---------------------------------------

!uncomment if parameters are instead included from input file
!include 'Run_circ03_J.par'

if (disc_number.eq.1)then
  call stripack_convert(struct, n)
else if (disc_number.eq.2)then
  call recursive_triangulation(struct, n)
end if

ne=size(struct%elements)

print *,' '
print *,'List of points:'
do i=1,5 !n
  write (*,"(I5,F10.5,F10.5,F10.5)"), i,struct%points(i)%point(1:3)
end do

```

```
print *, ' '
print *,'List of elements:'
do i=1,5 !ne
  write (*,"(10i5)"), i, struct%elements(i)%corners(1:struct &
  %elements(i)%nr_of_corners)
end do
print *, '      '

allocate(Area_list(ne))
! Calculates area of the elements and fills Area_list with
! the area of the corresponding element
area_tot=0.0
!print *,'Areas:'
do i=1,ne
  nc=struct%elements(i)%nr_of_corners
  allocate(point_list(nc))
  do j=1,nc
    point_list(j)=struct%points(struct%elements(i)%corners(j))
  end do
  temp_r=area(point_list)
  area_tot=area_tot+temp_r
  Area_list(i)=temp_r
  deallocate(point_list)
end do

! Uncomment if area distribution shall be calculated
!call area_dist(struct, disc_number, Area_list)
!read(*,*)

! Prints neighbours:
print *, ' '
print *, 'Printing neighbour elements: '
print *, 'pair nr: ', '      ','elements in pair:', &
'    ','common corners:'
do i=1, 5 !size(struct%neighbours)
  print *, i, struct%neighbours(i)%elements(1:2), &
  struct%neighbours(i)%corners(1:2)
end do

! Prints midpoints
print *, ' '
print *, 'Printing midpoints: '
print *, 'Element nr: ', '    ','Midpoint: '
```

```
do i=1,5 ! ne
  write (*,"(I5,F12.7,F12.7,F12.7)"), &
  i,struct%midpoints(i)%point(1:3)
end do

! Outputs some information
print *, ' '
print *, 'Number of elements: ', ne
print *, 'Total relative area: ', area_tot/(16*atan(1.0))

! Scattering Calculation

! Defining the constants
my_0=4*pi*1E-7
eps_0=1.0/(my_0*c**2)

! Defining Variables
E_inc_0=1.0
k=2.0*pi/lambda
omega=k*c
k_vec=k*k_hat
H_inc_hat=cross_r(k_hat, E_inc_hat)
H_inc_0=E_inc_0/(my_0*c)

! Printing Variable information
write (*, "(a,i5)"),    ' disc_number:           ', disc_number
write (*, "(a,i8)"),    ' discretization points: ', n
write (*, "(a,l4)"),    ' KA: ', KA
write (*, "(a,e12.5)")  ' eps_0=', eps_0
write (*, "(a,e12.5)")  ' my_0=', my_0
write (*, "(a,f9.3)")   ' E_inc_0=', E_inc_0
write (*, "(a,3f9.3)")  ' E_inc_hat=', E_inc_hat
write (*, "(a,e12.5)")  ' H_inc_0=', H_inc_0
write (*, "(a,3f9.3)")  ' H_inc_hat=', H_inc_hat
write (*, "(a,e12.5)")  ' k = ', k
write (*, "(a,3f9.3)")  ' k_hat=', k_hat
write (*, "(a,f9.3, a)")' lambda = ', lambda, 'R'
write (*, "(a,e12.5)")  ' omega = ', omega
write (*,*) ' '

allocate(J_list(quad_order*ne))
if ( KA .eqv. .true. )then
  ! Calculating incoming H-field at scatterer surface (KA approx)
  ! using either 1-point or 3-point Gaussian quadrature method
  if (quad_order.eq.1)then
    allocate(H_inc(ne))
    do i=1, ne
      temp_r=dot_prod_r(k_vec,struct%midpoints(i)%point)
      H_inc(i)%vector=H_inc_0*H_inc_hat*cexp(im*temp_r)
    end do
  else if (quad_order.eq.3)then
    allocate(H_inc(3*ne))
    do i=1,ne
      do j=1,3
        temp_r=dot_prod_r(k_vec,struct%quadpoints(i,j)%point)
        H_inc((i-1)*3+j)%vector=H_inc_0*H_inc_hat*cexp(im*temp_r)
      end do
    end do
  end if
  ! Calculating el current density J_list at surface (KA approx)
  do i=1, ne
    temp_c_vec=norm_vec_c(struct%points(struct%elements(i) &
    %corners(1))%point, &
    struct%points(struct%elements(i)%corners(2))%point, &
    struct%points(struct%elements(i)%corners(3))%point, &
    struct%midpoints(i)%point )
    do j=1,quad_order
      J_list((i-1)*quad_order+j)%vector &
      =2.0*cross_c(temp_c_vec, H_inc((i-1)*quad_order+j)%vector)
    end do
  end do
else
  ! Calculating el curr density J_list at surface (linear system)
  if (quad_order.eq.1)then
    print *, 'Generating current densities by '
    print *, '1st order gauss quadrature integrals...'
    J_list=J_gen(k, k_hat, struct, Area_list, my_0, &
          E_inc_hat*E_inc_0)
  else if (quad_order.eq.3)then
    print *, 'Generating current densities by '
    print *, '3rd order gauss quadrature integrals...'
    J_list=J_gen_quad(k, k_hat, struct, Area_list, my_0, &
          E_inc_hat*E_inc_0)
  else
    print *,'Wrong quadrature order!'
  end if
end if

open (unit=out_unit,file='J_data', action="write",status="replace")
  write (out_unit, *) n, ne, lambda, quad_order, &
  disc_number, E_inc_0
  write (out_unit, *) E_inc_hat, k_hat
  do i=1, quad_order*ne
    write (out_unit, *) J_list(i)%vector
  end do
close(out_unit)
print *, 'Done writing current densities, J, to file.'

end program
```

G.4 Main Program Part 2

The file *E_calc_omp.f90* contains the following FORTRAN code.

```
program E_calc
USE omp_lib
! Program calculating the electric field at a set of
! observation points based on the surface current
! density, J, fetched from the file, 'J_data'.
USE disc_mod
USE calculation_mod_omp
implicit none

! Declaring program parameters
integer, parameter :: out_unit=20, in_unit=2

! Declaring program variables
integer                           :: i, M, nx, ny, nz, ne, n, j, &
                                     l, nc, q, p, quad_order, &
                                     disc_number, counter, p_t, &
                                     ax1, ax2, circ_ax, tot_field
real                              :: y_min, y_max, x_min, x_max, &
                                     z_min, z_max, omega, my_0, &
                                     area_tot, temp_r, E_inc_0, &
                                     E_inc_hat(3), lambda, &
                                     k_vec(3), k_hat(3), k, de, &
                                     circ_r
complex                           :: integral, sum_p
real, dimension(:), allocatable :: Area_list
```

```
type(Vector_c) , dimension(3)   :: G_tensor
type(Point), dimension(:), allocatable     :: r_list
type(Structure)                            :: struct
type(Point) ,dimension(:), allocatable     :: point_list
type(Vector_c) , dimension(:), allocatable :: J_list, &
                                              J_list_quad, &
                                              E_res, E_inc
character (len=40)                         :: file_J


file_J='J_data'

! Reading input parameteres from J_data file
open(unit=in_unit, file=trim(file_J))
read(in_unit, *) n, ne, lambda, quad_order, disc_number, E_inc_0
read(in_unit, *) E_inc_hat, k_hat
close(in_unit)

!----------------------------------------------
!--------Input Parameters-E_calc-(begin)-------
!----------------------------------------------

! Choose type of observation point list
p_t=1 !(1: xy, 2:yz, 3:xz, 4: Circle, 5:Point)

x_min=-40.0
x_max=40.0
y_min=-40.0
y_max=40.0
z_min=0.0
z_max=0.0

! Choose Resolution
de=0.1*lambda

! Choose total or scattered field
tot_field=1.0 !(1 for total, 0 for scattered)

! if p_t =4,5
! Choose circle radius and plane
circ_r=40
circ_ax=2 !(1:xy, 2:xz, 3:yz)

```

```
! if p_t=5

!--------------------------------------
!--------Input Parameters-(end)-------
!--------------------------------------

!uncomment if parameters are included from input file
!include 'Run_circ03_E.par'



my_0=4*pi*1E-7
k=2.0*pi/lambda
omega=k*c
k_vec=k*k_hat

! Choosing discretization algorithm which fills struct
if (disc_number.eq.1)then
  call stripack_convert(struct, n)
else if (disc_number.eq.2)then
  call recursive_triangulation(struct, n)
end if
ne=size(struct%elements)

! Calculates area of the elements and fills Area_list with
! the area of the corresponding element
allocate(Area_list(ne))
area_tot=0.0
do i=1,ne
  nc=struct%elements(i)%nr_of_corners
  allocate(point_list(nc))
  do j=1,nc
    point_list(j)=struct%points(struct%elements(i)%corners(j))
  end do
  temp_r=area(point_list)
  area_tot=area_tot+temp_r
  Area_list(i)=temp_r
  deallocate(point_list)
end do

! Reading current density data from J_data file
open(unit=in_unit, file=trim(file_J))
read(in_unit, *) !n, ne, lambda
read(in_unit, *)
allocate(J_list(quad_order*ne))
do i=1,quad_order*ne
  read(in_unit, *), J_list(i)%vector
end do
close(in_unit)

print *, 'Done reading J_data!'

! Creating list of M observation points

nx=ceiling(1+(x_max-x_min)/de)
ny=ceiling(1+(y_max-y_min)/de)
nz=ceiling(1+(z_max-z_min)/de)

if (p_t < 4)then
  print *,'nx,ny,nz:', nx,ny,nz
        M=nx*ny*nz
        allocate(r_list(M))
end if

if (p_t .eq. 1)then ! xy-plot
  ax1=1
  ax2=2
  do i=1,ny
    do j=1,nx
      r_list((i-1)*nx+j)%point= &
      (/ x_min+de*(j-1), y_min+de*(i-1), z_min /)
    end do
  end do
else if (p_t .eq. 2)then ! yz-plot
  ax1=2
  ax2=3
  do i=1,nz
    do j=1,ny
      r_list((i-1)*ny+j)%point=(/ x_min, y_min+de*(j-1), &
      z_min+de*(i-1) /)
    end do
  end do
else if (p_t .eq. 3)then ! xz-plot
  ax1=1
  ax2=3
  do i=1,nz
```

```fortran
    do j=1,nx
      r_list((i-1)*nx+j)%point=(/ x_min+de*(j-1), y_min, &
      z_min+de*(i-1) /)
    end do
  end do
else if (p_t .eq. 4)then ! Circle
  ! Creates a circle in the plane set in the input file
  M=ceiling(2.0*pi*circ_r*200/lambda)
  allocate(r_list(M))
  if (circ_ax .eq. 1)then !xy
    ax1=1
    ax2=2
      do i=1, M
        r_list(i)%point=(/ circ_r*cos(2.0*pi*(i-1)/(M-1)), &
        circ_r*sin(2.0*pi*(i-1)/(M-1)) , 0.0/)
    end do
  else if (circ_ax .eq. 2)then !xz
    ax1=1
    ax2=3
      do i=1, M
        r_list(i)%point=(/ circ_r*cos(2.0*pi*(i-1)/(M-1)), 0.0, &
        circ_r*sin(2.0*pi*(i-1)/(M-1))/)
      end do
  else if (circ_ax .eq. 3)then !yz
    ax1=2
    ax2=3
    do i=1, M
      r_list(i)%point=(/ 0.0, circ_r*cos(2.0*pi*(i-1)/(M-1)), &
      circ_r*sin(2.0*pi*(i-1)/(M-1)) /)
    end do
  end if
else if (p_t .eq. 5)then ! Single Point
  M=1
  allocate(r_list(M))
  ax1=1
  ax2=2
  r_list(1)%point=(/ x_min, y_min, z_min /)
end if

print *, 'Total number of observation points:', M
print *, 'Calculating E-field amplitude...'

! Creating list of points on a straight line
```

```fortran
!M=2021
!allocate(r_list(M))
!do i=1,(M)
!  r_list(i)%point=(/ 5.0, 0.0, -0.2+(i-1)*0.01/)
!r_list(i+M/2)%point=(/ -1.99-(i-1)*0.01, 0.0, 0.0/)
!end do

! Calculating incoming electric field E_inc at
! all observation points in r_list
allocate(E_inc(M))
do i=1,(M)
  temp_r=dot_prod_r(k_vec,r_list(i)%point)
  E_inc(i)%vector=E_inc_0*E_inc_hat*exp(im*temp_r)
end do



! Calculating Resulting Electric field, E_res,
! in all observation points in r_list
! Quad order = 3
if (quad_order.eq.3)then
allocate(E_res(M))
counter=0

!$omp parallel private ( i,l, j,p,q, G_tensor, sum_p, integral) &
! !$omp reduction ( + : E_res(1)%vector(1)) ) &
!$omp shared ( Area_list, omega, my_0, r_list, ne, k, J_list) &
!$omp shared ( M, quad_order, E_res, counter, struct )

!$omp do
do i=1,M ! for all r
  ! M must here be more than 100. Else: comment out
  if (mod(counter,M/100).eq. 0 .and. counter .ne. 0)then
    write (*, "(i5,a1)")int(ceiling(counter*100.0/M)),'%'
  end if
  if (r_list(i)%point(1)**2+r_list(i)%point(2)**2 &
      +r_list(i)%point(3)**2>1.00)then
    do l=1,3 !for each of the three directions
      integral=(0.0,0.0)
      do j=1,ne !for every element in the discretization
        sum_p=(0.0,0.0)
        do p=1,3 ! for each of the three evaluation points
          do q=1,3
```

```
            G_tensor(l)%vector(q)= &
            G(r_list(i)%point,struct%quadpoints(j,p)%point,l,q,k)
          end do
          sum_p=sum_p+dot_prod_c(G_tensor(l)%vector, &
          J_list((j-1)*3+p)%vector)
        end do
        integral=integral+(1.0/3.0)*sum_p*Area_list(j)
      end do
      E_res(i)%vector(l)=(tot_field*E_inc(i)%vector(l) &
      -(omega*my_0/im)*integral)/E_inc_0
    end do
  else
    E_res(i)%vector=(/ (0.0, 0.0), (0.0,0.0), (0.0,0.0)/)
  end if
  counter=counter+1
end do
!$omp end do
!$omp end parallel
end if
if (quad_order.eq.1)then
! Calculating Resulting Electric field, E_res, in all
! observation points in r_list having when quad order = 1
allocate(E_res(M))
!allocate(point_list(3))
do i=1,M ! for alle r
  ! M must here be more than 100. Else: comment out
  if (mod(i,M/100).eq. 0)then
    write (*, "(i5,a1)")int(ceiling(i*100.0/M)),'%'
  end if
  if (r_list(i)%point(1)**2+r_list(i)%point(2)**2 &
  +r_list(i)%point(3)**2>1.00)then
    do l=1,3 !for each of the three directions
      integral=(0.0,0.0)
      do j=1,ne        !for every element in the discretization
        do q=1,3
          G_tensor(l)%vector(q)=G(r_list(i) &
          %point,struct%midpoints(j)%point,l,q,k)
        end do
        integral=integral+dot_prod_c(G_tensor(l)%vector, &
        J_list(j)%vector)*Area_list(j)
      end do
      E_res(i)%vector(l)=tot_field*E_inc(i)%vector(l) &
      -(omega*my_0/im)*integral
    end do
    else
      E_res(i)%vector=(/ (0.0, 0.0), (0.0,0.0), (0.0,0.0)/)
    end if
end do
end if
print *, 'Done calculating electric field amplitude!'

! Printing the resulting electric field
! amplitude, |E|/|E_0|, to file
open (unit=out_unit,file="results_E.dat",action="write", &
status="replace")
write (out_unit, "(a, i3,a,i8)") 'Output type:', p_t, ', ne: ', ne
write (out_unit, "(I7, f8.3, f9.2)") M, de, lambda
do i=1,M
  !write (*, "(f8.2,f8.2,f9.4)") r_list(i)%point(ax1), &
  !r_list(i)%point(ax2), sqrt(abs(E_res(i)%vector(1))**2 &
  !+abs(E_res(i)%vector(2))**2+abs(E_res(i)%vector(3))**2)
  write (out_unit, "(f8.2,f8.2,e14.7)") r_list(i)%point(ax1),&
  r_list(i)%point(ax2), sqrt(abs(E_res(i)%vector(1))**2 &
  +abs(E_res(i)%vector(2))**2+abs(E_res(i)%vector(3))**2)
end do
close (out_unit)


end program



```

G.5 Compiling the Code

The file *run_prog.sh* contains the following shell script.

```
#!/bin/bash
clear
T="$(date +%s)"

MYFOLDER="/home/rune/Dokumenter/fordypningsprosjekt/program"

# Compiles the modules disc_mod and calculation_mod
gfortran -c "$MYFOLDER/disc_mod.f90"
gfortran -fopenmp -c "$MYFOLDER/calculation_mod_omp.f90"
```

```

# Compiles electrodyn.f90 by using disc_mod.o,
# calculation_mod.o and the library discret
gfortran -fopenmp "$MYFOLDER/J_calc.f90" \
"$MYFOLDER/disc_mod.o" "$MYFOLDER/calculation_mod_omp.o" \
-o  J_calc -L/home/rune/Dokumenter/fordypningsprosjekt/lib/ \
-ldiscret -llapack -O3

#gfortran "$MYFOLDER/E_calc.f90" "$MYFOLDER/disc_mod.o" \
# "$MYFOLDER/calculation_mod.o" \
gfortran -fopenmp "$MYFOLDER/E_calc_omp.f90" \
"$MYFOLDER/disc_mod.o" "$MYFOLDER/calculation_mod_omp.o" \
-o  E_calc -L/home/rune/Dokumenter/fordypningsprosjekt/lib/ \
-ldiscret -llapack -O3

#echo "electrodyn.f90 compiled, saved to electrodyn!"
echo "Running program..."

# Running program electrodyn
"$MYFOLDER/J_calc" > output_J

#running program E_calc
"$MYFOLDER/E_calc" #> output2

echo "Done running program!"
T="$(($(date +%s)-T))"
#echo "Time in seconds: ${T}"
printf " %02d:%02d:%02d\n" "$((T/3600%24))" "$((T/60%60))" "$((T%60))"
```

G.6 Output File

The file *output_J* contains the following example of an output file from the *J_calc* program. (Only parts of the discretization data are printed to the file.)

```
 Creating discretization recursively with #points =          258

 List of points:
     1   1.00000   0.00000   0.00000
     2  -1.00000   0.00000   0.00000
     3   0.00000   1.00000   0.00000
     4   0.00000  -1.00000   0.00000
     5   0.00000   0.00000   1.00000
```

```

List of elements:
   1  67  68  69
   2  70  71  72
   3  73  74  75
   4  76  77  78
   5  79  80  81


Printing neighbour elements:
pair nr:             elements in pair:        common corners:
         1           1         129          67          68
         2           1         130          68          69
         3           1         131          67          69
         4           2         132          70          71
         5           2         133          71          72

Printing midpoints:
Element nr:          Midpoint:
   1  0.5685353   0.5685353   0.5685353
   2 -0.5685353   0.5685353   0.5685353
   3 -0.5685353  -0.5685353   0.5685353
   4  0.5685353  -0.5685353   0.5685353
   5  0.5685353   0.5685353  -0.5685353

Number of elements:          512
Total relative area:   0.98741060
disc_number:               2
discretization points:      258
KA:    F
eps_0= 0.88542E-11
my_0= 0.12566E-05
E_inc_0=    1.000
E_inc_hat=    0.000    0.000    1.000
H_inc_0= 0.26544E-02
H_inc_hat=    0.000   -1.000    0.000
k =  0.15708E+00
k_hat=    1.000    0.000    0.000
lambda =    40.000R
omega =  0.47091E+08

Generating current densities by
3rd order gauss quadrature integrals...
```

```
pair nr         100 of total         768
pair nr         200 of total         768
pair nr         300 of total         768
pair nr         400 of total         768
pair nr         500 of total         768
pair nr         600 of total         768
pair nr         700 of total         768
Linear system solved successfully!
avg rel error:  1.55312464E-05
J_gen initialized to 0
J_gen_quad calculated
Done writing current densities, J, to file.
```

References

[1] M. Moskovits. Surface roughness and the enhanced intensity of raman scattering by molecules adsorbed on metals. *J. Chem. Phys*, 69(9):4159–4161, 1978.

[2] G. Mie. Beiträge zur optik trüber medien, speziell kolloidaler metallösungen. *Ann. Phys.*, 330:377–445, 1908.

[3] A. M. Kern and O. J. F. Martin. Surface integral formulation for 3d simulations of plasmonic and high permittivity nanostructures. *J. Opt. Soc. Am. A*, 26(4):732–740, 2009.

[4] A. A. Maradudin I. Simonsen and T. A. Leskova. Scattering of electromagnetic waves from two-dimensional randomly rough perfectly conducting surfaces: The full angular intensity distribution. *Phys. Rev. A*, 81:013806, 2010.

[5] M. Taskinen I. Hanninen and J. Sarvas. Singularity subtraction integral formulae for surface integral equations with rwg, rooftop and hybrid basis functions. *Prog. Electromagn. Res.*, 63:243–278, 2006.

[6] M. Testorf and M. Fiddy. Kirchhoff's approximation in diffractive optics. In *Diffractive Optics and Micro-Optics*, page 154. Optical Society of America, 2000.

[7] G. R. Cowper. Gaussian quadrature formulas for triangles. *Int. J. Numer. Methods Eng*, 7:405–408, 1973.

[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[9] R. Renka. Delaunay triangulation on a sphere. `http://people.sc.fsu.edu/~jburkardt/f_src/stripack/stripack.html`.

[10] Recursive triangulated unit sphere, stored in numpy.array. https://www.youtube.com/watch?v=DLYBIpj3cKk&feature=player_embedded.

[11] OpenMP Architecture Review Board. OpenMP application program interface. http://www.openmp.org.

[12] L. Novotny and B. Hecht. *Principles of Nano-Optics.* Cambridge University Press, 2006.

[13] D. J. Griffiths. *Introduction to Electrodynamics, Third Edition.* Benjamin Cummings, 2008.

[14] I. Simonsen. Optics of surface disordered systems. *Eur. Phys. J. Special Topics*, 181:46, 2010.

[15] Centre for Atmospheric and Oceanic Sciences. Basics of atmospheric radiation. http://caos.iisc.ernet.in/pub/summer_school/atmospheric_radiation/radiation.pdf.

[16] M. C. Wirth. Symbolic vector and dyadic analysis. *SIAM J. Comput.*, 8(3):306–319, 1979.

[17] A. Chakrabarti. Galerkin methods in solving integral equations with applications to scattering problems. In *International Conference on Mathematical Methods in Electromagnetic Theory*, volume 1, pages 79–87, 1998.

[18] R. Harrington. *Field Computation by Moment Methods.* Wiley-IEEE Press, 1993.

[19] E. W. Weisstein. Linear operator. From *MathWorld*-A Wolfram Web Resource. http://mathworld.wolfram.com/LinearOperator.html.