Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Eilif Sommer Øyre

Electromagnetic Scattering Calculations for Arbitrarily Shaped Closed Surfaces using the Method of Moments

Master's thesis in Applied Physics and Mathematics
Supervisor: Ingve Simonsen
March 2021

NTNU
Norwegian University of Science and Technology

Eilif Sommer Øyre

Electromagnetic Scattering Calculations for Arbitrarily Shaped Closed Surfaces using the Method of Moments

Master's thesis in Applied Physics and Mathematics
Supervisor: Ingve Simonsen
March 2021

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

NTNU
Norwegian University of
Science and Technology

Norwegian University of Science and Technology

TFY4900 Physics, Master Thesis

Electromagnetic Scattering Calculations for Arbitrarily Shaped Closed Surfaces using the Method of Moments

Author:
Eilif Sommer Øyre

Supervisor:
Ingve Simonsen (NTNU)

Faculty of Natural Sciences
Department of Physics
NTNU
07.03.21

Preface

This thesis completes a five year Master's Degree Programme in Applied Physics and Mathematics at the Norwegian University of Science and Technology. It was conducted at the Department of Physics from September 2020 to March 2021.

I would like to express my gratitude to my supervisor Professor Ingve Simonsen, and thank him for his excellent support and guidance. I truly appreciate the learning opportunity provided by him.

Abstract

The Maxwell equations of electromagnetic theory are numerically solved for the two region scattering problem using a surface integral formulation (SIE) and the method of moments (MoM). The RWG basis function is applied to approximate the equivalent currents of the SIEs, and Galerkin's method is used for the weighted residuals. The electric and magnetic field integral equations are combined using the PMCHW-formulation and the resulting matrix equation is solved by LU-decomposition. The numerical methods were implemented using modular programming with an object-oriented approach in modern Fortran, and the numerical framework responsible for representing the discretised surface was designed to be general and versatile, so as to be applicable to scattering surfaces of arbitrary shapes and refractive index, and scattering problems using alternative basis functions and methodology.

The implementation was tested with scattering from a homogeneous sphere and the results were compared to the ones Mie theory. The results from the numerical simulation showed expected interference patterns and symmetric properties, but failed to consistently conserve energy and satisfyingly match the Mie solution. This was due to issues in the implementation believed to be minor, but not found because of time constraints. However, scattering from a single and multiple nonspherical objects was simulated, and the implementation proved successful in reproducing local surface plasmon resonance effects for a gold dipole at the incident resonance wavelength $\lambda = 662$ nm. Moreover, a face-by-face approach in evaluating the surface integrals was implemented, significantly increasing the memory usage, but reducing the computing time by a factor of 20, compared to a basis-by-basis approach.

Sammendrag

I denne oppgaven løses Maxwells likninger numerisk for det elektromagnetiske spredningsproblemet i to regioner ved bruk av overflateintegrallikninger og momentmetoden. For å kunne diskretisere de ekvivalente strømmene i overflateintegrallikningene, blir RWG basisfunksjoner brukt sammen med Galerkins metode for de vektede restene. Ved hjelp av PMCHW-formuleringen kombineres den elektriske og magnetiske integrallikningen, og den resulterende matriselikningen blir løst ved LU-dekomponering. Den numeriske implementasjonen ble utført i moderne Fortran ved bruk av moduler og objekt-orientert programmering. For å kunne bruke implementasjonen til å simulere spredning på en vilkårlig overflate, ble koden som lagrer diskretiseringen av overflaten laget generell og allsidig. På denne måten, vil koden også kunne brukes til å simulere spredning ved bruk av andre metoder og basisfunksjoner.

Den numeriske implementasjonen ble testet ut på spredning av en homogen kule og resultatene ble sammenlignet med de fra Mie teorien. Resultatene viste interferensmønster og symmetriske egenskaper som samsvarte med det vi forventet, men energien i systemet var ikke bevart for flere av eksemplene, og resultatene var ikke tilfredstillende lik de fra Mie teorien. Vi tror dette er grunnet mindre feil i implementasjonen som ikke ble funnet som følge av begrenset tid. Likevel, ble spredningsimuleringer ble utført på ikkesfæriske overflater, og for en innkommende elektromagnetisk bølge med bølgelengde på 662 nm, lykkes implementasjonen med å reprodusere lokalisert overflateplasmonresonans for en dipolantenne av gull. I tillegg ble en flate-for-flate tilnærming for evaluering av overflateintegralene prøvd ut. Den viste seg å bruke betydelig mer minne, med reduserte beregningstiden med en faktor på 20, sammenlignet med en basis-for-basis tilnærming.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Scattering and absorption of electromagnetic (EM) waves by small particles are responsible for many interesting phenomena. The multicoloured, curved rainbow occurs when EM waves at several hundred manometers in wavelength (light) meets much larger rain droplets, having a diameter of micrometers or millimetres. The rainbow may be explained by geometrical optics, using laws of refraction, reflection and dispersion. Scattering and absorption of light by particles much smaller than the wavelength, about size of gas molecules, is responsible for the contrast of the red sunset against the blue sky. This type of scattering is often called Rayleigh scattering [3], and concludes that the intensity of scattered EM waves is proportional to the incident wavelength to the power of four. The incandescent, auroral opal, on the other hand, is explained by the diffraction of light by glass spheres about the same size as the wavelength of the incident light.

Aside from eye-pleasing effects, there is the localised Surface plasmon resonance (LSPR), which is a phenomena caused by the scattering and absorption by metal particles smaller than the wavelength of the incident EM wave [4]. The incident wave interacts with the electrons causing plasmon oscillations inside the particle, which in the spacing between particles may have a resonance frequency allowing several orders of magnification of the scattered field. The resonance frequency depends on the shape and permitivity of the scattering particles, as well as their relative position. Particles of gold and silver have resonance frequency in the visible part of the light spectrum. As an example, Kern and Martin [1] found the resonance wavelength of a gold dipole antenna to be 662 nm. LSPR has many useful applications, such as surface-enhanced Raman spectroscopy, and biosensing using nanoparticles [5, 6].

Naturally, because of the advantages of numerical simulations in the radio- and telecommunication industry, numerical methods for accurate simulation of EM scattering have been greatly research for decades. Simulation of EM scattering is essentially the quest of solving the Maxwell equations. Analytical solutions to the Maxwell equations exists for ideal cases, such as the Mie theory [7, 8], which describes the EM scattering of an incident wave by a homogeneous sphere of arbitrary

radius and refractive index. However, there are no analytical solutions for the scattering by more complex geometries, not to mention multiple particles. As a result, numerical solutions of Maxwell's equations are necessary to simulate and predict phenomena such as LSPR. Fortunately, Maxwell's equations are linear, so that their solutions follow the superposition principle, i.e. the sum of any solution is also a solution. This way, following Fourier analysis, it is sufficient to solve Maxwell's equations for an incident plane wave, as any incoming wave may be represented by a sum of plane waves.

Popular numerical methods for solving Maxwell's equations include finite difference time-domain (FDTD) schemes and the finite element method (FEM). Both methods solves the equations on differential form, thus requiring a discretisation of the whole solution domain, and well defined boundary conditions. Unlike FDTD, FEM is not limited by the need of structured grids, and may be used on unstructured grids consisting of sets of various 2D or 3D elements, such as triangles, tetrahedrons or quadrilaterals. This makes FEM able to deal with highly complex geometries, as well as easily adjust local resolution by constructing finer grids in part of the region of interest. On the other hand, FEM is not as easily applicable to the time domain, and will in general have a higher computational cost [9].

In contrast to FDTD and FEM, the method of moments (MoM), or the boundary element method (BEM), uses an integral formulation of Maxwell's equations. Once the boundary conditions of the problem are fitted into the integral equation, it may be used to find the solution of the corresponding differential equations directly at any desired point in the solution domain, thus requiring discretisation of only the scatterer. Furthermore, using a surface integral formulation (SIE) instead of a volume integral formulation (VIE) or the discrete-dipole approximation (DDA) [10], reduces the discretisation to only the scattering surface, resulting in a significantly lower amount of discretisation points for a large system, compared to FDTD and FEM. On the other hand, the matrix equation emerging by applying MoM is dense, in contrast to the sparse matrix resulting from FDTD and FEM. In this way, although having fewer discretisation points, the MoM may consume considerably more memory.

In addition, the MoM has the additional challenges of singular integrals, which need to be carefully treated in order to avoid numerically inaccurate results. The singularities arises from the combination of using Green's function in deriving the integral formulations and the use of Galerkin's method for the testing integrals in order to discretise them. The SIE consists of the electric field integral equation (EFIE) and the magnetic field integral equation (MFIE). The EFIE has *weakly singular* integrands because of an $1/R$ dependency, where R, the source-to-observation distance, may be close to or equal to zero. The MFIE has integrands proportional to $1/R^3$, often called *hyper singular* integrands. Numerical evaluations of surface integrals over these singularities will of course be undefined if $R = 0$, and if R is close to zero, values may be too large to be accurately represented by the computer (especially the hyper singularity).

Be that as it may, a triangular patch approach in approximating a scattering surface is extremely versatile, being able to accurately represent complex topologies

and higher order curvature [11]. Moreover, efficient and accurate techniques for handling the singular integrals exists, such as singularity subtraction [12–14], singularity cancellation [15–17], Duffy's transformation [18], semi-analytic schemes [19], and even fully numerical methods [20].

Thus, when using triangular basis functions to discretise the SIE, such as the Rao-Wilton-Glisson (RWG) basis [11,21], and an appropriate singularity treatment, MoM with SIE become very suitable and efficient to simulate EM scattering by arbitrary surfaces, and in particular when investigating near-field effects such as LSPR. In contrast, the DDA is limited by having to evaluate the field distribution at least one dipole separation away from the boundary.

1.2 Objective

The goal of this thesis is to implement an efficient and versatile numerical framework for simulating EM scattering from one or multiple surfaces with arbitrary shape and refraction index, by using the MoM with an SIE formulation of Maxwell's equations. The framework, created by modular programming in Fortran with an object-oriented approach, should be extendable with any suitable basis function, test function and combined field formulation. However, in this thesis we have the objective of discretising the SIE using Galerkin's method and the RWG basis functions, as well as combining the EFIE and MFIE with the PMCHW-formulation [22]. The singularities of the Green's functions are subtracted and evaluated analytically using the formulas presented in Ref. [14], while the non singular terms are numerically integrated using Gaussian quadrature formulas.

The numerical framework should include a representation of the surface mesh in an suitable data structure, such that commonly required mesh properties and calculations are easily accessible and computationally efficient. Additionally, the framework should support the import of externally generated meshes, and be easily extendable to various file formats.

An additional goal is to implement an overall computational efficient solution, and investigate opportunities to reduce the amount of integral evaluations to a minimum.

Chapter 2

Theory and Method

Evaluating the two region electromagnetic scattering problem in light of Maxwell's equations presents us with a set of partial differential equations (PDEs). As discussed in the previous chapter, the goal of this project is to implement a simulation program to numerically solve these PDEs using the method of moments (MoM). Therefore, the following chapter starts by introducing the scattering problem and deriving these PDEs, which are then reformulated into the electric field integral equation (EFIE), and the magnetic field integral equation (MFIE), with the help of Green's function. The integral formulation of the PDEs is required for the MoM to be applied, which is introduced next, in section 2.2. The MoM expands the integral equations in a series of basis functions, mapped over a discretised surface. In this project, we will be using the Rao-Wilton-Glisson (RWG) basis functions (introduced in section 2.2.3), which consists of triangular elements. It is therefore natural to briefly present some theory on differential geometry, in particular triangulation. The application of MoM together with Galerkin's method of weighted residuals, results in a matrix equation whose matrix elements are double integrals over the chosen basis functions. Thus, a suitable numerical integration method is introduced (Gaussian quadrature) in Section 2.4.

The value of comparing the results of the numerical simulation using MoM with the results of an analytical solution can not be underestimated. Therefore, we give a brief presentation on the Mie theory, an analytic solution to the two region electromagnetic scattering problem (i.e. an analytic solution to the aforementioned PDEs) for a spherical and homogeneous scatterer of arbitrary radius and refraction index.

Finally, in Section 2.6, the MoM and Gaussian quadrature are put into action. They are applied directly to the EFIE and MFIE, and the resulting matrix equations are combined. Then, the numerical difficulty of integrating over the Green's function's singularities are discussed and treated, before looking into a simple way of reducing the computational cost of evaluating the numerous double integrals appearing in the matrix equations.

2.1 The Two Region Scattering Problem

The three independent equations of Maxwell's theory of electromagnetism in differential form reads [23]

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \qquad \text{(Faraday's law)}, \tag{2.1}$$

$$\nabla \times \mathbf{H} = \mathbf{j} + \frac{\partial \mathbf{D}}{\partial t} \qquad \text{(Maxwell-Ampere law)}, \tag{2.2}$$

$$\nabla \cdot \mathbf{j} = \mathbf{j} - \frac{\partial \boldsymbol{\rho}}{\partial t} \qquad \text{(Continuity equation)}, \tag{2.3}$$

where $\mathbf{E}$ is the electric field, $\mathbf{B}$ is the magnetic flux density, $\mathbf{H}$ is the magnetic field, $\mathbf{D}$ is the electric flux density, $\mathbf{j}$ is the electric current density, and $\boldsymbol{\rho}$ is the electric charge density. In a medium with zero polarisation and magnetisation, an isotropic medium, the flux densities are

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{2.4}$$

$$\mathbf{B} = \mu \mathbf{H}, \tag{2.5}$$

where ϵ and μ are the permitivity and permeability of the medium, respectively. Assuming the fields are time-harmonic, i.e. having form $\mathbf{U} = \mathbf{U_0} \exp\{-i\omega t\}$ where ω is the angular frequency of the wave, and the medium is isotropic, Eqs. (2.1) and (2.2) may be rewritten as

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

$$\nabla \times \mathbf{B} = \mu \mathbf{j} - i\mu\epsilon\omega \mathbf{E}. \tag{2.7}$$

Taking the curl of Eqs. (2.6) and (2.7) yields the PDEs

$$\nabla \times \nabla \times \mathbf{E} - k^2 \mathbf{E} = i\mu\omega \mathbf{j}, \tag{2.8}$$

$$\nabla \times \nabla \times \mathbf{H} - k^2 \mathbf{H} = \nabla \times \mathbf{j}, \tag{2.9}$$

where $k = \omega\sqrt{\epsilon\mu}$ is the wavenumber of the electromagnetic wave. These differential equations are often described as the *inhomogeneous vector wave equations* [23].

Consider the two region problem illustrated in Fig. 2.1. Region 1, having symbol V_1, is bounded from inside by region 2, having symbol V_2 and creating the surface S, and bounded from outside by the surface S_{inf}. The incident electric $\mathbf{E}_1^{\text{inc}}(\mathbf{r})$ and magnetic fields $\mathbf{H}_1^{\text{inc}}(\mathbf{r})$ are generated by the electric current density $\mathbf{j}(\mathbf{r})$ in V_1. Region 2 is assumed to be non-emitting, i.e. $\mathbf{E}_2^{\text{inc}}(\mathbf{r}) = 0$. The regions have different permitivity and permeability, and thus also wavenumber, ϵ_i, μ_i, and $k_i = \omega\sqrt{\epsilon_i\mu_i}$, respectively. As follows, the electric and magnetic field, $\mathbf{E}_i(\mathbf{r})$ and $\mathbf{H}_i(\mathbf{r})$, must satisfy Eqs. (2.8) and (2.9) in both regions i, in addition to the boundary conditions

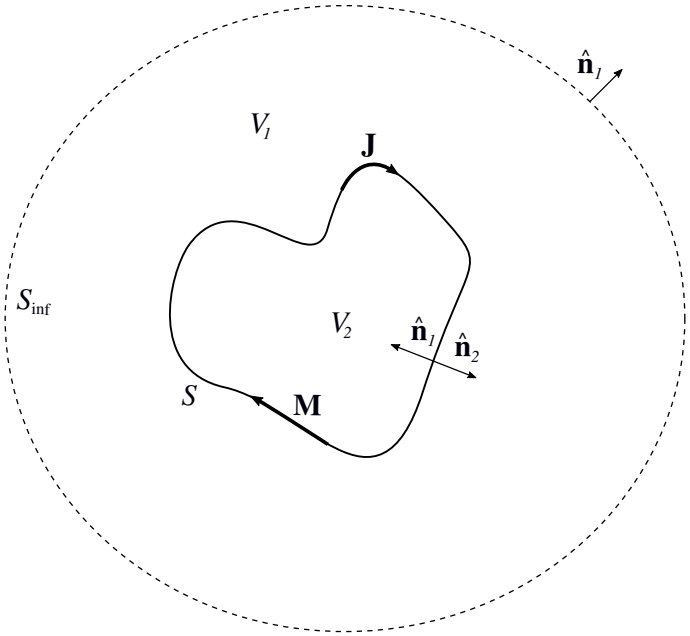

Figure 2.1: Illustration of the two region scattering problem. Region 1 and 2 are denoted as V_1 and V_2, respectively. $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are normal vectors pointing out of the surfaces bounding the regions. $\mathbf{J}$ and $\mathbf{M}$ are equivalent surface currents. Figure inspired by [1].

enforced on the surface S, which tells us that the field components tangential to the surface should be continuous

$$\hat{\mathbf{n}}_i(\mathbf{r}) \times [\mathbf{E}_1(\mathbf{r}) - \mathbf{E}_2(\mathbf{r})] = 0, \tag{2.10a}$$

$$\hat{\mathbf{n}}_i(\mathbf{r}) \times [\mathbf{H}_1(\mathbf{r}) - \mathbf{H}_2(\mathbf{r})] = 0, \quad \text{for } \mathbf{r} \in S, \tag{2.10b}$$

as long as S is not a perfect conductor. If region 2 was a perfect conductor, Eq. (2.10b) would not equal zero, but equal to a term involving the surface current on S [24]. The vectors $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$ are the unit normal vectors to the surface S, pointing out from region 1 and 2, respectively.

The two-region electromagnetic scattering problem boils down to solving the set of PDEs

$$\nabla \times \nabla \times \mathbf{E}_1 - k_1^2 \mathbf{E}_1 = i\mu_1 \omega \mathbf{j}, \quad \text{for } \mathbf{r} \in V_1 \tag{2.11a}$$

$$\nabla \times \nabla \times \mathbf{E}_2 - k_2^2 \mathbf{E}_2 = i\mu_2 \omega \mathbf{j}, \quad \text{for } \mathbf{r} \in V_2 \tag{2.11b}$$

$$\nabla \times \nabla \times \mathbf{H}_1 - k_1^2 \mathbf{H}_1 = \nabla \times \mathbf{j}, \quad \text{for } \mathbf{r} \in V_1 \tag{2.11c}$$

$$\nabla \times \nabla \times \mathbf{H}_2 - k_2^2 \mathbf{H}_2 = \nabla \times \mathbf{j}, \quad \text{for } \mathbf{r} \in V_2, \tag{2.11d}$$

satisfying the conditions (2.10). The Method of Moments (MoM) (discussed in Section 2.2), uses the surface integral formulation of Eq. (2.11) on the boundary S to numerically solve the PDEs. Thus, in Section 2.1.2, we give a short presentation

of the surface integral formulation of Eq. (2.11), which is derived by the use of a dyadic Green's function.

2.1.1 Green's function

Consider the differential equation

$$L[f] = s. \tag{2.12}$$

Assume L is a linear differential operator, s is a source distribution, and the solution f is a field. The Green's function is defined such that it satisfies

$$LG(\mathbf{r}, \mathbf{r}') = \delta^3(\mathbf{r} - \mathbf{r}'), \tag{2.13}$$

where L acts on the observation point $\mathbf{r}$, and $\delta^3(\mathbf{r} - \mathbf{r}')$ is the three dimensional Dirac delta distribution. Consider the integral

$$L \int G(\mathbf{r}, \mathbf{r}')s(\mathbf{r}')\mathrm{d}V'. \tag{2.14}$$

Since L is a linear operator on $\mathbf{r}$ it may be moved inside the integral and acted on $G(\mathbf{r}, \mathbf{r}')$, allowing integration over the Dirac delta distribution

$$\int LG(\mathbf{r}, \mathbf{r}')s(\mathbf{r}')\mathrm{d}V' = \int \delta^3(\mathbf{r} - \mathbf{r}')s(\mathbf{r}')\mathrm{d}V' = s(\mathbf{r}). \tag{2.15}$$

Hence, the solution of (2.12) is given on integral form in terms of its Green's function as

$$f(\mathbf{r}) = \int G(\mathbf{r}, \mathbf{r}')s(\mathbf{r}')\mathrm{d}V'. \tag{2.16}$$

Essentially, the Green's function represents the contribution from the source point $\mathbf{r}'$, to the field observed at point $\mathbf{r}$.

The free-space Green's function

Let the linear differential operator take the form $L = \nabla^2 + k^2$, where k is the wavenumber, and let f be a scalar field and s be a constant source term, such that Eq. (2.12) becomes

$$\nabla^2 f + k^2 f = g. \tag{2.17}$$

This is the *inhomogeneous scalar Helmholtz equation* or *inhomogeneous scalar wave equation* [9, 23]. The Green's function related to the integral equation (2.16) for the Helmholtz equation is called the *free-space* Green's function

$$G(\mathbf{r}, \mathbf{r}') = \frac{e^{ikR}}{4\pi R}, \tag{2.18}$$

where

$$R = |\mathbf{r} - \mathbf{r}'|. \tag{2.19}$$

The free-space dyadic Green's function

A dyadic function is formed by two vector functions, i.e. it is a rank 2 tensor. In itself, a dyad has no physical interpretation, but the result of acting it upon another vector function may be meaningful.

Consider the differential equations (2.8) and (2.9), which both are on the form of Eq. (2.12). Consider next a infinitesimal source current pointed in the $\hat{\mathbf{x}}$-direction

$$\mathbf{j}(\mathbf{r}) = \frac{1}{i\mu\omega}\delta^3(\mathbf{r}-\mathbf{r}')\hat{\mathbf{x}}. \tag{2.20}$$

Let $\mathbf{G}^{(x)}(\mathbf{r},\mathbf{r}')$ be the *free-space vector* Green's function for a field contribution at $\mathbf{r}$ by a source current pointed in the $\hat{\mathbf{x}}$-direction at $\mathbf{r}'$. The function $\mathbf{G}^{(x)}(\mathbf{r},\mathbf{r}')$ should then satisfy

$$\nabla\times\nabla\times\mathbf{G}^{(x)}(\mathbf{r},\mathbf{r}') - k^2\mathbf{G}^{(x)}(\mathbf{r},\mathbf{r}') = \delta^3(\mathbf{r}-\mathbf{r}')\hat{\mathbf{x}}, \tag{2.21}$$

in addition to the Sommerfield radiation condition at $R\to\infty$, i.e. that the energy is only radiating outward from the source. In the same way, we may introduce Green's functions for infinitesimal source currents pointed in the $\hat{\mathbf{y}}$- and $\hat{\mathbf{z}}$-direction, $\mathbf{G}^{(y)}(\mathbf{r},\mathbf{r}')$ and $\mathbf{G}^{(z)}(\mathbf{r},\mathbf{r}')$, respectively. Next, we define the dyadic function $\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')$ to consist of the three vector functions

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \mathbf{G}^{(x)}(\mathbf{r},\mathbf{r}')\hat{\mathbf{x}} + \mathbf{G}^{(y)}(\mathbf{r},\mathbf{r}')\hat{\mathbf{y}} + \mathbf{G}^{(z)}(\mathbf{r},\mathbf{r}')\hat{\mathbf{z}}, \tag{2.22}$$

such that it will satisfy

$$\nabla\times\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') - k^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \overline{\mathbf{1}}\delta^3(\mathbf{r}-\mathbf{r}'), \tag{2.23}$$

where the dyad

$$\overline{\mathbf{1}} = \hat{\mathbf{x}}\hat{\mathbf{x}} + \hat{\mathbf{y}}\hat{\mathbf{y}} + \hat{\mathbf{z}}\hat{\mathbf{z}} \tag{2.24}$$

is called the *idem factor*. Using vector identity (11) on the back cover of [24], we may rewrite Eq. (2.23) as

$$\nabla\left[\nabla\cdot\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right] - \nabla^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') - k^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \overline{\mathbf{1}}\delta^3(\mathbf{r}-\mathbf{r}'). \tag{2.25}$$

Taking the divergence of Eq. (2.23) yields

$$\nabla\cdot\left(\nabla\times\nabla\times\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right) - \nabla\cdot\left[k^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')\right] = \nabla\cdot\left[\overline{\mathbf{1}}\delta^3(\mathbf{r}-\mathbf{r}')\right] \tag{2.26}$$

$$\nabla\cdot\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = -\frac{1}{k^2}\nabla\cdot\left[\overline{\mathbf{1}}\delta^3(\mathbf{r}-\mathbf{r}')\right] \tag{2.27}$$

$$\nabla\cdot\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = -\frac{1}{k^2}\nabla\left[\delta^3(\mathbf{r}-\mathbf{r}')\right], \tag{2.28}$$

where in Eq. (2.27) we have used that the divergence of the curl is zero, and in Eq. (2.28) we have used the following property of the idem factor [23]

$$\nabla\cdot(\overline{\mathbf{1}}\psi) = \nabla\psi. \tag{2.29}$$

By inserting Eq. (2.28) in Eq. (2.25) we get

$$\nabla^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') + k^2\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = -\left(\overline{\mathbf{1}} + \frac{\nabla\nabla}{k^2}\right)\delta^3(\mathbf{r}-\mathbf{r}'). \tag{2.30}$$

Letting $\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')$ have the form

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \left(\overline{\mathbf{1}} + \frac{\nabla\nabla}{k^2}\right)\psi(\mathbf{r},\mathbf{r}'), \tag{2.31}$$

then $\psi(\mathbf{r},\mathbf{r}')$ should satisfy

$$\nabla^2\psi(\mathbf{r},\mathbf{r}') + k^2\psi(\mathbf{r},\mathbf{r}') = -\delta^3(\mathbf{r}-\mathbf{r}'), \tag{2.32}$$

which we recognise as the form of Eq. (2.17) with solution $G(\mathbf{r},\mathbf{r}')$ (2.18). Thus, the *free-space dyadic Green's function* is

$$\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') = \left(\overline{\mathbf{1}} + \frac{\nabla\nabla}{k^2}\right)G(\mathbf{r},\mathbf{r}'). \tag{2.33}$$

2.1.2 The Surface Integral Equations

This section presents a summary of the derivation of the surface integral equation for the electrical and magnetic field for the two region scattering problem. See [1] for a full, formal derivation.

The free-space dyadic Green's function $\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')$ (see Eq. (2.33)) satisfies Eq. (2.23) and is therefore a suitable Green's function for both regions i. Multiplying Eq. (2.11a) or Eq. (2.11b) by $\overline{\mathbf{G}}(\mathbf{r},\mathbf{r}')$ from the right hand side, Eq. (2.23) by $\mathbf{E}_i(\mathbf{r})$ from the left hand side, and subtracting the resulting expressions, gives

$$\begin{aligned}\nabla\times\nabla\times\mathbf{E}_i(\mathbf{r})\cdot\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r})\cdot\nabla\times\nabla\times\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \\ = i\mu_i\omega\mathbf{j}(\mathbf{r})\cdot\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') - \mathbf{E}_i(\mathbf{r})\delta^3(\mathbf{r}-\mathbf{r}').\end{aligned} \tag{2.34}$$

Here we have used properties of dyadic function to assert

$$\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{E}_i(\mathbf{r}) = \mathbf{E}_i(\mathbf{r})\cdot\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')^T = \mathbf{E}_i(\mathbf{r})\cdot\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}'), \tag{2.35}$$

$$\mathbf{E}_i(\mathbf{r})\cdot\overline{\mathbf{1}} = \mathbf{E}_i(\mathbf{r}). \tag{2.36}$$

Integrating Eq. (2.34) over V_i and using Gauss' theorem yields surface integrals, which, deduced from the radiation condition, is over the surface S (see equations 4-10 in [1]). Defining the equivalent surface current densities

$$\mathbf{J}(\mathbf{r}') = \hat{\mathbf{n}}_2\times\mathbf{H}_i(\mathbf{r}'), \tag{2.37}$$

$$\mathbf{M}(\mathbf{r}') = -\hat{\mathbf{n}}_2\times\mathbf{E}_i(\mathbf{r}'), \tag{2.38}$$

and regarding the limit $\mathbf{r} \to S$, at which from the boundary conditions (2.10) we know that the tangential components are continuous, we obtain

$$\begin{aligned}\left[\frac{\omega\mu_i}{i}\int_S \mathrm{d}S'\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{J}(\mathbf{r}') - \int_S \mathrm{d}S'[\nabla'\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\cdot\mathbf{M}(\mathbf{r}')\right]_{\tan} \\ = \begin{cases}[\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r})]_{\tan}\,, & i=1\\ 0, & i=2,\end{cases}\end{aligned} \tag{2.39}$$

where the subscript *tan* indicates the vector component tangential to the boundary surface S. This surface integral equation is called the electric field integral equation (EFIE). Similar treatment with Eqs. (2.11c) and (2.11d) leads to the magnetic field integral equation

$$\begin{aligned}\left(\frac{\omega\epsilon_i}{i}\int_S \mathrm{d}S'\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{M}(\mathbf{r}') - \int_S \mathrm{d}S'[\nabla'\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\cdot\mathbf{J}(\mathbf{r}')\right)_{\tan} \\ = \begin{cases}[\mathbf{H}_1^{\mathrm{inc}}(\mathbf{r})]_{\tan}\,, & i=1\\ 0, & i=2.\end{cases}\end{aligned} \tag{2.40}$$

2.2 Method of Moments

As discussed in Chapter 1, both finite element method (FEM) and MoM are used to numerically solve partial differential equations. FEM solves the equations on differential form, while MoM solves them on integral form. Nevertheless, the recipe for MoM follows somewhat that of FEM, which is presented first.

2.2.1 Finite Element Analysis

The differential equation to solve by FEM has the form of Eq. (2.12)

$$L[f] = s,$$

where L is a linear differential operator, f is the sought after solution function to be found in the domain Ω, and s is a known function. The unknown function f is approximated by expanding it in a series of basis functions $f_1, f_2, \ldots, f_N$

$$f \approx \sum_{n=1}^{N} \alpha_n f_n, \tag{2.41}$$

where α_n are expansion coefficients, and N is the number of basis functions. For our solution, we want the residual

$$r = L[f] - s \tag{2.42}$$

as small as possible for an arbitrary position vector $\mathbf{r}'$, and set the *tested* (or *weighted*) residual equal to zero

$$\langle w_m, r\rangle = \langle w_n, L[f]\rangle - \langle w_n, s\rangle = 0, \tag{2.43}$$

where w_m are called testing functions, and $\langle w_m, r\rangle$ defines a suiting inner product. By inserting the expansion (2.41) into Eq. (2.43), and since L is a linear operator we get

$$\sum_{n=1}^{N} \alpha_n \langle w_m, L[f_n]\rangle = \langle w_m, s\rangle . \tag{2.44}$$

This may be written on matrix form

$$\mathbf{Ax} = \mathbf{b}, \tag{2.45}$$

where

$$A_{mn} = \langle w_m, L[f_n]\rangle \tag{2.46}$$
$$x_n = \alpha_n \tag{2.47}$$
$$b_m = \langle w_m, n\rangle , \tag{2.48}$$

for $m, n = 1, \ldots, N$. The solution to the differential equation (2.12) may then be found by solving (2.45), and its accuracy depends on the choice of basis functions f_n and testing functions w_m [25]. Choosing $w_m = f_m$ is known as *Galerkin's method* [9, 25].

2.2.2 Integral Formulation

The source distribution $s(\mathbf{r}')$ may be expanded in N basis functions $s_n(\mathbf{r}')$

$$s(\mathbf{r}') = \sum_{n=1}^{N} \alpha_n s_n(\mathbf{r}'), \tag{2.49}$$

where α_n are expansion coefficients. Using the Green's function, linear differential equation (2.12) may be written as a integral equation on the form of Eq. 2.16. Using Eq. (2.49) in addition yields the approximated solution

$$\bar{f}(\mathbf{r}) = \sum_{n=1}^{N} \alpha_n f_n(\mathbf{r}), \tag{2.50}$$

where

$$f_n(\mathbf{r}) = \int G(\mathbf{r}, \mathbf{r}') s_n(\mathbf{r}') \mathrm{d}S'. \tag{2.51}$$

The goal is to minimise the weighted residual $\left\langle w_m, [f(\mathbf{r}) - \bar{f}(\mathbf{r})]\right\rangle$ on the boundary C, where $f(\mathbf{r})$ is known. Setting the residual equal to zero and inserting Eq. (2.50) leads to the equation

$$\int w_m f(\mathbf{r}) \mathrm{d}S = \int w_m \bar{f}(\mathbf{r}) \mathrm{d}S = \sum_{n=1}^{N} \alpha_n \int \mathrm{d}S w_m \int \mathrm{d}S' G(\mathbf{r}, \mathbf{r}') s_n(\mathbf{r}'), \tag{2.52}$$

which on matrix form (2.45) has the elements

$$A_{mn} = \int \mathrm{d}S w_m \int \mathrm{d}S' G(\mathbf{r}, \mathbf{r}') s_n(\mathbf{r}') \tag{2.53}$$

$$x_n = \alpha_n \tag{2.54}$$

$$b_m = \int \mathrm{d}S w_m f(\mathbf{r}), \tag{2.55}$$

where m and n takes on the values $1 = 1, 2, \ldots, N$. The field $f(\mathbf{r})$ may then be found at any desired observation point $\mathbf{r}$ using the solutions α_n

$$f(\mathbf{r}) \approx \bar{f}(\mathbf{r}) \sum_{n=1}^{N} \alpha_n \int \mathrm{d}S' G(\mathbf{r}, \mathbf{r}') s_n(\mathbf{r}'). \tag{2.56}$$

Choosing weighting functions equal to the basis functions is, as mentioned above, called Galerkin's method, while taking the weighting functions equal to Dirac delta functions $w_m(\mathbf{r}) = \delta(\mathbf{r} - \mathbf{r}_m)$ is known as *point matching* [9].

2.2.3 Rao-Wilton-Glisson Basis functions

The RWG basis, introduced by Glisson [21] and named after Rao, Wilton and Glisson [11, 26], is a commonly used basis function for solving electromagnetic scattering problems with the MoM. In particular, it is used to expand surface current densities by a linear combination of basis functions. This surface patch approach assumes a triangulated surface S where each basis function is associated with an *interior* edge, that is, an edge that is not a boundary edge (see Section 2.3 on triangulation). An RWG basis S_n comprises the two adjacent triangles (faces) $T_n^{\pm}$ that share the basis edge (see Fig. 2.2), and vanishes elsewhere on the surface. The basis function is defined by

$$\mathbf{f}_n(\mathbf{r}') = \begin{cases} \dfrac{\pm L_n}{2A_n^{\pm}}(\mathbf{r}' - \mathbf{p}_n^{\pm}), & \mathbf{r}' \in T_n^{\pm} \\ 0, & \text{otherwise,} \end{cases} \tag{2.57}$$

where L_n is the length of the associated edge, $A_n^{\pm}$ is the area of the pair of faces $T_n^{\pm}$, and $\mathbf{p}_n^{\pm}$ are the free vertices of these faces, i.e. the vertices not in common with the basis edge. Following this definition, the surface current on a basis is flowing from the positive face towards the negative. The normal component of the current vanishes on all edges of the faces except on the shared basis edge, where the normal component is constant and normalised to unity such that its value is continuous across the two adjacent faces. In this way, having continuous normal components of the current, the basis function assures that no line charges reside on any edges of $T_n^{\pm}$. Each face has current flowing normal to all three of its edges from three different bases, unless of course, any of its edges are boundary edges.

Finally, the current $\mathbf{J}$ on the triangulated surface S may be approximated as

$$\mathbf{J}(\mathbf{r}') = \sum_{n=1}^{N} \alpha_n \mathbf{f}_n(\mathbf{r}'), \tag{2.58}$$

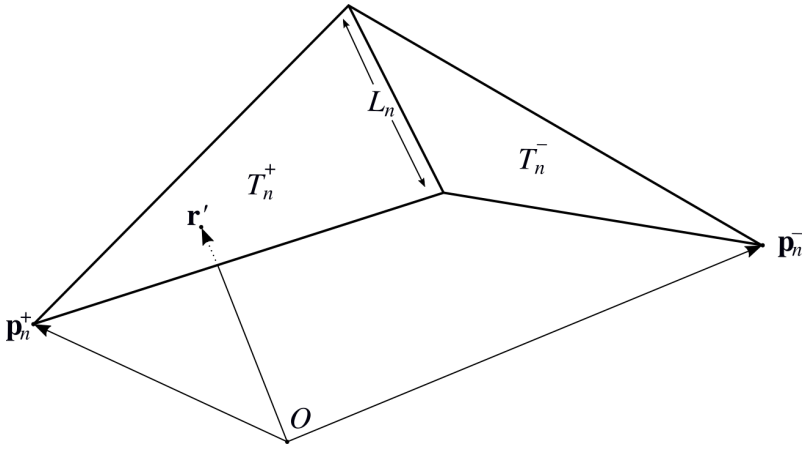

Figure 2.2: Sketch of an arbitrary nth RWG basis. The two adjacent triangles T_n^+ and T_n^-, having areas A_n^+ and A_n^-, share an edge of length L_n. The vectors $\mathbf{p}_n^\pm$ are the positions of the free vertices of $T_n^\pm$ with respect to the origin O, while $\mathbf{r}'$ is a source point in T_n^+ with respect to the same origin.

where N is the number of interior edges on S, and α_n are expansion coefficients.

Another commonly used basis function, called *rooftop* [13], uses rectangular elements instead of triangles. The elements share an edge, similarly to the RWG basis functions, hence the name rooftop. However, rectangular elements may only model curvature of one dimension, and planar triangular element models are thus particularly suited for approximating arbitrary shaped surfaces.

2.3 Triangulation

A triangulated surface is a discretised surface whose points are joined together by line segments in such a way that the surface is completely covered in triangular elements. This way, the points and line segments are the vertices and edges of the triangles, respectively. In other words, the convex hull of the set of points that represent the discretised surface is covered with 2-simplices. In general, a triangulation may also be a covering-up by other orders of simplices, e.g. a discretised volume may be represented by 3-simplices (tetrahedrons). If the line segments of a triangulation are linear, then every point represents the vertex of one or more triangles. Furthermore, a triangulation may be *structured* or *unstructured*, referring to the distribution of the points (see Fig. 2.3), and it may be composed of conformal or non-conformal triangles, referring to whether edges may intersect outside discretisation points (see Fig. 2.4). Delaunay triangulation [27] is a common method for triangulating unstructured surfaces. Its criterion states that the open discs that circumscribes any of its triangular elements should not contain any other vertex. Following this criterion there exists a unique triangulation for each set of points [27], assuming the points are in general position. A Delaunay triangulation may be constructed from a Voronoï-diagram [27], by joining the vertices belonging to adjacent cells.

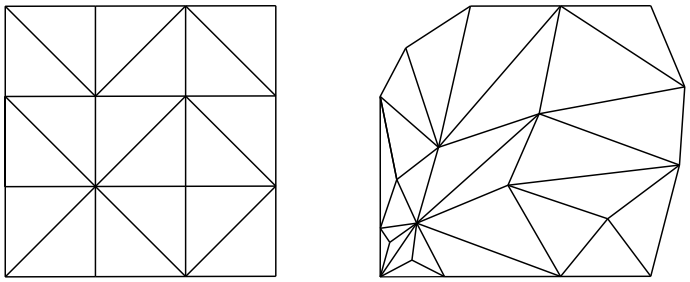

Figure 2.3: A arbitrary triangulation of a structured grid (left). An arbitrary triangulation of an unstructured grid (right).

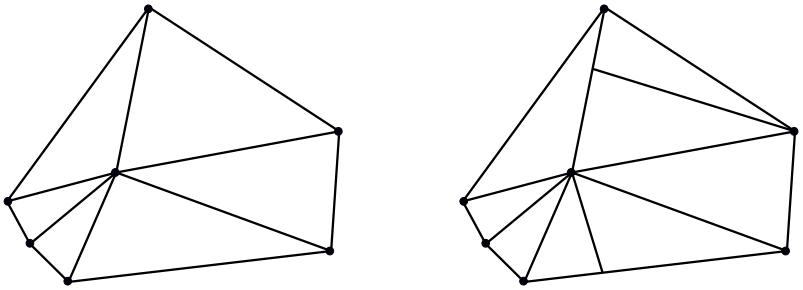

Figure 2.4: A conformal triangulation (left), and a non-conformal triangulation (right). Circular points represents vertices, and line segments represents triangle edges.

2.3.1 Topological Properties of a Triangulated Surface

Let N_v, N_e, and N_f be the number of vertices, edges and triangle elements (faces) on a triangulated surface, respectively. Then the number

$$X = N_v - N_e + N_f \tag{2.59}$$

is called the *Euler-Poincaré characteristic* [28]. For a tetrahedron, with 4 vertices, 6 edges, and 4 faces, this number equals 2. In fact, the Euler-Poincaré characteristic equals 2 for all triangulated surfaces that have the same topology as the tetrahedron. That is, all triangulated surfaces that are closed (they lack boundary edges and apertures) and have no handles, or holes (see Fig. 2.5). The triangulated surface of a torus on the other hand, has a single handle, and its Euler-Poincaré characteristic equals 0. For a double torus, it is -2. Hence, we may conclude the following equation for triangulations of closed surfaces, or *polyhedrons* [26]

$$2(1 - N_h) = N_v - N_e + N_f, \tag{2.60}$$

where N_h is the number of handles.

We may also develop an Euler-Poincaré characteristic for open surfaces. Consider the triangulated surface in Fig. 2.5. It has a single aperture, and a set of boundary edges conforming it. Drawing an auxiliary vertex, suspended above the aperture, and auxiliary edges, joining the auxiliary vertex from the vertices on the boundary,

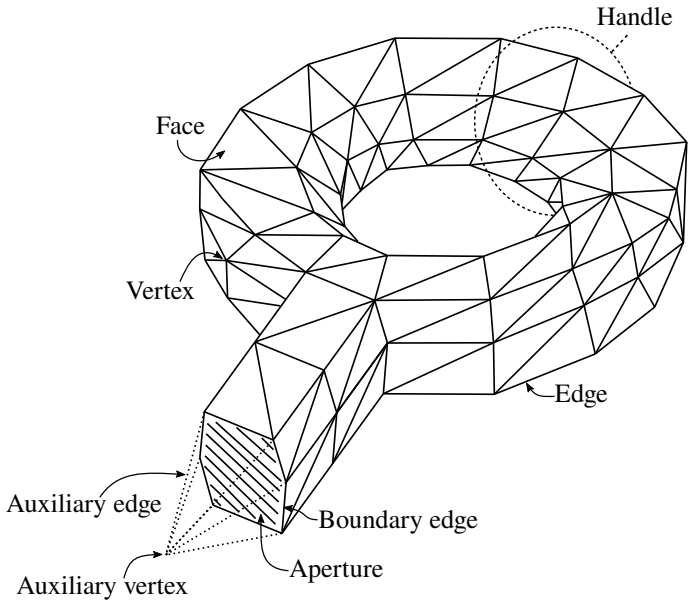

Figure 2.5: A triangulated surface having a single handle and a single aperture.

we may easily conclude the following relations

$$N_v = N'_v + N'_a \tag{2.61}$$
$$N_e = N'_e + N'_b \tag{2.62}$$
$$N_f = N'_f + N'_b, \tag{2.63}$$

where the prime marks an open triangulated surface. Furthermore, N'_a and N'_b are the numbers of apertures and boundary edges on the surface, respectively. Substituting the equations above into Eq. (2.60) yields the Euler-Poincaré characteristic for an open surface

$$2(1 - N_h) - N'_a = N'_v - N'_e + N'_f. \tag{2.64}$$

Now, for a triangulation on a closed surface, each triangle has three edges, and each edge is shared by two triangles, giving $N_e = 3N_f/2$. This may be used to eliminate N'_f in Eq. (2.64), giving

$$N'_e = 3N'_v + 3N'_a - 6(1 - N_h) - N'_b, \tag{2.65}$$

which is a useful expression as the number of unknowns in a finite element analysis may equal the number of edges. When using the MoM with RWG basis functions (see section 2.2) each edge, except boundary edges, constitute a basis. Thus the number of unknowns is $N'_e - N'_b$. Eq. (2.65) is valid for closed surfaces by setting $N'_a = N'_b = 0$, giving N_e unknowns when using RWG basis functions and MoM.

2.4 Gaussian Quadrature

Gaussian quadrature is a numerical integration technique that approximates an integral of a weighted function, $W(x)f(x)$, by a linear combination of function evalu-

ations, $f(x_j)$, at non-uniformly spaced abscissa locations [29],

$$\int_a^b W(x)f(x)\mathrm{d}x \approx \sum_{j=1}^{N} w_j f(x_j). \tag{2.66}$$

Let $p_N(x)$ be part of the set of orthogonal polynomials $\{p_0, \ldots, p_N\}$. The abscissas x_j of the Gaussian quadrature above are equal to the roots of the orthogonal polynomial $p_N(x)$ with the same weighting function $W(x)$ on the same interval. Here p_N has N distinct roots, which is used together with $W(x)$, the interval $[a, b]$, and the rest of the set of polynomials to find the weights w_j. This arrangement gives Gaussian quadrature formulas a degree of precision $2N - 1$ (proof is given in [30]), which means they integrate polynomials of degree up to $2N - 1$ exactly (if we can neglect round-off errors).

2.4.1 Gauss-Legendre Quadrature Formula

The Gauss-Legendre quadrature formula has weight function $W(x) = 1$ on the interval $[-1, 1]$. The set of orthogonal polynomials is the Legendre polynomials,

$$p_j(x) = \frac{1}{2^j j!}\frac{\mathrm{d}^j}{\mathrm{d}x}[(x^2 - 1)^j]. \tag{2.67}$$

Thus, the integral of the function $f(x)$ on the interval $[-1, 1]$ can be approximated by

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \sum_{j=1}^{N} w_j f(x_j), \tag{2.68}$$

where x_j are the roots of the Legendre polynomials (2.67), and w_j are the corresponding weights. Both roots and weights for Gauss-Legendre Quadrature formulas are well tabulated (e.g. pg. 276 in [30]).

2.4.2 Gaussian Quadrature Formulas for Triangles

Numerical integration becomes significantly more challenging as the number of variables increases. Both because the number of evaluation points necessary to achieve decent accuracy increases enormously, and because multidimensional boundaries are much more complicated than the one-dimensional limits. In this project however, there is the need of evaluating integrals over 2-simplices, i.e. triangles, located in three-dimensional space.

Fortunately, highly efficient and numerically accurate Gaussian quadrature formulas for triangles exists [31, 32]. These formulas are of the form

$$\int_T f\mathrm{d}S \approx A\sum_{j=1}^{N} w_j f(\xi_j, \eta_j, \zeta_j), \tag{2.69}$$

where the surface integral of the function $f(\mathbf{r})$, evaluated over the triangle T with area A, is approximated by a sum of N weighted function evaluations at quadrature

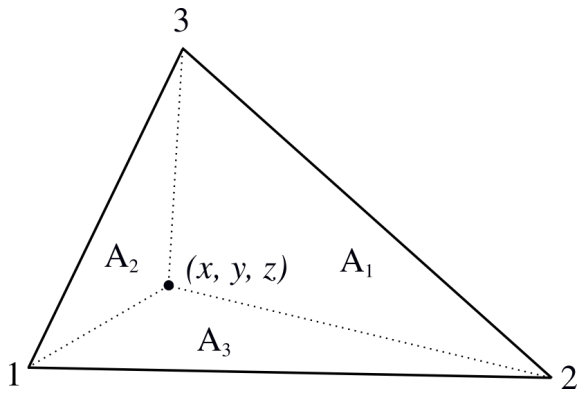

Figure 2.6: A triangle whose surface is split into three elements of area A_1, A_2, and A_3. The borders of the elements are defined by the Cartesian coordinates of the point on the triangle, and by the location of its three vertices, as illustrated.

points given by barycentric coordinates (ξ, η, ζ). The weights w_j are all proportional to the triangle area, and in the above expression the area is factored out of the sum. The sampling points (ξ_j, η_j, ζ_j) are arranged symmetrically within the triangle, and their values and corresponding weights are tabulated for degree of precision up to 7 (13 point formula) in [32].

Barycentric Coordinates

Let the Cartesian point (x, y, z) be on a triangle of area A. Split the surface into three elements with areas A_1, A_2, and A_3, as shown in Fig. 2.6. The barycentric coordinates are defined as the following ratios

$$\xi = \frac{A_1}{A}, \eta = \frac{A_2}{A}, \zeta = \frac{A_3}{A}. \tag{2.70}$$

As $A_1 + A_2 + A_3 = A$, barycentric coordinates have the following constraints

$$1 = \xi + \eta + \zeta, \qquad 0 < \xi < 1, \quad 0 < \eta < 1, \quad 0 < \zeta < 1. \tag{2.71}$$

Thus, there are two independent coordinates, and it can be shown that a surface integral over a triangle may be transformed into a double integral of the form [11]

$$\int_T f \mathrm{d}S = 2A \int_0^1 \int_0^{1-\eta} f(\xi, \eta) \mathrm{d}\xi \mathrm{d}\eta, \tag{2.72}$$

where ζ has been eliminated by the use of the constraint

$$\zeta = 1 - \xi - \eta. \tag{2.73}$$

A point $\mathbf{r}$ in Cartesian coordinates on the face of a triangle is related to barycentric coordinates of the triangle as

$$\mathbf{r} = \xi \mathbf{r}_1 + \eta \mathbf{r}_2 + \zeta \mathbf{r}_3, \tag{2.74}$$

where $\mathbf{r}_i$ is the position of the ith vertex of the triangle, oriented as shown in Fig. 2.6.

2.4.3 Lebedev Quadrature

Lebedev quadrature is a numerical integration method for approximating integrals over a unit sphere. Let A a unit sphere centred in origo, and let $f(\mathbf{r})$ be a function that varies on it. The surface integral of $f(\mathbf{r})$ over A is then

$$\int_A f \mathrm{d}S \approx 4\pi \sum_{j=1}^{N} w_j f(\theta_j, \phi_j), \tag{2.75}$$

where w_j is the jth weight of N, and θ and ϕ are the azimuthal and polar angles, respectively, of the quadrature points. The approximation easily scales to spheres of arbitrary radius, R, such that, in Cartesian coordinates, Eq. (2.75) becomes

$$4\pi R^2 \sum_{j=1}^{N} w_j f(x_j, y_j, z_j), \tag{2.76}$$

where

$$\begin{aligned} x_j &= R\sin(\theta_j)\cos(\phi_j) \\ y_j &= R\sin(\theta_j)\sin(\phi_j) \\ z_j &= R\cos(\theta_j). \end{aligned}$$

Fortran routines for computing the quadrature angles (θ_j, ϕ_j) and their corresponding weight w_j up to an order of 5870 is given by Burkardt [33].

2.5 Mie Theory

The *Mie Theory* is the common term used for the exact solution to the problem of scattering and absorption of electromagnetic waves by a homogeneous sphere of arbitrary radius and refractive index. It was developed by Gustav Mie in 1908 [7], but was also investigated by Peter Debye at the same time, and Lorentz is believed to be the actual first discoverer [8].

The theory uses separation of variables to separate the solution of the *homogeneous scalar wave equation*

$$\nabla^2\psi + k^2\psi = 0, \tag{2.77}$$

in spherical coordinates, into a radial and angular parts, $\psi(r,\theta,\phi) = R(r)\Theta(\theta)\Phi(\phi)$. The vector function

$$\mathbf{M} = \nabla \times (\mathbf{r}\psi), \tag{2.78}$$

where $\mathbf{r}$ is the radius vector, is then found to satisfy the *homogeneous vector wave equation* in spherical coordinates by taking the double curl and subtracting the term $k^2\mathbf{M}$ from both sides

$$\begin{aligned} \nabla \times \nabla \times \mathbf{M} - k^2\mathbf{M} &= \nabla \times \Big[\nabla \times \nabla \times (\mathbf{r}\psi) - k^2\mathbf{r}\psi\Big] \\ = -\nabla^2\mathbf{M} - k^2\mathbf{M} &= \nabla \times \Big[-\mathbf{r}(\nabla^2\psi + k^2\psi)\Big] = 0. \end{aligned} \tag{2.79}$$

Here, we have used vector identity (11) on the back cover of Griffiths [24], and the fact that the divergences $\nabla \cdot \mathbf{M} = \nabla \cdot \psi = 0$. Introducing the second vector function

$$\mathbf{N} = \frac{\nabla \times \mathbf{M}}{k}, \tag{2.80}$$

which also satisfies the homogeneous vector wave equation

$$\nabla^2 \mathbf{N} + k^2 \mathbf{N} = 0, \tag{2.81}$$

we see that $\mathbf{M}$ and $\mathbf{N}$ have the same properties as the electromagnetic field satisfying the homogeneous version of the differential equations (2.8) and (2.9)

$$\nabla^2 \mathbf{E} + k^2 \mathbf{E} = 0, \tag{2.82}$$

$$\nabla^2 \mathbf{H} + k^2 \mathbf{H} = 0. \tag{2.83}$$

Using the solution of ψ, $\mathbf{E}$ and $\mathbf{H}$ may be expanded in an infinite series of the resulting vector spherical harmonics $\mathbf{M}$ and $\mathbf{N}$. The expansion for an incident plane wave is derived by Bohren and Huffman [8] along with the scattered electric and magnetic field, both inside and outside the sphere.

In addition to the extensive uses of the Mie theory when studying scattering from spherical or nearly spherical homogeneous particles, it is very convenient to use as reference for testing numerical programs whose purpose is to simulate scattering and absorption from particles of non-spherical geometries. A scattering simulation using MoM or finite element method should yield approximately the same result for a spherical and homogeneous scatterer as an implementation of the analytical solution of the Mie theory.

2.6 MoM on the Two Region Scattering Problem

In this section, the numerical methods discussed in the previous sections are applied to the two region electromagnetic scattering problem. We start by applying the MoM with RWG basis functions to the EFIE and MFIE. Then, we look at a formulation combining both fields into a single matrix equation. Next, a method for treating the singularities occurring in the Green's function is applied. Then, we present the direct formula for evaluating the resulting electric and magnetic field distribution using the solution of the matrix equation. Following this, we demonstrate a useful parameterisation of line integrals over triangle edges, and finally, advantageous for computational efficiency, we derive expressions for evaluating single and double integral over RWG bases using a face-by-face approach.

2.6.1 Expanding EFIE and MFIE in terms of RWG basis functions

Having approximated the scattering surface between region 1 and 2 (see Fig. 2.1) by an appropriate closed surface triangulation S, one may proceed by approximating

the surface current densities on S. The EFIE (Eq. (2.37)) and $\mathbf{M}(\mathbf{r}')$ (Eq. (2.38)) both depend on the equivalent electric and the equivalent magnetic surface current densities, $\mathbf{J}(\mathbf{r}')$ (2.37) and $\mathbf{M}(\mathbf{r}')$ (2.38) respectively, Both currents may be expanded as a linear combination of the RWG basis function (2.57)

$$\mathbf{J}(\mathbf{r}') \approx \sum_{n=1}^{N} \alpha_n \mathbf{f}_n(\mathbf{r}'), \tag{2.84}$$

$$\mathbf{M}(\mathbf{r}') \approx \sum_{n=1}^{N} \beta_n \mathbf{f}_n(\mathbf{r}'). \tag{2.85}$$

This leads to two sets of expansion coefficients α_n and β_n, resulting in $2N$ unknowns, where N is the number of RWG basis functions on S. Inserting these expansions into the EFIE (2.39) yields

$$\begin{aligned}\left[\sum_{n=1}^{N}\left(\alpha_n \frac{\omega\mu_i}{i}\int_{S_n} \mathrm{d}S' \overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{f}_n(\mathbf{r}') - \beta_n \int_{S_n} \mathrm{d}S'[\nabla'\times\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\cdot\mathbf{f}_n(\mathbf{r}')\right)\right]_{\tan} \\ = \begin{cases}\left[\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r})\right]_{\tan}, & i=1\\ 0, & i=2.\end{cases}\end{aligned} \tag{2.86}$$

Applying Galerkin's method of weighted residuals, i.e. setting the weighted residual to zero using the basis function $\mathbf{f}_m(\mathbf{r})$ as testing function gives

$$\begin{aligned}\int_{S_m} \mathrm{d}S\mathbf{f}_m(\mathbf{r})\cdot\sum_{n=1}^{N}\Bigg(\alpha_n\frac{\omega\mu_i}{i}\int_{S_n}\mathrm{d}S'\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')\cdot\mathbf{f}_n(\mathbf{r}') \\ -\beta_n\int_{S_n}\mathrm{d}S'[\nabla'\times\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')]\cdot\mathbf{f}_n(\mathbf{r}')\Bigg) \\ = \begin{cases}\int_{S_m}\mathrm{d}S\mathbf{f}_m(\mathbf{r})\cdot\mathbf{E}_1^{\mathrm{inc}}(\mathbf{r})\mathrm{d}S, & i=1\\ 0, & i=2,\end{cases}\end{aligned} \tag{2.87}$$

where S_m and $\mathbf{f}_m(\mathbf{r})$ are the basis and basis function associated with the observation point $\mathbf{r}$. The subscript indicating the tangential component have been omitted since the basis functions are always tangential to the boundary surface S. Next, assuming that both regions of the scattering problem are homogeneous, we may express the dyadic Green's function $\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$ in terms of the scalar Green's function $G_i(\mathbf{r},\mathbf{r}')$ as in Eq. (2.33). The double integral present in the first term on the left hand side of

Eq. (2.87) becomes

$$\begin{aligned}
&\int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' \overline{\mathbf{G}}(\mathbf{r},\mathbf{r}') \cdot \mathbf{f}_n(\mathbf{r}') \\
&\quad = \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' \left(\frac{\nabla\nabla}{k_i^2} + \bar{\mathbf{1}} \right) G_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{f}_n(\mathbf{r}') \\
&\quad = \frac{1}{k_i^2} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \nabla\nabla \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}') \\
&\qquad + \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}').
\end{aligned} \tag{2.88}$$

The first term on the right hand side of this equation may be transformed to [1,12,13]

$$\begin{aligned}
&\frac{1}{k_i^2} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \nabla\nabla \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}') \\
&\quad = -\frac{1}{k_i^2} \int_{S_m} \mathrm{d}S [\nabla \cdot \mathbf{f}_m(\mathbf{r})] \int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}').
\end{aligned} \tag{2.89}$$

Following Ref. [23] we have that

$$\nabla' \times \overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') = [\nabla' G_i(\mathbf{r},\mathbf{r}')] \times \bar{\mathbf{1}} \tag{2.90}$$

and since $\bar{\mathbf{1}} \cdot \mathbf{f}_n(\mathbf{r}') = \mathbf{f}_n(\mathbf{r}')$, the double integral in Eq. (2.87) involving the curl of the dyadic Green's function may be transformed as

$$\begin{aligned}
&\int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' [\nabla' \times \overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')] \cdot \mathbf{f}_n(\mathbf{r}') \\
&\quad = \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' [\nabla' G_i(\mathbf{r},\mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}').
\end{aligned} \tag{2.91}$$

In this way, Eq. (2.87) may be rewritten in the matrix form

$$\begin{bmatrix} \boldsymbol{\mathcal{D}}^{(1)} & \boldsymbol{\mathcal{K}}^{(1)} \\ \boldsymbol{\mathcal{D}}^{(2)} & \boldsymbol{\mathcal{K}}^{(2)} \end{bmatrix} \boldsymbol{\psi} = \mathbf{q}^E, \tag{2.92}$$

where $\boldsymbol{\psi}$ is the set of unknown expansion coefficients

$$\boldsymbol{\psi} = [\alpha_1, \alpha_2, \ldots, \alpha_{N-1}, \alpha_N, \beta_1, \beta_2, \ldots, \beta_{N-1}, \beta_N]^T, \tag{2.93}$$

the mth element (q_m^E) of the right hand side vector q_m^E is given by

$$q_m^E = \begin{cases} \displaystyle\int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}), & m = 1, 2, \ldots, N \\ 0, & m = N+1, N+2, \ldots, 2N, \end{cases} \tag{2.94}$$

and finally the matrix elements are given by

$$\begin{aligned}\mathcal{D}_{mn}^{(i)} &= \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \mathbf{f}_n(\mathbf{r}') \\ &\quad - \frac{1}{k_i^2} \int_{S_m} \mathrm{d}S [\nabla \cdot \mathbf{f}_m(\mathbf{r})] \int_{S_n} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}'),\end{aligned} \tag{2.95a}$$

$$\mathcal{K}_{mn}^{(i)} = \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' [\nabla' G_i(\mathbf{r}, \mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}'), \tag{2.95b}$$

for $m, n = 1, 2, \ldots, N$.

Similarly, starting from the MFIE (Eq. (2.40)) we get the matrix equation

$$\begin{bmatrix} \boldsymbol{\mathcal{K}}^{(1)} & \frac{1}{Z_1^2} \boldsymbol{\mathcal{D}}^{(1)} \\ \boldsymbol{\mathcal{K}}^{(2)} & \frac{1}{Z_2^2} \boldsymbol{\mathcal{D}}^{(2)} \end{bmatrix} \boldsymbol{\psi} = \mathbf{q}^H, \tag{2.96}$$

with

$$Z_i = \sqrt{\frac{\mu_i}{\epsilon_i}}, \tag{2.97}$$

and

$$q_m^H = \begin{cases} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{H}_1^{\mathrm{inc}}(\mathbf{r}), & m = 1, 2, \ldots, N \\ 0, & m = N+1, N+2, \ldots, 2N. \end{cases} \tag{2.98}$$

2.6.2 Combining EFIE and MFIE

When the surfaces S_m and S_n in Eq. (2.95) fully overlap, i.e. when $m = n$, the basis functions $\mathbf{f}_m(\mathbf{r})$ and $\mathbf{f}_n(\mathbf{r}')$ are parallel and the cross product in Eq. (2.95b) is parallel to $\mathbf{f}_m(\mathbf{r})$, giving $\mathcal{K}_{mn}^{(i)} = 0$ at any value of the gradient of the Green's function. The inner integrals are then poorly tested.

Combining the MFIE and EFIE will improve the accuracy of the results. An approach is the PMCHW-formulation [22]

$$\boldsymbol{\mathcal{H}} \boldsymbol{\psi} = \mathbf{q}, \tag{2.99}$$

where

$$\boldsymbol{\mathcal{H}} = \begin{bmatrix} \boldsymbol{\mathcal{D}}^{(1)} + \boldsymbol{\mathcal{D}}^{(2)} & -\boldsymbol{\mathcal{K}}^{(1)} - \boldsymbol{\mathcal{K}}^{(2)} \\ \boldsymbol{\mathcal{K}}^{(1)} + \boldsymbol{\mathcal{K}}^{(2)} & \frac{1}{Z_1^2} \boldsymbol{\mathcal{D}}^{(1)} + \frac{1}{Z_2^2} \boldsymbol{\mathcal{D}}^{(2)} \end{bmatrix}, \tag{2.100}$$

and

$$q_m = \begin{cases} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}) \mathrm{d}S, & m = 1, 2, \ldots, N \\ \int_{S_{m-N}} \mathbf{f}_{m-N}(\mathbf{r}) \cdot \mathbf{H}_1^{\mathrm{inc}}(\mathbf{r}) \mathrm{d}S, & m = N+1, N+2, \ldots, 2N. \end{cases} \tag{2.101}$$

The coefficients ψ may be found by solving the matrix equation (2.99) with a dense solver, such as the direct method of LU-decomposition [30], or an suitable iterative method. The matrix elements (2.95) may be calculated with an appropriate numerical method, such as Gaussian quadrature for triangles (see Section 2.4.2), which is demonstrated in Section 2.6.6. However, the Green's functions are singular at $\mathbf{r} = \mathbf{r}'$, which needs to be dealt with for the Gaussian quadrature formulas to give accurate results.

2.6.3 Singularity extraction of the Green's function

The scalar Green's function $G_i(\mathbf{r}, \mathbf{r}')$ and its gradient $\nabla' G_i(\mathbf{r}, \mathbf{r}')$ are both singular at $\mathbf{r} = \mathbf{r}'$. In particular, when using the MoM, singularity occurs if the basis function of the testing integral overlaps with the basis function associated with the source point. Numerical methods to solve this singularity problem exists, such as Duffy's transformation [18]. Duffy's transformation however, applies to $1/R$ singularities, and is only accurate for sufficiently regular triangles [12]. The gradient of the Green's function do in fact introduce a $1/R^3$ singularity. Another approach, based on singularity subtraction, is regarded as the most successful approach [12, 13]. In this section, we will show that subtracting terms from the Green's function will leave a smoothed, regularised part, which may be integrated numerically, and a singular part, the subtracted terms, that may be evaluated analytically on closed-form. This method will also improve the accuracy of integral evaluations Eq. (2.95) in the nearly singular case, when S_n is close to S_m [1, 12].

Subtracting terms from the Green's function

The scalar Green's function for homogeneous media may be expanded using the Taylor series of the exponential function [34]

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{2.102}$$

to become

$$G_i(\mathbf{r}, \mathbf{r}') = \frac{e^{ik_i R}}{4\pi R} = \frac{1}{4\pi}\left(\frac{1}{R} + ik_i - \frac{k_i^2 R}{2} - \frac{ik_i^3 R^2}{6} + \dots\right). \tag{2.103}$$

Subtracting only the singular term (the first term of the right hand side) is not sufficient to yield a smooth function, because the derivative of the Green's function is still discontinuous at $R = 0$ (a consequence of R being an absolute value). Thus, to achieve an accurate evaluation of the integrals numerically, all odd terms must be subtracted. With a small $k_i R$ however, the higher powers of $k_i R$ are negligible, and it is sufficient to subtract only the first and third term, leaving the terms:

$$\begin{aligned} G_i(\mathbf{r}, \mathbf{r}') &= \left[G_i(\mathbf{r}, \mathbf{r}') - \frac{1}{4\pi}\left(\frac{1}{R} - \frac{k_i^2 R}{2}\right)\right] + \frac{1}{4\pi}\left(\frac{1}{R} - \frac{k_i^2 R}{2}\right) \\ &= G_i^s(\mathbf{r}, \mathbf{r}') + \frac{1}{4\pi}\left(\frac{1}{R} - \frac{k_i^2 R}{2}\right), \end{aligned} \tag{2.104}$$

where the smoothed scalar Green's function is

$$G_i^s(\mathbf{r},\mathbf{r}') = \frac{e^{ik_iR}-1}{4\pi R} + \frac{k_i^2 R}{8\pi}. \tag{2.105}$$

This regularised function is nonsingular everywhere and have a well-defined derivative. Its limit as R approaches zero is found by l'Hôpital's rule

$$\lim_{R\to 0} G_i^s(\mathbf{r},\mathbf{r}') = \frac{ik_i}{4\pi}. \tag{2.106}$$

Before looking the gradient of the smoothed Green's function, we evaluate the gradient of R:

$$\nabla' R = -\nabla R = \nabla'\sqrt{(x-x')^2+(y-y')^2+(z-z')^2} = -\frac{1}{R}(\mathbf{r}-\mathbf{r}') = -\hat{\mathbf{R}}, \tag{2.107}$$

where we have used the following definitions

$$\mathbf{R} = \mathbf{r}-\mathbf{r}', \tag{2.108}$$

$$\hat{\mathbf{R}} = \frac{\mathbf{R}}{R}. \tag{2.109}$$

The gradient of the original Green's function is then

$$\begin{aligned}\nabla' G_i(\mathbf{r},\mathbf{r}') &= \frac{e^{ik_iR}}{4\pi R}\left(ik_i - \frac{1}{R}\right)\nabla' R \\ &= G_i(\mathbf{r},\mathbf{r}')\left(\frac{1}{R} - ik_i\right)\hat{\mathbf{R}},\end{aligned} \tag{2.110}$$

such that the gradient of the smoothed Green's function becomes

$$\begin{aligned}\nabla' G_i^s(\mathbf{r},\mathbf{r}') &= \nabla' G_i(\mathbf{r},\mathbf{r}') - \frac{1}{4\pi}\left(\frac{1}{R^3} + \frac{k_i^2}{8\pi R}\right)(\mathbf{r}-\mathbf{r}'), \\ &= \frac{1}{4\pi}\left[\frac{e^{ik_iR}}{R}\left(\frac{1}{R} - ik_i\right) - \frac{1}{R^2} - \frac{k_i^2}{2}\right]\hat{\mathbf{R}}.\end{aligned} \tag{2.111}$$

Finally, by creating a common denominator and using l'Hôpital's rule, the limit of $\nabla' G_i^s(\mathbf{r},\mathbf{r}')$ as R approaches zero is

$$\lim_{R\to 0} \nabla' G_i^s(\mathbf{r},\mathbf{r}') = \frac{ik_i^3}{12\pi}. \tag{2.112}$$

Integrals over the regularised functions $G_i^s(\mathbf{r},\mathbf{r}')$ and $\nabla' G_i^s(\mathbf{r},\mathbf{r}')$ may be evaluated using a numerical method such as Gaussian Quadrature, while integrals over the subtracted terms may be evaluated analytically on closed-forms.

Inner integral over the subtracted terms

Above we suggested the following replacements in Eq. (2.95) when $G_i(\mathbf{r},\mathbf{r}')$ and $\nabla' G_i(\mathbf{r},\mathbf{r}')$ become singular or nearly singular

$$G_i(\mathbf{r},\mathbf{r}') \longrightarrow G_i^s(\mathbf{r},\mathbf{r}') + \frac{1}{4\pi}\left(\frac{1}{R} - \frac{k_i^2 R}{2}\right), \tag{2.113a}$$

$$\nabla' G_i(\mathbf{r},\mathbf{r}') \longrightarrow \nabla' G_i^s(\mathbf{r},\mathbf{r}') + \frac{1}{4\pi}\left(\nabla'\frac{1}{R} - \frac{k_i^2}{2}\nabla' R\right). \tag{2.113b}$$

The resulting double integrals over the smoothed Green's function and its continuous gradient may, as mentioned above, be numerically calculated using double Gaussian quadrature. The inner integrals of the subtracted terms however, which have the form

$$X_1^q(T_n^\pm) := \int_{T_n^\pm} R^q \mathrm{d}S', \tag{2.114}$$

$$\mathbf{X}_2^q(T_n^\pm) := \int_{T_n^\pm} R^q(\mathbf{r}' - \mathbf{p}_n^\pm)\mathrm{d}S', \tag{2.115}$$

$$\mathbf{X}_4^q(T_n^\pm) := \int_{T_n^\pm} [\nabla' R^q] \times (\mathbf{r}' - \mathbf{p}_n^\pm)\mathrm{d}S' \tag{2.116}$$

are still singular for $q = -1$. Here the vector $(\mathbf{r}' - \mathbf{p}_n^\pm)$ originates from the RWG basis function $\mathbf{f}_n(\mathbf{r}')$ (2.57). Fortunately, using iterative methods, these integrals may be solved exactly [12–14].

The analytical solution uses coordinates in a local, orthogonal coordinate system (u, v, w) (See Fig. 2.7). Here $\partial^{(i)}T$, $i = 1, 2, 3$, are the edges of the triangular element T, such that the ith edge is opposite vertex i. The triangle vertices in Cartesian coordinates, $\mathbf{r}_i'$, have the transformed coordinates,

$$\mathbf{r}_1' = (0, 0, 0), \tag{2.117}$$

$$\mathbf{r}_2' = (l_3, 0, 0), \tag{2.118}$$

$$\mathbf{r}_3' = (u_3, v_3, 0), \tag{2.119}$$

where l_i is the length of edge $\partial^{(i)}T$. The formulas for u_3 and v_3 in terms of the triangle vertices in the Cartesian coordinate system are given below, where A is the area of T,

$$u_3 = (\mathbf{r}_3' - \mathbf{r}_1') \cdot \frac{(\mathbf{r}_2' - \mathbf{r}_1')}{l_3}, \tag{2.120}$$

$$v_3 = \frac{2A}{l_3}. \tag{2.121}$$

The observation and source point have the transformed coordinates

$$\mathbf{r} = (u_0, v_0, w_0) \tag{2.122}$$

$$\mathbf{r}' = (u', v', 0), \tag{2.123}$$

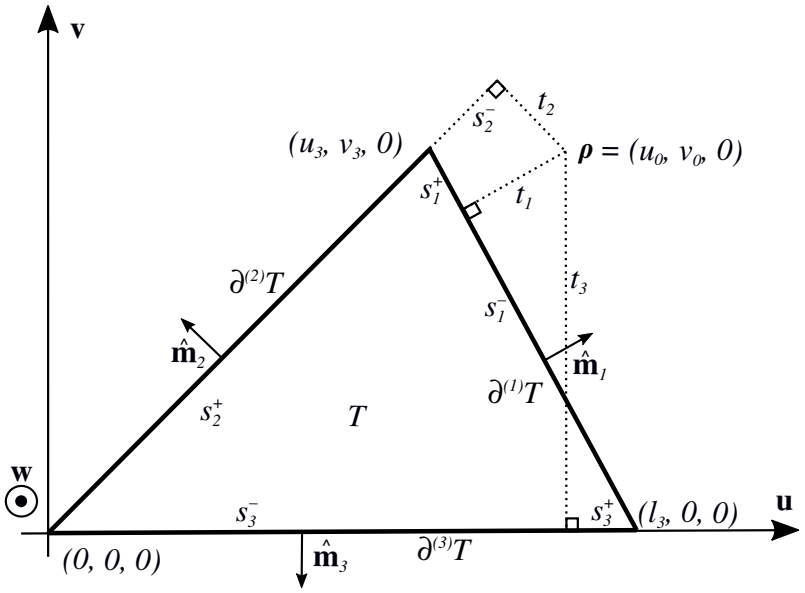

Figure 2.7: The triangle T in the local coordinate system (u, v, w). The edges $\partial^{(i)}T$ of T have lengths length l_i, and $\hat{\mathbf{m}}_i$ are their corresponding outer unit normals. The vector $\boldsymbol{\rho}$ is the projection of the observation point $\mathbf{r}$ onto the uv-plane.

where

$$u_0 = \mathbf{u} \cdot (\mathbf{r} - \mathbf{r}_1'), \tag{2.124}$$

$$v_0 = \mathbf{v} \cdot (\mathbf{r} - \mathbf{r}_1'), \tag{2.125}$$

$$w_0 = \hat{\mathbf{n}} \cdot (\mathbf{r} - \mathbf{r}_1'), \tag{2.126}$$

$$\mathbf{u} = \frac{\mathbf{r}_2' - \mathbf{r}_1'}{l_3}, \tag{2.127}$$

$$\mathbf{v} = \hat{\mathbf{n}} \times \mathbf{u}, \tag{2.128}$$

and $\hat{\mathbf{n}}$ is the unit normal vector of T. Furthermore, u' and v' are arbitrary points in T. The projection of $\mathbf{r}$ onto the transformed triangle is defined by w_0

$$\boldsymbol{\rho} = \mathbf{r} - w_0 \hat{\mathbf{n}}. \tag{2.129}$$

The coordinates $s_i^{\pm}$ and t_i are given in Eq. (2.135).

In the transformed coordinate system, the line and surface integrals

$$I_q^l(\partial^{(i)}T) = \int_{\partial^{(i)}T} R^q \mathrm{d}l', \tag{2.130}$$

$$I_q^S(T) = \int_T R^q \mathrm{d}S', \tag{2.131}$$

have the following solutions [13, 14]

$$I^l_{-1}(\partial^{(i)}T) = \ln\left(\frac{R_i^+ + s_i^+}{R_i^- + s_i^-}\right), \tag{2.132}$$

$$I^S_{-3}(T) = \begin{cases} 0, & \text{if } w_0 = 0 \\ \frac{1}{|h|}\sum_{i=1}^3 \beta_i, & \text{otherwise} \end{cases} \tag{2.133}$$

with

$$\beta_i = \arctan\left(\frac{t_i s_i^+}{(R_i^0)^2 + |w_0|R_i^+}\right) - \arctan\left(\frac{t_i s_i^-}{(R_i^0)^2 + |w_0|R_i^-}\right), \text{ if } w_0 \neq 0, \tag{2.134}$$

and the quantities

$$\begin{aligned}
R_1^+ &= R_2^- = |\mathbf{r} - \mathbf{r}'_3| & s_1^- &= -\frac{(l_3 - u_0)(l_3 - u_3) + v_0 v_3}{l_1} \\
R_2^+ &= R_3^- = |\mathbf{r} - \mathbf{r}'_1| & s_2^- &= -\frac{u_3(u_3 - u_0) + v_3(v_3 - v_0)}{l_2} \\
R_3^+ &= R_1^- = |\mathbf{r} - \mathbf{r}'_2| & s_3^- &= -u_0 \\
R_i^0 &= \sqrt{t_i^2 + h^2} & t_1 &= \frac{v_0(u_3 - l_3) + v_3(l_3 - u_0)}{l_1} \\
s_1^+ &= s_1^- + l_1 & t_2 &= \frac{u_0 v_3 - v_0 u_3}{l_2} \\
s_2^+ &= s_2^- + l_2 & t_3 &= v_0. \\
s_3^+ &= s_3^- + l_3 &&
\end{aligned} \tag{2.135}$$

Recursive rules for higher powers of Eqs. (2.130) and (2.131) are derived in Ref. [13] and Ref. [12] with the result

$$I^l_q(\partial^{(i)}T) = \frac{1}{q+1}\left[q(R_i^0)^2 I^l_{q-2}(\partial^{(i)}T) + s_i^+(R_i^+)^q - s_i^-(R_i^-)^q\right], \tag{2.136}$$

$$I^S_q(T) = \frac{1}{q+2}\left(q w_0^2 I^S_{q-2}(T) - \sum_{i=1}^m t_i I^l_q(\partial^{(i)}T)\right). \tag{2.137}$$

Since the cross product

$$(\mathbf{r} - \mathbf{r}') \times (\mathbf{r}' - \mathbf{p}_n^\pm) = -(\mathbf{r} - \mathbf{p}_n^\pm) \times (\mathbf{r} - \mathbf{r}') \tag{2.138}$$

and since the gradient

$$\nabla' R^q = -q(\mathbf{r} - \mathbf{r}')R^{q-1}, \tag{2.139}$$

we may rewrite Eq. (2.116) as

$$\mathbf{X}_4^q(T_n^\pm) = -(\mathbf{r} - \mathbf{p}_n^\pm) \times \mathbf{X}_3^q(T_n^\pm), \tag{2.140}$$

where

$$\mathbf{X}_3^q(T_n^\pm) := \int_{T_n^\pm} \nabla' R^q \mathrm{d}S'. \tag{2.141}$$

In the end, by reducing Eqs. (2.114), (2.115), and (2.141) to a series of closed-form integrals of the forms of Eqs. (2.130) and (2.131), we get the iterative formulas

$$X_1^q(T_n^\pm) = I_q^S(T_n^\pm) \tag{2.142}$$

$$\mathbf{X}_2^q(T_n^\pm) = \frac{1}{q+2}\sum_{i=1}^{3} \hat{\mathbf{m}}_i I_{q+2}^l(\partial^{(i)} T_n^\pm) + (\boldsymbol{\rho} - \mathbf{p}_n^\pm) I_q^S(T_n^\pm) \tag{2.143}$$

$$\mathbf{X}_3^q(T_n^\pm) = \sum_{i=1}^{3} \hat{\mathbf{m}}_i I_q^l(\partial^{(i)} T_n^\pm) - w_0 q \hat{\mathbf{n}} I_{q-2}^S(T_n^\pm), \tag{2.144}$$

where $\hat{\mathbf{m}}_i$ is the outer unit normal of the triangle edge $\partial^{(i)} T_n^\pm$.

Outer integrals of the subtracted terms

Consider the double integrals over the subtracted terms:

$$\int_{T_m} \mathrm{d}S(\mathbf{r} - \mathbf{p}_m) \cdot \int_{T_n} \mathrm{d}S' \left(\frac{1}{R} - \frac{k_i^2}{2} R\right)(\mathbf{r}' - \mathbf{p}_n), \tag{2.145}$$

$$\int_{T_m} \mathrm{d}S(\mathbf{r} - \mathbf{p}_m) \cdot \int_{T_n} \mathrm{d}S' \left(\nabla' \frac{1}{R} - \frac{k_i^2}{2} \nabla' R\right) \times (\mathbf{r}' - \mathbf{p}_n). \tag{2.146}$$

As discussed in Section 2.6.3, the inner integrals over T_n of both terms in both equations may be exactly evaluated using the iterative formulas (2.143) and (2.140) with $q = -1$ and $q = 1$. The outer integral of both terms in Eq. (2.145), and the second term in Eq. (2.146), are regular and may be evaluated using Gaussian quadratures. However, $\nabla' R^{-1}$ is a $1/R^3$ singularity, which makes the outer integral of the first term in Eq. (2.146) singular if T_m and T_n have a common point. We will now present the method used in Ref. [12] to modify this outer integral with a singular integrand, to an outer integral of a regular function.

Observing that

$$(\mathbf{r}' - \mathbf{p}_n) = (\mathbf{p}_m - \mathbf{p}_n) + (\mathbf{r}' - \mathbf{r}) + (\mathbf{r} - \mathbf{p}_m), \tag{2.147}$$

and that (for integer q)

$$\nabla' R^q \times (\mathbf{r}' - \mathbf{r}) = 0, \tag{2.148}$$

we may write the first term of Eq. (2.146) as

$$\int_{T_m} \mathrm{d}S(\mathbf{r} - \mathbf{p}_m) \cdot \int_{T_n} \mathrm{d}S' \nabla' \frac{1}{R} \times \left[(\mathbf{r} - \mathbf{p}_n) + (\mathbf{p}_m - \mathbf{p}_n)\right]. \tag{2.149}$$

Both $(\mathbf{r} - \mathbf{p}_n)$ and $(\mathbf{p}_m - \mathbf{p}_n)$ are independent of $\mathbf{r}'$ and may be moved outside the inner integral, and since

$$(\mathbf{r} - \mathbf{p}_m) \cdot \left[(\mathbf{r} - \mathbf{p}_m) \times \int_{T_n} \mathrm{d}S' \nabla' R^q\right] = 0, \tag{2.150}$$

we get

$$-\int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\left[(\mathbf{p}_m-\mathbf{p}_n)\times\int_{T_n}\mathrm{d}S'\nabla'\frac{1}{R}\right]. \tag{2.151}$$

Next, by splitting the gradient ∇' into surface and normal components, $\nabla' = \nabla'_s + \nabla'_n$, Eq. (2.151) becomes

$$-\int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\left[(\mathbf{p}_m-\mathbf{p}_n)\times\int_{T_n}\mathrm{d}S'\left(\nabla'_n\frac{1}{R}+\nabla'_s\frac{1}{R}\right)\right]. \tag{2.152}$$

Consider first the normal gradient

$$\nabla'_n\frac{1}{R} = \hat{\mathbf{n}}\left(\hat{\mathbf{n}}\cdot\nabla'\frac{1}{R}\right) = \hat{\mathbf{n}}\left[\hat{\mathbf{n}}\cdot\frac{1}{R^3}(\mathbf{r}-\mathbf{r}')\right]. \tag{2.153}$$

Noting that in the transformed coordinate system (u, v, w), $\hat{\mathbf{n}} = (0, 0, 1)$, we insert Eqs. (2.122) and (2.123) giving

$$\nabla'_n\frac{1}{R} = \hat{\mathbf{n}}\frac{w_0}{R^3}. \tag{2.154}$$

Thus, the first term in Eq. (2.152) becomes

$$\begin{aligned}-\int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\left((\mathbf{p}_m-\mathbf{p}_n)\times\int_{T_n}\mathrm{d}S'\hat{\mathbf{n}}\frac{w_0}{R^3}\right)\\ = -\int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\Big[(\mathbf{p}_m-\mathbf{p}_n)\times\hat{\mathbf{n}}w_0X_1^{-3}(T_n)\Big],\end{aligned} \tag{2.155}$$

where $X_1^{-3}(T_n)$ is defined in Eq. (2.142) with $q = -3$.

Using Gauss theorem in the plane, the we may rewrite the inner integral of the second term of Eq. (2.152) to a line integral

$$\begin{aligned}\int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\left((\mathbf{p}_m-\mathbf{p}_n)\times\int_{T_n}\mathrm{d}S'\nabla'_s\frac{1}{R}\right)\\ = \int_{T_m} \mathrm{d}S(\mathbf{r}-\mathbf{p}_m)\cdot\left[(\mathbf{p}_m-\mathbf{p}_n)\times\int_{\partial T_n}\mathrm{d}l'\hat{\mathbf{m}}\frac{1}{R}\right],\end{aligned} \tag{2.156}$$

where ∂T_n denotes all three edges of the triangle T_n, and $\hat{\mathbf{m}}$ is their outer unit normal. By changing the order of integration, the expression in Eq. (2.156) becomes

$$\begin{aligned}\int_{\partial T_n}\mathrm{d}l'\left[(\mathbf{p}_m-\mathbf{p}_n)\times\hat{\mathbf{m}}(\mathbf{r}')\right]\cdot\int_{T_m}\mathrm{d}S\frac{(\mathbf{r}-\mathbf{p}_m)}{R}\\ = \int_{\partial T_n}\mathrm{d}l'\Big\{\left[(\mathbf{p}_m-\mathbf{p}_n)\times\hat{\mathbf{m}}(\mathbf{r}')\right]\cdot\mathbf{X}_2^{-1}(T_m)\Big\},\end{aligned} \tag{2.157}$$

where $\mathbf{X}_2^{-1}(T_m)$ is defined in Eq. (2.143) with $q = -1$.

As a result, the inner integral of both Eq. (2.155) and Eq. (2.157) may be evaluated analytically, and the outer integral of Eq. (2.157) is regular, and may be evaluated using an appropriate numerical method, such as Gauss-Legendre quadrature (see Section 2.6.5). The outer integral of Eq. (2.155) is still singular, but at

common points in T_m and T_n the surface gradient term will be dominant (unless T_m and T_n lies in the same plane in which both terms will be zero) and Gaussian quadrature will still yield reasonably accurate results [12].

Although it is tempting to apply singularity subtraction on every integral to reduce implementation complexity, it is important to note, that singularity subtraction should only be applied in the singular or near singular case. Otherwise, $k_i R$ may be too large when S_n and S_m are far away from each other, so that the higher order terms in Eq. (2.103) are no longer negligible, resulting in reduced numerical accuracy.

2.6.4 Field Distribution

After solving the matrix equation (2.99) and finding the expansion coefficients α_n and β_n, the equivalent surface currents $\mathbf{J}(\mathbf{r}')$ (Eq. (2.84)) and $\mathbf{M}(\mathbf{r}')$ (Eq. (2.85)), may be used to yield direct expressions for evaluating the electric and magnetic field at any point inside either region [1]:

$$\begin{aligned}\mathbf{E}_i(\mathbf{r}) = \begin{Bmatrix}+\\-\end{Bmatrix} \sum_{n=1}^{N} \Bigg[& \\ - \alpha_n \frac{\omega\mu_i}{i} \left(\int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}') + \frac{1}{k_i^2} \int_{S_n} \mathrm{d}S' \nabla G_i(\mathbf{r},\mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}') \right) & \quad (2.158a) \\ + \beta_n \int_{S_n} \mathrm{d}S' [\nabla' G_i(\mathbf{r},\mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}') \Bigg] + \begin{cases} \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}), & i = 1 \text{ and } \mathbf{r} \in V_1, \\ 0, & i = 2 \text{ and } \mathbf{r} \in V_2, \end{cases} & \end{aligned}$$

$$\begin{aligned}\mathbf{H}_i(\mathbf{r}) = \begin{Bmatrix}+\\-\end{Bmatrix} \sum_{n=1}^{N} \Bigg[& \\ - \beta_n \frac{\omega\mu_i}{i} \left(\int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}') + \frac{1}{k_i^2} \int_{S_n} \mathrm{d}S' \nabla G_i(\mathbf{r},\mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}') \right) & \quad (2.158b) \\ - \alpha_n \int_{S_n} \mathrm{d}S' [\nabla' G_i(\mathbf{r},\mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}') \Bigg] + \begin{cases} \mathbf{H}_1^{\mathrm{inc}}(\mathbf{r}), & i = 1 \text{ and } \mathbf{r} \in V_1, \\ 0, & i = 2 \text{ and } \mathbf{r} \in V_2. \end{cases} & \end{aligned}$$

Here, to replace the dyadic Green's function with the scalar, we have used the transformation

$$\begin{aligned}&\int_{S_n} \mathrm{d}S' \overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}') \cdot \mathbf{f}_n(\mathbf{r}') \\ &\qquad = \frac{1}{k_i^2} \int_{S_n} \mathrm{d}S' \nabla G_i(\mathbf{r},\mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}') + \int_{S_n} \mathrm{d}S' G_i(\mathbf{r},\mathbf{r}') \mathbf{f}_n(\mathbf{r}'),\end{aligned} \quad (2.159)$$

which is similar to Eqs. (2.88) and (2.89), and the transformation of the curl of $\overline{\mathbf{G}}_i(\mathbf{r},\mathbf{r}')$, Eq. (2.90).

The scattered field is obtained by omitting the final term of Eq. (2.158), i.e.

$$\mathbf{E}^{\mathrm{sca}}(\mathbf{r}) = \mathbf{E}_1(\mathbf{r}) - \mathbf{E}_1^{\mathrm{inc}}(\mathbf{r}), \quad \mathbf{r} \in V_1. \quad (2.160)$$

When the observation point is away from the scattering surface, Gaussian quadrature may be applied to evaluate the integrals above. However, the Green's function and its gradient still have a singularity. Thus, when calculating field distributions at observation points close to the surface of the scattering surface, the singularity subtraction techniques in Eq. (2.113) are appropriate. As follows, the calculation of Eq. (2.158) involves both Gaussian quadrature, and evaluation of analytical integrals using iterative formulas.

2.6.5 Line integral over a triangle

The singularity subtraction technique presented in Section 2.6.3 requires the evaluation of a line integral over the boundary of a triangle (see Eq. (2.157)). The 1D Gauss-Legendre quadrature formula discussed in Section 2.4.1 is a suitable numerical method for such an integral. Be that as it may, the line integral in Eq. (2.157) is over a line segment in 3D space, and needs to be transformed onto an axis with the 1D limits $[-1, 1]$. To do this, we first note that a line integral over ∂T_n equals the sum of the line integrals of all the edges $\partial^{(i)} T_n$ of the triangle T_n:

$$\int_{\partial T_n} \mathrm{d}l' g(\mathbf{r}') = \int_{\partial^{(1)} T_n} \mathrm{d}l' g(\mathbf{r}') + \int_{\partial^{(2)} T_n} \mathrm{d}l' g(\mathbf{r}') + \int_{\partial^{(3)} T_n} \mathrm{d}l' g(\mathbf{r}'). \tag{2.161}$$

Here $g(\mathbf{r}')$ is a function dependent on the source point $\mathbf{r}'$, similarly to the integrand in Eq. (2.157) where the $\mathbf{r}'$-dependency lies in R and hence in $\mathbf{X}_2^{-1}(T_m)$ (2.115).

Let $\mathbf{r}'_1$, $\mathbf{r}'_2$ and $\mathbf{r}'_3$ be the vertices of T_n. Then the edge $\partial^{(1)} T_n$ is a line segment from $\mathbf{r}'_2$ to $\mathbf{r}'_3$. If $\mathbf{r}'$ is bound to that line segment, it may be parameterised as

$$\mathbf{r}'(t) = \frac{\mathbf{r}'_3 - \mathbf{r}'_2}{2} t + \frac{\mathbf{r}'_3 + \mathbf{r}'_2}{2}, \quad -1 \leq t \leq 1, \tag{2.162}$$

and we have the integral transformation

$$\int_{\partial^{(1)} T} \mathrm{d}l' g(\mathbf{r}') = \frac{1}{2} \left| \mathbf{r}'_3 - \mathbf{r}'_2 \right| \int_{-1}^{1} \mathrm{d}t g(\mathbf{r}'(t)). \tag{2.163}$$

The right hand side of this equation is on the form Eq. (2.68), and Gauss-Legendre quadrature formulas may now be applied.

2.6.6 Reducing the Amount of Integral Evaluations

One of the major computational efforts in simulating electromagnetic scattering using the methods presented in this chapter is to calculate the matrix elements of $\boldsymbol{\mathcal{H}}$ (see Eq. (2.100)). The number of matrix elements to be evaluated increases proportional to the number of RWG basis functions squared, N^2. For a closed surface, which is the case for the combined EFIE and MFIE formulation leading to the matrix equation (2.99) with the elements $\mathcal{D}_{mn}^{(i)}$ and $\mathcal{K}_{mn}^{(i)}$, the number of bases is equal to the number of edges, $N = N_e$. However, integrating over a basis S_m, identified by its basis edge with length L_m, involves integrating over both of its

adjacent faces (triangles) $T_m^{\pm}$. Since a face has three edges, it appears in three different RWG basis functions. Thus, evaluating the integrals in Eq. (2.95) with a face-by-face approach, instead of basis-by-basis approach, will reduce the amount of integral evaluations by a factor proportional to

$$\frac{N_e^2}{N_f^2} = \frac{N_e^2}{(\frac{2}{3}N_e)^2} = \frac{9}{4}, \tag{2.164}$$

where N_f is the number of faces in the surface triangulation.

This section considers the formulation of the double integrals in Eq. (2.95) using a face-by-face approach and double Gaussian quadrature.

Basis-by-basis approach when calculating $\boldsymbol{\mathcal{D}}^{(i)}$

Consider the second term of the matrix element $\mathcal{D}_{mn}^{(i)}$, defined in Eq. (2.95a). Writing out the divergence of $\mathbf{f}_m(\mathbf{r})$ and $\mathbf{f}_n(\mathbf{r}')$ yields

$$\begin{aligned} \nabla \cdot \mathbf{f}_m(\mathbf{r}) &= \frac{L_m}{2A_m^{\pm}} \nabla_s \cdot (\mathbf{r} - \mathbf{p}_m^{\pm}) = \frac{L_m}{A_m^{\pm}}, \quad \mathbf{r} \in T_m^{\pm} \\ \nabla' \cdot \mathbf{f}_n(\mathbf{r}') &= \frac{L_n}{A_n^{\pm}}, \quad \mathbf{r}' \in T_n^{\pm}. \end{aligned} \tag{2.165}$$

The integrals over S_m and S_n may be split into two integrals each, integrating over both faces $T_m^{\pm}$ and $T_n^{\pm}$ separately

$$\begin{aligned} &-\frac{1}{k_i^2} \int_{S_m} [\nabla \cdot \mathbf{f}_m(\mathbf{r})] \int_{S_n} G_i(\mathbf{r}, \mathbf{r}') \nabla' \cdot \mathbf{f}_n(\mathbf{r}') \\ &\quad = -\frac{9L_mL_n}{4k_i^2} \Bigg(\frac{1}{A_m^+ A_n^+} \int_{T_m^+} \int_{T_n^+} G_i(\mathbf{r}, \mathbf{r}') \mathrm{d}S' \mathrm{d}S \\ &\qquad - \frac{1}{A_m^+ A_n^-} \int_{T_m^+} \mathrm{d}S \int_{T_n^-} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \\ &\qquad - \frac{1}{A_m^- A_n^+} \int_{T_m^-} \mathrm{d}S \int_{T_n^+} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \\ &\qquad + \frac{1}{A_m^- A_n^-} \int_{T_m^-} \mathrm{d}S \int_{T_n^-} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \Bigg). \end{aligned} \tag{2.166}$$

To rewrite the integrals on Gaussian quadrature form, such as Eq. (2.69), we use the following transformations onto barycentric coordinates

$$\mathbf{r} = \alpha \mathbf{r}_1 + \beta \mathbf{r}_2 + \gamma \mathbf{r}_3 \tag{2.167}$$

$$\mathbf{r}' = \xi \mathbf{r}_1' + \eta \mathbf{r}_2' + \zeta \mathbf{r}_3', \tag{2.168}$$

where (α, β, γ) and (ξ, η, ζ) are the barycentric coordinates of the triangles $T_m^{\pm}$ and $T_n^{\pm}$, respectively. The double integral in Eq. (2.166) may then be approximated as

$$\begin{aligned}&\int_{S_m} \mathrm{d}S \int_{S_n} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \\ &\quad = \sum_{p=1}^{2}\sum_{q=1}^{2}(-1)^{p+q}\sum_{j=1}^{N_j} w_j \sum_{k=1}^{N_k} w_k G_i(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p, \xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q),\end{aligned} \tag{2.169}$$

where $\mathbf{r}_i^1$, $\mathbf{r}_i^2$, $\mathbf{r}_i'^1$, and $\mathbf{r}_i'^2$ are the vertices of the faces T_m^+, T_m^-, T_n^+, and T_n^-, respectively. The factor $(-1)^{p+q}$ makes sure the outer terms have the correct sign. There are N_j quadrature points, $(\alpha_j, \beta_j, \gamma_j)$, on the faces $T_m^{\pm}$, and N_k quadrature points, (ξ_k, η_k, ζ_k), on the faces $T_n^{\pm}$, with w_j and w_k being the corresponding weights.

With similar treatment of the first term of $\mathcal{D}_{mn}^{(i)}$, Gaussian quadrature in barycentric coordinates yields:

$$\begin{aligned}&\int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \mathbf{f}_n(\mathbf{r}') \\ &\quad = \frac{L_m L_n}{4}\Bigg[\sum_{p=1}^{2}\sum_{q=1}^{2}(-1)^{p+q}\sum_{j=1}^{N_j} w_j(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \mathbf{p}_m^p) \\ &\qquad \cdot \sum_{k=1}^{N_k} w_k G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}')(\xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q - \mathbf{p}_n^q)\Bigg],\end{aligned} \tag{2.170}$$

where

$$G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') = G_i(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p, \xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q), \tag{2.171}$$

and we have used the notations $\mathbf{p}_m^1$, $\mathbf{p}_m^2$, $\mathbf{p}_n^1$, and $\mathbf{p}_m^2$ for the free vertices $\mathbf{p}_m^+$, $\mathbf{p}_m^-$, $\mathbf{p}_n^+$, and $\mathbf{p}_n^-$, respectively. Thus, calculation of the matrix element $\mathcal{D}_{mn}^{(i)}$ using a basis-by-basis approach is achieved by simply implementing the quadruple sums (2.169) and (2.170).

Face-by-face approach for calculating $\boldsymbol{\mathcal{D}}^{(i)}$

To evaluate the integrals of the matrix elements $\mathcal{D}_{mn}^{(i)}$ using a face-by-face approach, they must first be independent of quantities associated with a particular basis function. Because of the free vertices $\mathbf{p}_m^p$ and $\mathbf{p}_n^q$, the Gaussian quadrature sums in

Eq. (2.170) are associated with the bases S_m and S_n. Nevertheless, by defining

$$I_{\xi}^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k \xi_k G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') \tag{2.172a}$$

$$I_{\eta}^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k \eta_k G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') \tag{2.172b}$$

$$I^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') \tag{2.172c}$$

$$I_{\zeta}^{\mathbf{r}q} = I^{\mathbf{r}q} - I_{\xi}^{\mathbf{r}q} - I_{\zeta}^{\mathbf{r}q}, \tag{2.172d}$$

where the last equation emerges from the constraint

$$\zeta = 1 - \xi - \eta,$$

the inner sum of Eq. (2.170) may be written as

$$\begin{aligned} &\sum_{k=1}^{N_k} w_k G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}')(\xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q - \mathbf{p}^q) \\ &\quad = I_{\xi}^{\mathbf{r}q} \mathbf{r}_1'^q + I_{\eta}^{\mathbf{r}q} \mathbf{r}_3'^q + I_{\zeta}^{\mathbf{r}q} \mathbf{r}_3'^q - I^{\mathbf{r}q} \mathbf{p}^q. \end{aligned} \tag{2.173}$$

The Gaussian quadrature sums (2.172) are in fact not associated with any particular basis. They are only dependent on the observation point $\mathbf{r}$ and the face represented by q. To get the outer integral independent of basis as well, we use the Eq. (2.173) and expand the dot product of the second innermost sum of Eq. (2.170) to obtain

$$\begin{aligned} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' G_i(\mathbf{r}, \mathbf{r}') \mathbf{f}_n(\mathbf{r}') = \frac{L_m L_n}{4} \Bigg[\sum_{p=1}^{2} \sum_{q=1}^{2} (-1)^{p+q} \Big(\\ Q_{\alpha\xi}^{pq} \mathbf{r}_1^p \cdot \mathbf{r}_1'^q + Q_{\alpha\eta}^{pq} \mathbf{r}_1^p \cdot \mathbf{r}_2'^q + Q_{\alpha\zeta}^{pq} \mathbf{r}_1^p \cdot \mathbf{r}_3'^q - Q_{\alpha}^{pq} \mathbf{r}_1^p \cdot \mathbf{p}_n^q \\ + Q_{\beta\xi}^{pq} \mathbf{r}_2^p \cdot \mathbf{r}_1'^q + Q_{\beta\eta}^{pq} \mathbf{r}_2^p \cdot \mathbf{r}_2'^q + Q_{\beta\zeta}^{pq} \mathbf{r}_2^p \cdot \mathbf{r}_3'^q - Q_{\beta}^{pq} \mathbf{r}_2^p \cdot \mathbf{p}_n^q \\ + Q_{\gamma\xi}^{pq} \mathbf{r}_3^p \cdot \mathbf{r}_1'^q + Q_{\gamma\eta}^{pq} \mathbf{r}_3^p \cdot \mathbf{r}_2'^q + Q_{\gamma\zeta}^{pq} \mathbf{r}_3^p \cdot \mathbf{r}_3'^q - Q_{\gamma}^{pq} \mathbf{r}_3^p \cdot \mathbf{p}_n^q \\ - Q_{\xi}^{pq} \mathbf{p}_m^p \cdot \mathbf{r}_1'^q - Q_{\eta}^{pq} \mathbf{p}_m^p \cdot \mathbf{r}_2'^q - Q_{\zeta}^{pq} \mathbf{p}_m^p \cdot \mathbf{r}_3'^q + Q^{pq} \mathbf{p}_m^p \cdot \mathbf{p}_n^q \Big) \Bigg]. \end{aligned} \tag{2.174}$$

In obtaining this result, we have defined the following quantities,

$$
\begin{aligned}
Q^{pq}_{\alpha\xi} &= \sum_{k=1}^{N_k} w_k \alpha_k I^{\mathbf{r}q}_{\xi}, & Q^{pq}_{\eta} &= \sum_{k=1}^{N_k} w_k I^{\mathbf{r}q}_{\eta}, \\
Q^{pq}_{\beta\xi} &= \sum_{k=1}^{N_k} w_k \beta_k I^{\mathbf{r}q}_{\xi}, & Q^{pq}_{\alpha} &= \sum_{k=1}^{N_k} w_k \alpha_k I^{\mathbf{r}q}, \\
Q^{pq}_{\xi} &= \sum_{k=1}^{N_k} w_k I^{\mathbf{r}q}_{\xi}, & Q^{pq}_{\beta} &= \sum_{k=1}^{N_k} w_k \beta_k I^{\mathbf{r}q}, \\
Q^{pq}_{\alpha\eta} &= \sum_{k=1}^{N_k} w_k \alpha_k I^{\mathbf{r}q}_{\eta}, & Q^{pq} &= \sum_{k=1}^{N_k} w_k I^{\mathbf{r}q}, \\
Q^{pq}_{\beta\eta} &= \sum_{k=1}^{N_k} w_k \beta_k I^{\mathbf{r}q}_{\eta}, &
\end{aligned}
\tag{2.175}
$$

and

$$
\begin{aligned}
Q^{pq}_{\alpha\zeta} &= Q^{pq}_{\alpha} - Q^{pq}_{\alpha\xi} - Q^{pq}_{\alpha\eta}, & Q^{pq}_{\gamma} &= Q^{pq} - Q^{pq}_{\alpha} - Q^{pq}_{\beta}, \\
Q^{pq}_{\beta\zeta} &= Q^{pq}_{\beta} - Q^{pq}_{\beta\xi} - Q^{pq}_{\beta\eta}, & Q^{pq}_{\gamma\xi} &= Q^{pq}_{\xi} - Q^{pq}_{\alpha\xi} - Q^{pq}_{\beta\xi}, \\
Q^{pq}_{\zeta} &= Q^{pq} - Q^{pq}_{\xi} - Q^{pq}_{\eta}, & Q^{pq}_{\gamma\eta} &= Q^{pq}_{\eta} - Q^{pq}_{\alpha} - Q^{pq}_{\beta\xi}, \\
& & Q^{pq}_{\gamma\zeta} &= Q^{pq}_{\zeta} - Q^{pq}_{\alpha\zeta} - Q^{pq}_{\beta\zeta}.
\end{aligned}
\tag{2.176}
$$

as consequence of the constraints $\gamma = 1 - \alpha - \beta$ and $\zeta = 1 - \xi - \eta$.

The sums (2.175) are only dependent on the vertices of the face-pair identified by the superscript p and q, and on the particular Gaussian quadrature formula applied. If we let p and q take any integer value from 1 to N_f, such that Eq. (2.175) may be evaluated for all combinations of face pairs p and q, each evaluation contributes to nine elements in $\boldsymbol{\mathcal{D}}^{(i)}$. This makes sense because instead of evaluating 4 double integrals per basis pair, we evaluate a single double integral per face pair, i.e. reducing the number of Gaussian quadrature sums by a factor of

$$
\frac{4N_e^2}{N_f^2} = \frac{4N_e^2}{(\frac{2}{3}N_e)^2} = 9, \tag{2.177}
$$

If the same quadrature formulas are used to evaluate the second term in $\mathcal{D}^{(i)}_{mn}$, then no additional quadrature calculation is necessary for this term since the inner double sum of Eq. (2.169) is in fact Q^{pq} in Eq. (2.175).

Calculation of $\boldsymbol{\mathcal{K}}^{(i)}$

Similarly to the terms in $\mathcal{D}_{mn}^{(i)}$, the double integral in $\mathcal{K}_{mn}^{(i)}$ may be written out as Gaussian quadrature sums

$$\begin{aligned}
&\int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \int_{S_n} \mathrm{d}S' [\nabla' G_i(\mathbf{r}, \mathbf{r}')] \times \mathbf{f}_n(\mathbf{r}') \\
&\quad = \frac{L_m L_n}{4} \Bigg[\sum_{p=1}^{2} \sum_{q=1}^{2} (-1)^{p+q} \sum_{j=1}^{N_j} w_j (\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \mathbf{p}_m^p) \\
&\quad \cdot \sum_{k=1}^{N_k} w_k \left[G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') \left(\frac{1}{R^{\mathrm{Bary}}} - ik_i \right) \right] \hat{\mathbf{R}}^{\mathrm{Bary}} \times (\xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q - \mathbf{p}_n^q) \Bigg],
\end{aligned} \tag{2.178}$$

where

$$R^{\mathrm{Bary}} = \left| \alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \xi_k \mathbf{r}_1'^q - \eta_k \mathbf{r}_2'^q - \zeta_k \mathbf{r}_3'^q \right| \tag{2.179}$$

$$\hat{\mathbf{R}}^{\mathrm{Bary}} = \frac{1}{R^{\mathrm{Bary}}} \left(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \xi_k \mathbf{r}_1'^q - \eta_k \mathbf{r}_2'^q - \zeta_k \mathbf{r}_3'^q \right). \tag{2.180}$$

Straightforward implementation of Eq. (2.178) constitute a basis-by-basis approach. By expanding the cross product, the vector $\mathbf{p}_n^q$ may be factored out of the innermost sum such that the two inner sums in Eq. (2.178) become

$$\begin{aligned}
&\sum_{j=1}^{N_j} w_j (\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \mathbf{p}_m^p) \cdot \sum_{k=1}^{N_k} w_k \left[G_i^{\mathrm{Bary}}(\mathbf{r}, \mathbf{r}') \left(\frac{1}{R^{\mathrm{Bary}}} - ik_i \right) \right] \hat{\mathbf{R}}^{\mathrm{Bary}} \\
&\qquad \times (\xi_k \mathbf{r}_1'^q + \eta_k \mathbf{r}_2'^q + \zeta_k \mathbf{r}_3'^q - \mathbf{p}_n^q) = \\
&\sum_{j=1}^{N_j} w_j (\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \mathbf{p}_m^p) \cdot \Big\{ \mathbb{I}_\xi^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_1'^q + \mathbf{r}_1'^q \times \mathbf{p}_n^q \right] \\
&\qquad + \mathbb{I}_\eta^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_2'^q + \mathbf{r}_2'^q \times \mathbf{p}_n^q \right] \\
&\qquad + \mathbb{I}_\zeta^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_3'^q + \mathbf{r}_3'^q \times \mathbf{p}_n^q \right] \\
&\qquad - \mathbb{I}^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{p}_n^q \right] \Big\},
\end{aligned} \tag{2.181}$$

where

$$\mathbb{I}_{\xi}^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k \xi_k G_i^{\text{Bary}}(\mathbf{r}, \mathbf{r}') \left(\frac{1}{R^{\text{Bary}}} - ik_i \right) \frac{1}{R^{\text{Bary}}} \tag{2.182a}$$

$$\mathbb{I}_{\eta}^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k \eta_k G_i^{\text{Bary}}(\mathbf{r}, \mathbf{r}') \left(\frac{1}{R^{\text{Bary}}} - ik_i \right) \frac{1}{R^{\text{Bary}}} \tag{2.182b}$$

$$\mathbb{I}^{\mathbf{r}q} = \sum_{k=1}^{N_k} w_k G_i^{\text{Bary}}(\mathbf{r}, \mathbf{r}') \left(\frac{1}{R^{\text{Bary}}} - ik_i \right) \frac{1}{R^{\text{Bary}}} \tag{2.182c}$$

$$\mathbb{I}_{\zeta}^{\mathbf{r}q} = \mathbb{I}^{\mathbf{r}q} - \mathbb{I}_{\xi}^{\mathbf{r}q} - \mathbb{I}_{\zeta}^{\mathbf{r}q}. \tag{2.182d}$$

The free vertex $\mathbf{p}_m^p$ may be factored out of the remaining sum by expanding the dot product in Eq. (2.174). The result is presented in Appendix A. In this expansion, there are 18 independent, scalar Gaussian quadrature sums (of similar form as Eq. (2.175)), and a face-by-face approach is achieved by calculating these for every face-pair, and let them contribute in the elements of $\boldsymbol{\mathcal{K}}^{(i)}$. As with $\boldsymbol{\mathcal{D}}^{(i)}$, a face-by-face approach will reduce the number of integral evaluations by a factor of 9.

However, the cost you pay to increase the computational efficiency by reducing the number of integrals is obviously the need to store more variables. In fact, you need to store at least 5 of the 9 independent sums in Eq. (2.175) (the ones that are multiplied with a dot product involving $\mathbf{p}_m^p$ or $\mathbf{p}_n^q$), and 12 of the 18 independent sums in Eq. (A.4). This means storing a minimum of $21N_f^2$ numbers of complex type. The extra procedural implementations as opposed to a more direct implementation is also an disadvantage, as it introduces the possibility for more bugs and errors.

Chapter 3

Numerical Implementation

The methods presented in Section 2.6 was implemented in a set of modules, becoming the building blocks of a program aiming to simulate the electromagnetic scattering problem introduced in Section 2.1. The modules uses an object-oriented design, written in the Fortran programming language. The first section of the chapter discusses all modules, their interfaces, functionalities and implementation approaches.

Fortran was chosen because of its speed, its excellent handling of scientific computing, and the vast availability of scientific libraries and code. The calculation of the matrix elements in Eq. (2.99) requires evaluation of a large number of integrals, meaning plenty of number crunching, of which Fortran is particularly suitable. The reliable scientific libraries may then be used to solve the resulting dense matrix equation. Modern Fortran also support object-oriented programming.

A large number of routines were designed to test the procedures of the modules, making sure they performed as expected. Section 3.2 introduces the testing programs used to validate the modules and the simulation results.

Finally, the chapter contains a discussion on the compilation of the software, and the Makefiles used.

3.1 Simulation Design

The goal of the program was to numerically simulate, based on the electric and magnetic field integral equations (EFIE/MFIE), the scattering of electromagnetic waves by an arbitrary shaped object using the method of moments (MoM) with Galerkin's method of weighted residuals and the RWG basis functions. This was accomplished through execution of the following steps,

1. Discretising of the scattering surface using a surface triangulation technique.
2. Storing the triangulation in a suitable data structure.
3. Mapping the triangulation onto the RWG basis functions.
4. Calculating the matrix elements of Eq. (2.100) by numerical integration using double Gaussian quadrature, or by a combination of numerical integration and

analytical evaluation if singularity subtraction of the integrands are appropriate.

5. Finding the expansion coefficients Eq. (2.93) by solving the resulting matrix equation.

6. Using the solution to estimate the total and scattered electric and magnetic field distribution.

A major goal in all of the above steps was to keep generality and versatility, while ensuring computational efficiency and balancing computation speed and memory cost. Before going into details on each module and how they process the steps above, we give a short introduction to all modules.

The discretisation is accomplished through the use of external software, which outputs a file representing the triangulation. This file is read by functionality implemented in the I/O module `io_mod`. The main data type in module `mesh_mod`, called `mesh_mod_type`, uses `io_mod` to store the discretised surface in a convenient hierarchical data structure based on faces (also called elements), edges, vertices and nodes. The module `RWG_basis_mod` contains the main data type which inherits `mesh_mod_type` and maps its data structure onto a new structure, suitable for evaluating the Rao-Wilton-Glisson (RWG) basis functions. The type `mesh_mod_type` is not restricted to storing the triangulation required to construct RWG basis functions. It is designed to be able to store elements having an arbitrary number vertices, and elements having quadratic or cubic edges. In this way, `mesh_mod_type` is useful for other basis functions as well, and especially suitable when a hierarchical data structure is desired. Thus, the discretised surface is often be referred to as the *mesh*, as it may or may not be a triangulation.

The main type of `PMCHW_RWG_mod`, called `PMCHW_RWG_mod_type`, inherits `RWG_basis_mod_type`, and uses its data structure to calculate the matrix elements in Eq. (2.100) and the vector (2.101). These values are successively stored such that the matrix equation (2.99) may be solved by internal procedures. The solution(s) are then stored, also in `PMCHW_RWG_mod_type`, which encapsulate member procedures for calculating the resulting electric and magnetic field distributions at any given observation point. In addition, `PMCHW_RWG_mod_type` contains functionality to calculate bistatic scattering cross sections.

Finally, results from `PMCHW_RWG_mod` may be written to file using a routine in `io_mod`. In this project, scripts written in Python was used to produce plots from the results calculated by `PMCHW_RWG_mod`.

The modules mentioned above are dependent on several additional, essential modules called `working_precision_mod`, `math_funcs_mod` and `constants_mod`. The module `working_precision_mod`, being imported by all other modules, defines the working precision of the simulation. The module `math_funcs_mod` contains frequently used mathematical functions, and `constants_mod` defines physical and mathematical constants.

The implementation details and interface of each module are presented after a description of the discretisation process.

3.1.1 Discretisation

Several versatile software packages for mesh generation and finite element method (FEM) analysis exists. Among these are *Netgen/NgSolve* [35] and *Gmsh* [2]. They both use high performance unstructured meshing algorithms to discretise arbitrary surfaces and volumes. The surface mesh elements may be simplices or quadrilaterals, or even a mix, and the shape of the elements may be locally altered using mesh adaptation. Unstructured and adaptive meshing allows the user to chose a finer discretisation in specific areas to increase numerical accuracy of the results, e.g. near the boundaries of an open surface, while using coarser discretisation resolution in less important regions to reduce the computational cost. Additionally, in FEM analysis, it may be useful to distinguish elements according to materials or boundaries. In Gmsh, such grouping of entities is accomplished through *physical groups*. It is also possible to easily increase the order of the element edges from linear to quadratic and cubic. In this project, surface meshes was produced by Delaunay triangulation (see Section 2.3) using the Gmsh software package.

The generated surface and volume meshes may be exported to various formats, both by Netgen and Gmsh. Normally, the format represents the mesh through the coordinates of the discretisation nodes and the nodes each element is composed of. The module `mesh_mod` is designed such that it is capable of reading and storing a surface mesh represented by any format. Currently, it is compatible with the *MSH* ASCII file format version 2 (*Gmsh2*), a fileformat available in both Gmsh and Netgen. Moreover, tools are available in the I/O module (to be detailed later) to ease the expansion of format versatility. An example of a surface mesh in the *Gmsh2* format is presented in Appendix C.1.

3.1.2 Modules

As stated above, several of the modules are depended on other modules. Figure 3.1 shows a module diagram listing every module and their dependencies, indicated by arrows, where the module at the arrowhead is the dependee. A dashed arrow indicates inheritance with the arrowhead pointing at the descendant. It is implicit that the inheritance applies to the main type of module. Notice that there are only single headed arrows, meaning there are no circular dependencies, which is a useful feature when compiling Fortran code. A description of all modules shown in the figure will follow in this section. Note that this diagram does not distinguish direct from indirect dependency. As an example, the module `PMCHW_RWG_mod` depends directly on `working_precision_mod` and `RWG_basis_mod`, while it depends indirectly on `mesh_mod`.

`mesh_mod`

The module `mesh_mod` was designed to completely represent a discretised surface, be it closed or open, triangular or quadrilateral, linear or cubic. Consequently, it is versatile in the sense that you can use it together with different types of basis

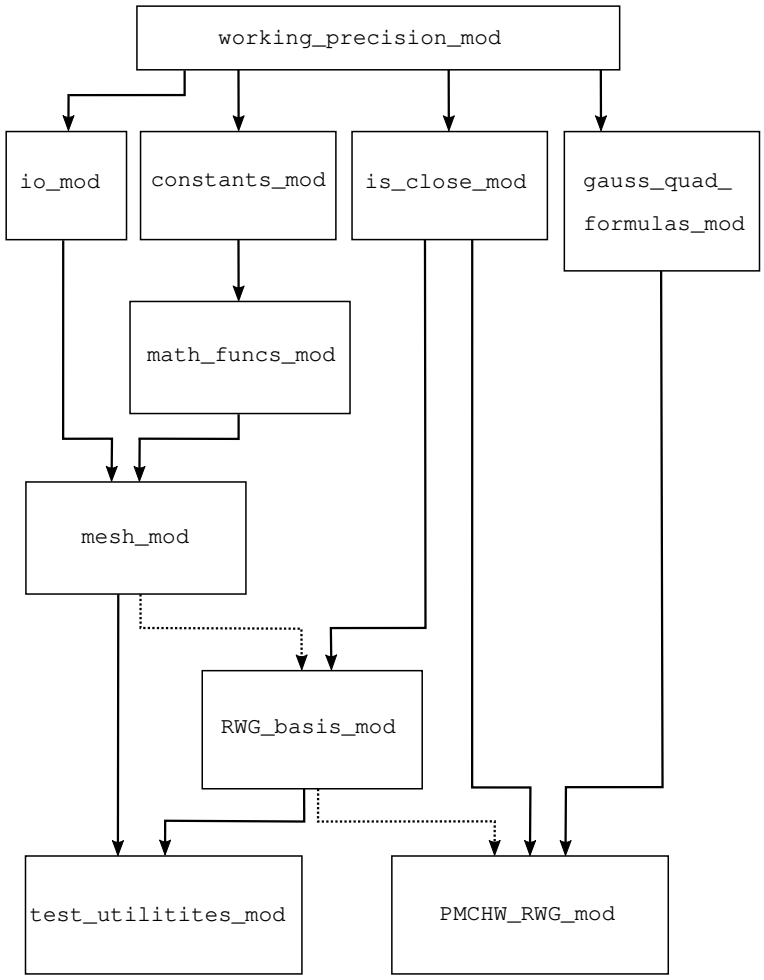

Figure 3.1: A module diagram illustrating the dependencies between the modules used in the simulation program. Each box is a module and an arrow represents an dependence, where the arrowhead points towards the dependee. A dashed arrow represents inheritance of derived types.

functions, and it is generic in the sense that it may represent a broad group of surfaces.

The module consists of a main derived type called `mesh_mod_type` and its member procedures, a few constants, and some additional public and private procedures. The interface of its Fortran file *mesh_mod.f90* is printed in Appendix B.1 for reference.

The data structure of `mesh_mod_type` determines the memory cost of representing the mesh. It also determines the computational cost of accessing specific entities. Storing only the coordinates of each entity will be memory efficient, but it will be significantly computationally expensive to locate neighbouring entities. In FEM and MoM, basis functions are usually local, consisting of adjacent entities, such that storing adjacency relations will be computational efficient. However, storing every possible adjacency relation for every entity consumes a lot of memory, so it is advantageous to balance the amount of adjacency relations with the computational cost of retrieving them. Consider a mesh entity $M_i^{d_i}$, where d_i is the entity order. A vertex has entity order 0, while an edge and a face have orders 1 and 2, respectively. Let $M_i^{d_i}\left\langle M^{d_i-1}\right\rangle$ be the set of lower order entities that lies on the boundary of $M_i^{d_i}$. E.g. the set of edges on the boundary of a triangle M_i^2 could be

$$M_i^2[M_\pm^1] = [M_{+i}^1, M_{-j}^1, M_{+k}^1],$$

where the indices $i \neq j \neq k$ and the $\pm$ subscript indicate the orientation of the edge in relation to the orientation of the adjacent face. The derived type `mesh_mod_type` represents a mesh by storing the following relations

$$M_i^2[M_\pm^1], M_i^1\lfloor M^0 \rfloor, \tag{3.1}$$

for all faces and edges. For a closed triangulated surface, this requires the storage of

$$3N_f + 2N_e = \frac{12}{2}N_e,$$

relations. Here N_f and N_e is the number of faces and edges in the triangulation, respectively. The three dimensional Cartesian coordinates of the vertices, M_i^0, are stored as well to complete the spatial representation. The adjacency relations in Eq. (3.1) are of first order, so that `mesh_mod_type` has a topology-based and hierarchical data structure [36], making it efficient to retrieve the coordinates of both faces and edges, as well as retrieving the boundary edges of a face. An example of a second order adjacency relation is the set of faces which share a specific vertex with a particular face.

Tables 3.1 and 3.2 shows the attributes and member procedures of `mesh_mod_type`. The dimension `(1)` signifies an array of rank 1 with a single element, i.e. a scalar. The dimension `(4, 3)` would be a rank 2 array with dimensions 4 and 3, i.e. a 4 by 3 matrix. If the dimensions are colons instead of numbers, then the attribute is a dynamically allocatable array. As such, the attributes `faces`, `edges`, `vertices`, and `nodes` are all allocatable rank 1 arrays of the derived type `edge_type`,

Attributes of `mesh_mod_type`		
Name	**Type**	**Dimension**
`faces`	`face_type`	`(:)`
`edges`	`edge_type`	`(:)`
`vertices`	`vertex_type`	`(:)`
`nodes`	`node_type`	`(:)`
`face_order`	`integer`	`(1)`
`edge_order`	`integer`	`(1)`
`spatial_dim`	`integer`	`(1)`
`num_faces`	`integer`	`(1)`
`num_edges`	`integer`	`(1)`
`num_vertices`	`integer`	`(1)`
`num_nodes`	`integer`	`(1)`
`num_handles`	`integer`	`(1)`
`num_apertures`	`integer`	`(1)`
`num_boundary_edges`	`integer`	`(1)`
`closed_surface`	`logical`	`(1)`

Table 3.1: The names, type and dimension of the attributes of the derived type `mesh_mod_type`, which is the main type of the module `mesh_mod`. All attributes are rank 1 arrays, most of them are scalars or vectors dimension 1 (identified by `(1)`), and some of them have a dynamically allocatable dimension (identified by the colon, `:`).

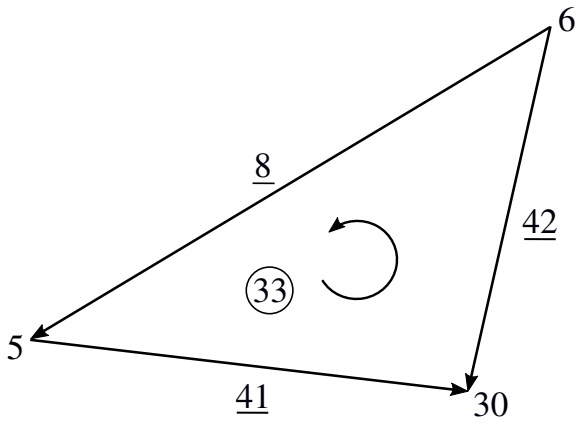

Figure 3.2: A face of order 3 with edges of order 1. The numbers represents index identifiers of the entities. A number surrounded by a circle represents the index of a face, an underlined number the index of an edge, and a normal number the index of a vertex. The orientations of the edges are indicated by the arrowheads. The orientation of the face is indicated by the curled arrow in the centre of the face. The resulting direction of the face's normal vector is obtained by using the right hand rule on the face orientation.

`vertex_type`, and `node_type`, respectively. These derived types are also defined in `mesh_mod`.

The derived type `face_type` has itself one attribute; an allocatable array of integers identifying which particular edges the face is enclosed by. Technically, these identifiers represents index numbers of the array `edges`. In this way, `face_type` stores the adjacency relation $M_i^2[M_\pm^1]$. Similarly, `edge_type` stores the relation $M_i^1\lfloor M^0 \rfloor$ (the vertices of which lies on the boundary of an edge) by having an array of two integer identifies representing index numbers of the array `vertices`. Finally, the `vertex_type` consists of only a single scalar integer attribute, pointing to an index location in the array `nodes`, where `node_type` holds and array of 3 floating numbers (type `real`), representing the Cartesian coordinates of the node.

Why have the array `nodes` in addition to the array `vertices`? Do they not represent the same topological entity? Yes, and no. The derived type `edge_type` has in fact an additional allocatable rank 1 array, where the integers represents index numbers in the array `nodes`. This is because an edge may be quadratic or cubic instead of linear, requiring storage of one or two additional nodes (see Section 5.2 in Ref. [36]). However, the total number of vertices in the mesh is independent of the edge order, introducing the need to distinguish between nodes and vertices.

In terms of physical calculations it is important that the normal vector of every face points in the same direction. If the mesh represents a closed surface the desired directions of the face normals are outward. Whether a face normal is inward or outward is determined by the the right hand rule with the face orientation, see Fig. 3.2. The current direction in a RWG basis function (see Section 2.2.3) also depends on the orientation of the basis edge. Therefore, it is desired to store the orientation of faces and edges along with its adjacency relation. The orientation

of an `edge_type` is easily stored indirectly by defining the positive direction from the vertex represented by the first element in its array `vertices`, to the vertex represented by the second element in the same array. Similarly, the orientation of a `face_type` may be defined by the order of the edge indices in its array `edges`. The face and edges in the example Fig. 3.2 are then stored as,

```
faces(33)%edges = [ 41, 42, 8]
edges(41)%nodes = [ 5, 30 ]
edges(42)%nodes = [ 6, 30 ]
edges(8)%nodes = [ 6, 5 ]
```

where `edges(41)` accesses the `edge_type` element of index 41 in the `edges` array, listed in table 3.1. Here the % operator accesses a derived type's attributes. In the case above, the attributes are the first order adjacency relation sets

$$M^2_{33}[M^1],$$
$$M^1_{41}\lfloor M^0 \rfloor,$$
$$M^1_{42}\lfloor M^0 \rfloor,$$
$$M^1_{8}\lfloor M^0 \rfloor,$$

respectively.

The mesh format that is currently supported by the I/O module, *Gmsh2*, contains the nodes and faces of a mesh. The nodes are represented by their Cartesian coordinates and a designated index, and the faces are represented by the node indices of which the face is comprised (faces are called elements by the format). The order of the node indices is such that, by following our definition of face orientation above, all faces have normal vectors directed outwards. If this was not the case, careful ordering of the constructed edges would have been necessary when storing the relations $M^2_1[M^1]$. An example of such careful ordering is described in the appendix of Ref. [26].

The attributes `num_faces`, `num_edges`, `num_vertices`, and `num_nodes` in Table 3.1 are helping integers representing the number of faces, edges, vertices, and nodes, respectively. These are equal to the size of the arrays `faces`, `edges`, `vertices`, and `nodes`, respectively. The integers `num_handles`, `num_apertures`, and `num_boundary_edges` are additional topological parameters, as by the definitions in Section 2.3.1 storing N_h, N_a, and N_b, respectively. They are used to validate the mesh using topological evaluations such as Eq. (2.64). The integer `face_order` stores the number of vertices per face, and `edge_order` indicates whether the edges are linear, quadratic, or cubic. Lastly, `closed_surface` is of Boolean data type, and indicates whether the surface mesh has a closed or open topology.

Table 3.2 lists some of `mesh_mod_type`'s member procedures. Even though there are many public member functions, the user only needs to utilise `initialise` to create a usable mesh object. This routine takes a series of arguments; the path and file format of a mesh-file, along with certain parameters describing the surface, such as edge and face order, spatial dimension, and topological parameters. The

Selected procedures in `mesh_mod_type`	
Name	**Encapsulation**
`initialise`	`public`
`get_edges_on_face`	`public`
`get_vertices_of_face`	`public`
`get_face_coords`	`public`
`face_normal`	`public`
`volume`	`public`
`edge_length`	`public`
`face_area`	`public`

Table 3.2: The names and encapsulation of selected member procedures of the derived type `mesh_mod_type`, the main type of the module `mesh_mod`.

amount of topological parameters required depends on the edge and face order, and whether the surface is open or closed. The routine `initialise` calls on functionality in `io_mod` to read the given file, then it validates its arguments together with data from the file, and finally allocates and initialises the attributes of its derived type, `mesh_mod_type`.

The function `get_edges_on_face` is an example of a *get*-function which returns a first order adjacency relation set, in this case the edges on the boundary of a face. Because of the data structure of `mesh_mod_type`, this connectivity fetch has very few operations and thus a very low computational cost. The functions `get_vertices_of_face` and `get_face_coords` are also connectivity fetches with low computational cost because of the hierarchical data structure. They return the indices and Cartesian coordinates of the vertices on the closure of the face, respectively. On the other hand, the latter get-functions are a bit more computationally expensive than `get_vertices_of_face`, since the relation $M_i^2[M^0]$ is not stored directly, but via Eq. (3.1).

The derived type `mesh_mod_type` has several calculation procedures, such as `face_normal` which uses `get_face_coords` to calculate the unit normal vector of a particular face, and `volume` which calculates the volume of a closed surface mesh using `face_normal` and other procedures while iterating through the faces of the mesh. The procedure `edge_length` is another useful function, especially when mapping onto RWG basis functions. Indeed, all of the calculation procedures are very useful for validating the mesh representation. As is done by the testing programs described in Section 3.2.

`RWG_basis_mod`

Now, using `mesh_mod`, we have a way reading, storing, and accessing an arbitrary discretised surface. However, we want to apply MoM using RWG basis functions, giving the need to evaluate integrals over these basis functions. An RWG basis, shown in Fig. 2.2, consists of two triangular faces sharing an edge. Hence, on a

Attributes of `RWG_basis_mod_type`		
Name	**Type**	**Dimension**
`mesh`	`mesh_type`	`(1)`
`basis_edges`	`integer`	`(:)`
`adjacent_faces`	`integer`	`(:, :)`
`num_bases`	`integer`	`(1)`
`basis_edge_length`	`real`	`(:)`

Table 3.3: The names, types and dimensions of the attributes of the derived type `RWG_basis_mod_type`, the main type of the module `RWG_basis_mod`. Dynamically allocatable dimensions are signified by the colon, `:`.

closed surface, the number of RWG basis functions, N, equals the number of edges, while on an open surface there will be noncontributing boundary edges such that

$$N = \begin{cases} N_e : & \text{closed surface,} \\ N_e - N_b : & \text{open surface,} \end{cases} \tag{3.2}$$

where N_e and N_b are the number of edges and boundary edges on the surface, respectively.

A surface integral over the RWG basis S_m means integrating over both faces $T_m^{\pm}$ adjacent to the basis edge L_m. However, the adjacency relation

$$M_i^1\{M^2\},$$

i.e. the set of faces sharing the edge M_i^1, is not stored in the data structure of `mesh_mod_type`. Adding this relation to the data structure would reduce the computational cost when iterating through the bases and solving the integrals. On the other hand, adding this adjacency relation to `mesh_mod_type` will reduce the versatility of the mesh representation, as it may be superfluous to other basis functions and unnecessarily use of memory space. Additionally, if the surface is open, the relation is not necessary for every edge.

Another option is to derive a new type, who inherits `mesh_mod_type`, selects the edges which form a basis, and store their adjacency relation $M_i^1\{M^2\}$. This is the purpose of the main derived type in the module `RWG_basis_mod`, called `RWG_basis_mod_type`.

Table 3.3 lists the attributes of `RWG_basis_mod_type`. See Appendix B.2 for the full interface of the module.

The inheritance of `mesh_mod_type` is explicitly stated with the first attribute `mesh`. Essentially, this yields `RWG_basis_mod_type` all attributes listed in Table 3.1 as well, easily accessed via the `mesh%` operator.

The attribute `basis_edges` is an allocatable array of integers identifying which edges in `mesh` that feature an RWG basis. If the surface is closed, then `basis_edges` equals `mesh%edges`.

Procedures in `RWG_basis_mod_type`	
Name	**Encapsulation**
`initialise`	`public`
`deallocate_attributes`	`public`
`get_free_vertices`	`public`
`get_basis_edge_coords`	`public`
`get_basis_edge_length`	`public`
`get_adjacent_faces`	`public`
`get_num_bases`	`public`
`integrate_tested_func`	`public`

Table 3.4: The names and encapsulation of all member procedures of `RWG_basis_mod_type`, the main type of the module `RWG_basis_mod`. Their implementations are printed in Appendix B.2.

The matrix `adjacent_faces` contains the set $M_i^1\{M^2\}$ for all edges in `basis_edges`. Hence, its first dimension is always equal to the dimension of `basis_edges`, while its second dimension is always equal to 2.

Furthermore, the integer `num_bases` is a helping quantity storing the size of `basis_edges`.

The RWG basis function $\mathbf{f}_m(\mathbf{r})$ (see Eq. (2.57)) contains the length of the basis edge L_m. Often it is necessary to integrate over the RWG bases multiple times with different integrands, as in the double integrals in Eq. (2.95), the vector (2.101), and the field distributions defined by Eq. (2.158). Thus, storing L_m in `RWG_basis_mod_type` will improve the computational efficiency.

All member procedures of `RWG_basis_mod_type` are listed in Table 3.4.

Similarly to `mesh_mod_type`, the user only needs to run the member routine `initialise` to be able to use the derived type in calculations. The routine `initialise` takes a single argument; an instance of the `mesh_type`. It then uses the topological parameters and variables included with the `mesh_type` to find the edges corresponding to an RWG basis function and stores their adjacency relation.

The routine `deallocate_attributes` deallocates the member attributes, if allocated.

The function `get_free_vertices` returns the set of free vertices $(\mathbf{p}_m^+, \mathbf{p}_m^-)$ of the basis function. This operation is not very expensive, since it is only a matter of finding the vertices of $T_m^\pm$ which are not common with the vertices of the basis edge.

The functions `get_basis_edge_length` and `get_adjacent_faces` simply returns the corresponding attribute of `RWG_basis_mod_type`, while the function `get_basis_edge_coords` calls on internal procedures of the `mesh_type` attribute `mesh` to get the spatial coordinates of the vertices of a basis edge.

The function `integrate_tested_func` is designed to evaluate the quantity

$$\sum_{m=1}^{N_{\mathrm{RWG}}} \int_{S_m} \mathrm{d}S \mathbf{f}_m(\mathbf{r}) \cdot \mathbf{g}(\mathbf{r}), \tag{3.3}$$

where $\mathbf{g}(\mathbf{r})$ is an arbitrary function, passed as an argument. The integrals are solved by the method of Gaussian quadrature, transforming the triangle onto barycentric coordinate system, as shown in Section 2.4. The integrals are also calculated face-by-face, not basis-by-basis, reducing the number of integral evaluation by a factor of 9. Face-by-face evaluation was introduced in Section 2.6.6. The function `integrate_tested_func` is very useful for validating the stored basis functions and mesh structure.

`PMCHW_RWG_mod`

The electromagnetic scattering problem simulated in this thesis is solved by MoM using RWG basis functions, Galerkin's method of weighted residuals, and the PMCHW-formulation for combining the EFIE and MFIE. Nevertheless, RWG basis functions may be used with other methods than Galerkin's, and other combined field formulations. E.g if region 2 in Fig. 2.1 is a perfect conductor, then the electrical field inside the conductor would vanish, and the magnetic current $\mathbf{M}(\mathbf{r}')$ in Eq. (2.95) would be zero, reducing the problem to solving

$$\boldsymbol{\mathcal{D}}^{(1)}\psi_\alpha = \mathbf{q}_\alpha^E \tag{3.4}$$

for

$$\psi_\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_{N-1}, \alpha_N]^T, \tag{3.5}$$

where $q_{\alpha n}^E = q_n^E$ for $n = 1, 2, \ldots, N$. Thus, it is useful to separate the module storing the RWG basis functions from the module performing the calculations and solving the matrix equation. This is the reasoning behind implementing the module `PMCHW_RWG_mod`, specialised in solving the matrix equation (2.99).

The attributes of the main derived type in `PMCHW_RWG_mod` are listed in Table 3.5, and the entire interface of the module is printed in Appendix B.3. The module inherits the discretised surface mapped onto RWG basis functions through the attribute `RWG_basis`. The variables `permeabilities`, `permitivities`, and `angular_frequency` characterises the electromagnetic scattering problem. They represent the permeability (μ_i) and permitivity (ϵ_i) of both regions i, and the angular frequency (ω) of the incident electromagnetic wave, respectively.

The attribute `PMCHW_matrix` stores the matrix defined by the PMCHW-formulation, that is, it stores Eq. (2.100). Hence, its dimensions are $(2N, 2N)$.

Furthermore, the array `q_vectors` stores the vector $\mathbf{q}$ in Eq. (2.99). The attribute is of rank 2, allowing several variations of $\mathbf{q}$, corresponding to different values of $\mathbf{E}_1^{\text{inc}}(\mathbf{r})$ and $\mathbf{H}_1^{\text{inc}}(\mathbf{r})$. Allowing variations is computationally efficient, as one might want to solve ψ for a set of $\mathbf{q}$ and a constant matrix using LU-decomposition. The number of $\mathbf{q}$ variations is stored in the attribute `num_q_vectors`.

The matrices `expansion_coeff_alpha` and `expansion_coeff_beta` will store the solution ψ to the matrix equation (2.99), when solved. These attributes are also of rank 2 for the same reasons as `q_vectors`.

Moreover, the attributes `inc_E_field_amp` and `inc_H_field_amp` stores the complex amplitudes, of the incident electric and magnetic field, respectively. Thus,

Attributes of PMCHW_RWG_mod_type		
Name	**Type**	**Dimension**
RWG_basis	RWG_basis_type	(1)
permeabilities	complex	(2)
permitivities	complex	(2)
angular_frequency	real	(1)
PMCHW_matrix	complex	(:, :)
q_vectors	complex	(:, :)
expansion_coeff_alpha	complex	(:, :)
expansion_coeff_beta	complex	(:, :)
inc_E_field_amp	complex	(:, :)
inc_H_field_amp	complex	(:, :)
inc_wave_direction	real	(:, :)
inc_field_type	integer	(:)
num_q_vectors	integer	(1)

Table 3.5: The names, types and dimensions of the attributes of the derived type PMCHW_RWG_mod_type, the main type of the module PMCHW_RWG_mod. Dynamically allocatable dimensions are signified by the colon, :.

they have first dimensions equal to the spatial dimension of the scattering problem, i.e. first dimensions equal to 3. Since variations of $\mathbf{q}$, while having a constant matrix $\boldsymbol{\mathcal{H}}$, essentially constitute variations in the incident field (see Eq.(2.101)), the second dimension of inc_E_field_amp and inc_H_field_amp equals the number of different $\mathbf{q}$-vectors to solve the matrix equation (2.99) for. The attribute inc_wave_direction represents the unit wave vector $\hat{\mathbf{k}}$ of the incident electromagnetic wave, i.e. the direction of propagation of the incident wave. As follows, the attribute have the same dimensions as inc_E_field_amp and inc_H_field_amp. The wavevector $\mathbf{k}$ of the incident wave is then found by multiplying the wavenumber

$$k_i = \omega\sqrt{\epsilon_i \mu_i},$$

with the unit wavevector $\hat{\mathbf{k}}$.

The attribute inc_field_type indicates whether the incoming electromagnetic wave is planar or spherical.

Selected member procedures of PMCHW_RWG_mod_type are listed in Table 3.6. They are specifically chosen to demonstrate the implementation and numerical methods chosen to calculate and solve the matrix equation (2.99).

Contrary to mesh_mod_type and RWG_basis_mod_type, calling PMCHW_RWG_mod_type's routine initialise is not sufficient to solve the matrix equation (2.99). The routine initialise only initialises the problem specific attributes permeabilities, permitivities, and angular_frequency, allocates memory space for the attribute PMCHW_matrix, and sets the attribute RWG_basis, all according to the arguments passed with the routine call.

Selected procedures in `PMCHW_RWG_mod_type`	
Name	**Encapsulation**
`initialise`	`public`
`calc_PMCHW_matrix`	`public`
`calc_q_vectors`	`public`
`solve_matrix_equation`	`public`
`E_and_H_field_at_obs_pnt`	`public`

Table 3.6: The names and encapsulation of all member procedures of `RWG_basis_mod_type`, the main type of the module `RWG_basis_mod`. Their implementations are printed in Appendix B.2.

The elements of `PMCHW_matrix` are calculated and set when calling the member routine `calc_PMCHW_matrix`. The reason for performing this task in a separate routine call is to allow resetting the matrix using different Gaussian quadrature formulas, without having to reinitialise the instance. The quadrature formulas are required to be passed as arguments. The routine `calc_PMCHW_matrix` uses, similarly to the procedure `integrate_tested_func` of `RWG_basis_mod_type`, Gaussian quadrature formulas on the form of Eq. (2.69) to evaluate the double integrals in Eq. (2.95). By switching and argument, the user may opt to evaluate the integrals using a face-by-face approach or basis-by-basis approach. A face-by-face approach will reduce the number of integral evaluations by a factor of 9 (see Section 2.6.6), and consequently reducing the computing time, but will increase the memory usage.

The routine `calc_q_vectors` takes the amplitudes and direction of the incoming electromagnetic waves as arguments, along with a Gaussian quadrature formula for triangles, and calculates the elements of $\mathbf{q}$ via Eq. (2.101). The routine stores the results in the attribute `q_vectors`, in addition to storing the characteristics of the incident wave in their suitable attributes. Storing these characteristics lets the user call the function `E_and_H_field_at_obs_pnt`, which uses Eq. (2.158) to calculate the electric and magnetic field distribution, without re-specifying the characteristics of the incident wave. This implementation also prevents the user from calculating the field distribution erroneously by using a different incident wave than the one used to calculate $\mathbf{q}$. The arguments of `calc_q_vectors` may have several columns, resulting in various $\mathbf{q}$-vectors.

Both the calculation of the matrix elements $\mathcal{D}_{mn}^{(i)}$ and $\mathcal{K}_{mn}^{(i)}$ in $\boldsymbol{\mathcal{H}}$, and the field distributions defined by Eq. (2.158), involves evaluation of the scalar Green's function for homogeneous media, $G_i(\mathbf{r}, \mathbf{r}')$, defined by Eq. (2.18). Its singularity is treated carefully by the methods presented in Section 2.6.3. In the calculation of matrix elements, the Green's function is always considered near singular when the faces, of which the double integrals are evaluated over, share one or more vertices. If the faces do not share a vertex, then the separation distance between the faces' centroid are compared with a constant, proceeding with singularity subtraction if the separation distance is the smallest. When calculating the field distributions, the separation

distance between the observation point and the quadrature points of the relevant face is compared with the same constant. The value of this constant is based on the computer systems ability to accurately represent the value $1/R^3$ for very small R. If $G_i(\mathbf{r}, \mathbf{r}')$ is considered near singularity, it is replaced by its smoothed version $G_i^s(\mathbf{r}, \mathbf{r}')$, defined by Eq. (2.105), and the integration is performed numerically using Gaussian quadrature formulas for triangles. The integrals over the subtracted terms are evaluated by the public routine `eval_subtracted_terms` using the exact iteration formulas (2.142), (2.143) and (2.140). When relevant, the outer testing integral is evaluated also using Gaussian quadrature. Because of their dependence on the free vertices of the basis functions, $\mathbf{p}_n^{\pm}$, the integrals of the subtracted terms must be evaluated on a basis-by-basis approach, which is why they are handled separately from the integrals over $G_i^s(\mathbf{r}, \mathbf{r}')$, which may be evaluated face-by-face. The double integral of the subtracted term $\nabla' R^{-1}$ has special treatment. It is evaluated as a line integral on the form of Eq. (2.157). The integrals over the face edges are transformed to 1D integrals with limits $[-1, 1]$ by parameterising the source point $\mathbf{r}'$, as shown in Section 2.6.5. On this form, they are evaluated using Gauss-Legendre quadrature.

Finally, the matrix equation (2.99) is solved by calling the routine `solve_matrix_equation`. This routine takes a single optional argument, an integer representing the numerical method used to solve the matrix equation. Its default is to use LU-decomposition [29, 30]. All matrix solvers are imported from the OpenBlas Fortran library [37] and the particular precision of the solver is selected based on the value of the working precision parameter `wp` defined by the module `working_precision_mod` (described below). As the design of `PMCHW_RWG_mod` dictates, `solve_matrix_equation` has to be called after calling the routines `initialise`, `calc_q_vectors`, and `calc_PMCHW_matrix`. Thus, the algorithm, or the sequential procedure calls, or the pseudo code, to fully solve the scattering problem, is shown in Fig. 3.3. Starting with a file representing a surface mesh in a supported format, pass it to the member procedure `initialise` of `mesh_mod_type` and pass the resulting instance to `initialise` of `RWG_basis_mod_type`. Then initialise `PMCHW_RWG_mod_type` by passing the latter instance, call its routines `calc_PMCHW_matrix` and `calc_q_vectors` with required arguments in either order. Then solve Eq. (2.99) by calling `solve_matrix_equation` of `PMCHW_RWG_mod_type`. In the end, calculate the field distribution at any given observation point by repeatedly calling the function `E_and_H_field_at_obs_pnt`.

`io_mod`

This I/O module contains all functionality concerned with input and output. There are no main type, only a series of public and private procedures with specific purposes. All procedures are listed in Table 3.7

The routine `open_read_gmsh2` is concerned with opening and reading a mesh file with the *gmsh2* format[1]. This format lists all nodes and elements in the mesh. A single node is given by a row of four numbers; an integer representing its index

[1]The file format is described at http://www.manpagez.com/info/gmsh/gmsh-2.2.6/gmsh_63.php

```
use mesh_type from mesh_mod
use RWG_basis_type from RWG_basis_mod
use PMCHW_RWG_type from PMCHW_RWG_mod

mesh_type%initialise(mesh_filename, topological_parameters, ...)
RWG_basis_type%initilise(mesh)
PMCHW_RWG_type%initialise(RWG_basis, physical_parameters, ...)
PMCHW_RWG_type%calc_q_vectors(incident_EM_wave_characteristics)
PMCHW_RWG_type%calc_PMCHW_matrix(Gaussian_quadrature_formula)
PMCHW_RWG_type%solve_matrix_equation(numerical_method_to_use)

loop over observation points
    field_distribution = PMCHW_RWG_type%E_and_H_field_at_obs_pnt(
                                                    observation_point)
end loop
```

Figure 3.3: Pseudo code of the algorithm the a program uses to simulate the two region scattering problem. The top three lines describes the derived types that are used directly by the program, and which module the type is defined within.

Procedures in `io_mod`	
Name	**Encapsulation**
`open_read_gmsh2`	`public`
`string_to_int4`	`public`
`string_to_real_wp`	`public`
`read_nth_int4`	`public`
`read_n_last_int4`	`public`
`read_n_last_real_wp`	`public`
`count_int4_on_string`	`public`
`count_real_wp_on_sting`	`public`
`capitalise_char`	`public`
`r8mat_write`	`public`
`get_unit`	`public`
`check_ioerr_opening`	`private`
`check_ioerr_reading`	`private`

Table 3.7: The names and encapsulation of all procedures defined in `io_mod`. Their interface of the module and the routine `open_read_gmsh2` are printed in Appendix B.4.

number, and three floating numbers representing its Cartesian coordinates in three dimensional space. E.g. the line

```
43 3.56 -43.4 5.3
```

describes node number 43 positioned at $(x, y, z) = (3.56, -43.4, 5.3)$.

An element is given by a row of 5+ integers, e.g

```
2 2 1 0 1 3 4
```

where the first integer is the element number, the second is the element type (2 describes a triangle), the third number signals the number of tags followed, and the last three integers are the node numbers on the closure of the element. The tags are used to separate physical from geometrical entities, or to partition the mesh.

The row sections of nodes and elements are separated by key lines such as `$Nodes`, `$EndNodes`, `$Elements`, and `$EndElements`. As a result of this varying number of columns on each row, and the alternations between integers and decimals, `open_read_gmsh2` needs to be carefully designed, reading line by line, while separating the process into several sections or *levels*. Level one is concerned with finding the key line `$Nodes`. Level two stores the integer on the following line which represents the total number of nodes. Level three reads node-rows such as the one above until it reaches `$EndNodes`, and so on. Functions such as `string_to_int4`, `string_to_real_wp`, `read_n_last_int4`, `read_n_last_real_wp`, and `read_nth_int4` are used by `open_read_gmsh2` to read all or specific integers and decimals in the lines of the file. After reading and storing all nodes and elements, their values are validated by checking for invalid integers and decimals, checking for duplicates etc.

As noted above, there is only functionality for the reading of Gmsh2 formated files, but using the already existing framework of reading procedures, implementation of further format compatibility is facilitated.

In addition to reading procedures, `io_mod` includes procedures for writing matrices to files, which is essential to save the simulation results.

The procedures `r8mat_write` and `get_unit` are entirely written by John Burkardt [33], while the rest of the procedures in the module are strongly influenced by the design of John Burkardt's code in *gmsh_io.f90* (See Ref. [33]).

`math_funcs_mod`

General mathematical functions and operations that are required by any module are implemented as procedures in `math_funcs_mod`. Examples of procedures are `cross_prod_3D` which calculates the cross product of two vectors in three dimensional Cartesian space, and `plane_wave` which returns the value of a three dimensional plane wave at any position (in Cartesian coordinates) given by the wavenumber, direction, and amplitude.

`working_precision_mod`

It is crucial that the precision of all variables and attributes throughout every module are the same. If not, fatal errors may occur, or even worse, loss of precision.

Constant	Constants defined in `constants_mod` Name	Value
π	`PI`	`4._wp*atan(1._wp)`
μ_0	`PERMEABILITY_VACUUM`	`1.25663706212e-6_wp`
ϵ_0	`PERMITIVITY_VACUUM`	`8.8541878128e-12_wp`
Imaginary unit i	`I_IMAG`	`cmplx(0._wp, 1._wp)`
$0 + 0i$	`ZERO_CMPLX`	`cmplx(0._wp, 0._wp)`

Table 3.8: The constants defined as parameters in `constants_mod`. All numbers have a `_wp` post fix to set their precision equal to the working precision `wp` imported for `working_precision_mod`.

Therefore the parameter `wp`, representing the working precision of the program, is defined in the module `working_precision_mod` and successively imported by all and every other module, in which determination of precision is exclusively used by the parameter. To change the precision of simulation, simply change the value of `wp` in `working_precision_mod`.

`constants_mod`

It is useful to implement a module defining all physical and mathematical constants used by other modules to avoid differences in values and precision across modules. In this implementation, constants are accessed by importing `constants_mod`. The constants defined in the module are listed in Table 3.8.

`gauss_quad_formulas_mod`

This module was created to easily define and import Gaussian quadrature Formulas. It consists solely of rank 2 arrays, defined as parameters, representing a Gaussian quadrature formula including weights and abscissa values.

`is_close_mod`

The module `is_close_mod` contains a single elemental function definition called `is_close`. The purpose of the function is to compare two numbers of type `real` and returns `.true.` if the numbers are close to each other. The evaluation depends on a tolerance of absolute or relative difference. These tolerances are optional arguments to the function.

`test_utilities_mod`

This module contains utility procedures useful while testing the modules discussed in this section. Some of the procedures are concerned with run-time verbose, while others are specialised to test a particular module.

3.2 Testing

Thorough testing of the module procedures is an essential part of the validation of simulation results. Especially when the program uses several modules that depend on each other, it is important to test each functionality individually, making sure the dependencies are working correctly before testing the dependant. This testing technique is also known as *unit testing*. Each module used by the simulation program have a custom and adapted testing program, designed to test critical features of the corresponding module.

In this section, some of the testing methods used in each testing program will be described. Lastly, methods for testing the final results of the simulation program running as a whole will be presented.

3.2.1 Module Testing

io_test.f90

The module `io_mod` does not depend on any other module than `working_precision_mod`, which itself is so trivial that it, in practical terms, does not need testing. Therefore, testing of the I/O module is a good place to start.

Every procedure in `io_mod` (listed in Table 3.7) has its corresponding testing routine defined and called in the main program of *io_test.f90*. Most of the test are pretty straight forward, as the desired result after passing an arbitrary string as argument is trivial.

However, the routine `test_open_read_gmsh`, which tests the procedure `open_read_gmsh` in the module `io_mod`, uses a custom made Gmsh2 file and compares every resulting node and element with their expected value sequentially. The Gmsh2 file includes nodes and elements representing the surface of a tetrahedron, such as the one illustrated in Fig. 3.4. The contents of the file is printed in Appendix C.1. This exact surface mesh is also used when testing the modules `mesh_mod` and `RWG_basis_mod`, emphasising the need of this particular test.

mesh_test.f90

This program tests `mesh_mod` by verifying the geometry and topology of meshes imported using the member routine `initialise`. The function is tested with three different Gmsh2 files. One representing the regular tetrahedron (see Fig. 3.4), one representing the closed cube illustrated in Fig. 3.5, and one representing the open cube illustrated in Fig. 3.6. The following topological parameters and geometrical values are compared after initialising each surface mesh:

- The number of faces N_f,
- the number of edges N_e,
- the number of vertices N_v,

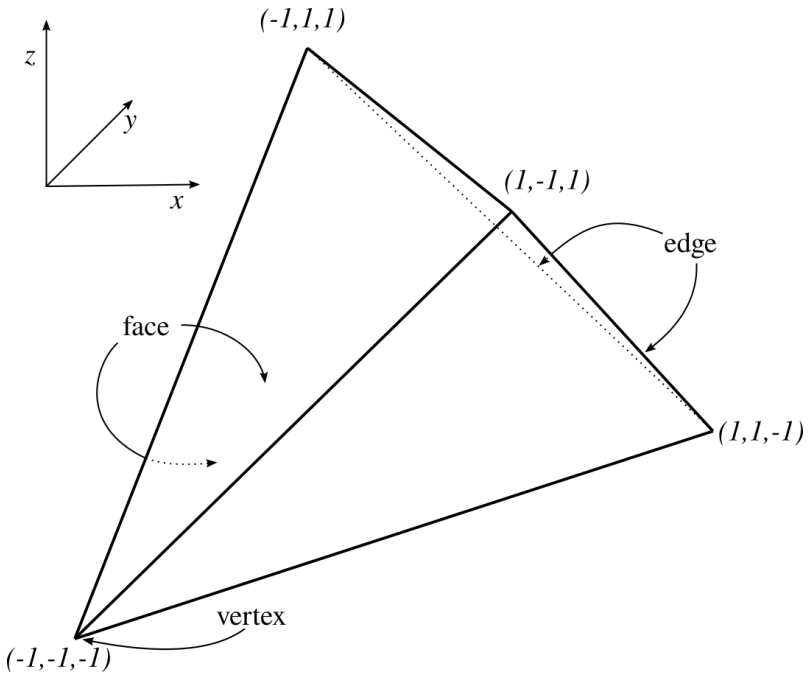

Figure 3.4: A regular tetrahedron with edge lengths of $2\sqrt{2}$. The face areas are $2\sqrt{3}$ with the resulting surface area of $8\sqrt{3}$. Its volume equals $8/3$.

- the number of nodes N_n,
- the number of handles N_h,
- the number of apertures N_a,
- the number of boundary edges N_b,
- the face areas,
- the surface area,
- and the volume (if applicable).

In addition to testing `initialise`, the functions `surface_area` and `volume` are tested individually. In this case a Gmsh2 file is not imported. Instead, a constructor defined within `mesh_mod` to initialise a surface mesh exactly equal to the tetrahedron is used, thus bypassing the many potential errors involved with importing the mesh from a file. The tetrahedron constructor is also tested in isolation itself.

RWG_basis_test.f90

The `RWG_basis_mod_type` object produced by the function `initialise` in `RWG_basis_mod` is tested using the same three surfaces meshes used by *mesh_test.f90*. This way, it is less probable that the mesh instance, passed as an argument, is the source of the problem. The following member attributes of the `RWG_basis_mod_type` objects are checked against expected values:

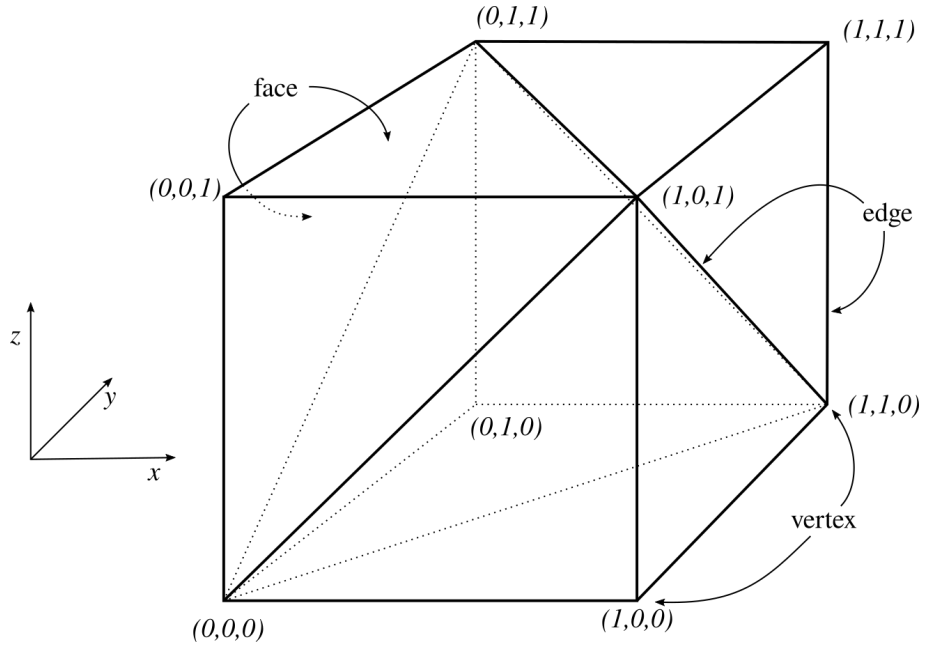

Figure 3.5: A triangulated cube with sides equal to unity. The face areas equals 0.5, the surface area 6, and the volume 1.

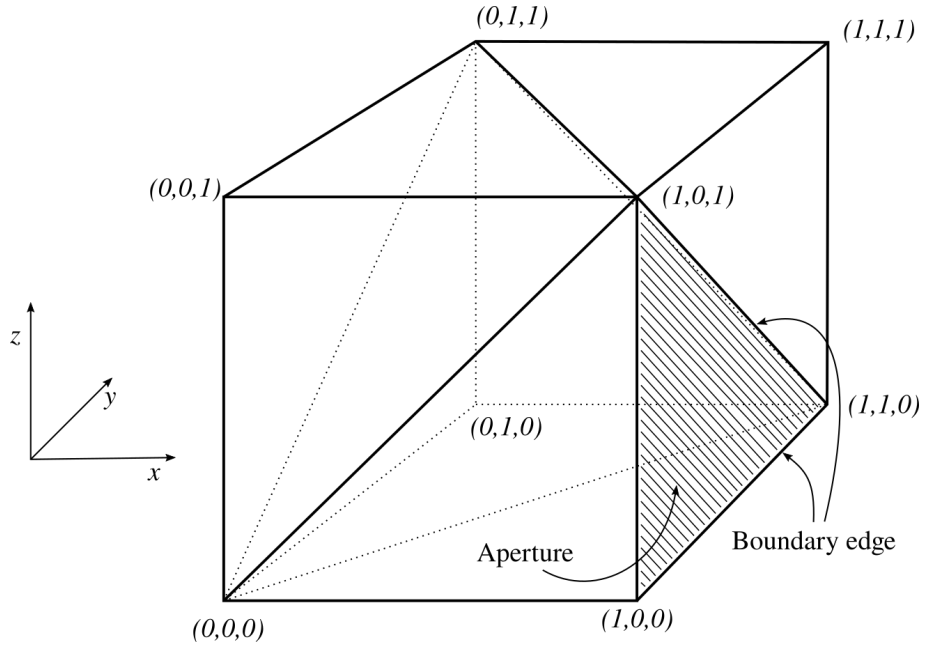

Figure 3.6: The triangulated cube in Fig. 3.5 with one of the faces removed, resulting in an open surface with one aperture and three boundary edges.

- The number of RWG bases N,
- the integer values and order of RWG basis edges,
- the integer values and order of the adjacent faces to each RWG basis,
- and the value of each basis edge length.

PMCHW_RWG_test.f90

The module `PMCHW_RWG_mod` has several procedures performing complex tasks involving delicate summations and integral evaluations. This makes for careful testing to verify the modules performance, as even a single erroneous summation sign might cause completely unrealistic results.

A majority of the procedures in `PMCHW_RWG_mod` uses Gaussian quadrature to evaluate integrals. These integrals are complicated, integrating exponential functions over triangles. Hence, it may not be possible to evaluate them analytically, making it difficult to validate the results of the procedures. One way to partly verify the procedures, is by testing cases where the integrands are simplified such that the integrals have analytical results. For instance, the sum (2.172a) equals

$$2A \int_0^1 \int_0^{1-\eta} \xi G_i(\mathbf{r}, \xi \mathbf{r}_1' + \eta \mathbf{r}_2' + \zeta \mathbf{r}_3') \mathrm{d}\xi \mathrm{d}\eta \tag{3.6}$$

by following Eqs. (2.69) and (2.72). This equation may be simplified to

$$2 \int_0^1 \int_0^{1-\eta} \xi \mathrm{d}\xi \mathrm{d}\eta \tag{3.7}$$

by setting the Green's function, $G_i(\mathbf{r}, \xi \mathbf{r}_1' + \eta \mathbf{r}_2' + \zeta \mathbf{r}_3')$ to unity. As this expression may easily be evaluated analytically, this simplification allows for testing the implementations of the Gaussian quadrature sums. Then, by making sure the evaluations of the Green's functions are correct, one may verify the procedures as a whole. The testing routines that uses integrand simplification to verify Gaussian quadrature integration are `test_E_and_H_field_at_obs_pnt`, `test_eval_green_func_integrals`, and `test_eval_outer_integrals`.

The functions `face_pair_integral_EFIE` and `face_pair_integral_MFIE` are tested by passing arbitrary defined arguments and comparing the results with analytical calculations.

The member procedure `inc_E_and_H_field_at_obs_pnt` of `PMCHW_RWG_mod_type` evaluates the incident electromagnetic field at a certain observation point. The field is evaluated analytically and thus, testing this routine is easily performed by comparing the procedure's result with the expected result.

Finally, the routine `test_solve_matrix_equation` uses the solution ψ produced by the member routine `solve_matrix_equation` to calculate the relative difference

$$\frac{1}{2N} \sum_{m=1}^{2N} \left| \frac{\sum_{n=1}^{2N} \mathcal{H}_{mn} \psi_n - q_m}{q_m} \right|, \tag{3.8}$$

where $\mathcal{H}_{mn}$ and q_m are element (m, n) and the mth element of $\boldsymbol{\mathcal{H}}$ and $\mathbf{q}$ in Eq. (2.99), respectively. All elements are gathered from the same instance of `PMCHW_RWG_mod_-type`, by which the solution ψ is calculated. With double precision, the expression (3.8) should take a value on the order of 10^{-15} if the the routine works correctly.

3.2.2 Validating Simulation Results

Energy conservation

In the case where both region 1 and 2 are non-magnetic materials ($\mu_i = \mu_0$ for $i = 1, 2$) and have real permitivities (ϵ_i), i.e. there is no absorption of the electromagnetic waves, the energy of the system as a whole will be conserved. Thus, assuming that the energy density stored in the electromagnetic field itself time averages to zero, and that the electromagnetic force do no work on any charges, the net rate of scattered energy crossing an imaginary sphere A of radius R surrounding the scattering region, which we denote by P_{out}, should be equal to net rate of energy incident on region 2, which we denote by P_{in}. The *Poynting vector* $\mathbf{S}$ is defined as the energy per unit time per unit area flowing through a surface [24]. Thus, we may define P_{out} as the integral of the time average of the Poynting vector to the scattered field over A, i.e.

$$P_{\text{out}} = \int_A \mathrm{d}S \left\langle \mathbf{S}_{\text{sca}} \right\rangle_t \cdot \hat{\mathbf{n}} \tag{3.9}$$

where $\hat{\mathbf{n}}$ is the unit normal pointing out of the imaginary sphere A. From Bohren and Huffman [8] we have the following definitions for time-harmonic fields

$$\left\langle \mathbf{S}_{\text{sca}} \right\rangle_t = \frac{1}{2}\mathrm{Re}\left\{ \mathbf{E}_1^{\text{sca}} \times (\mathbf{H}_1^{\text{sca}})^* \right\}, \tag{3.10}$$

$$\left\langle \mathbf{S}_{\text{inc}} \right\rangle_t = \frac{1}{2}\mathrm{Re}\left\{ \mathbf{E}_1^{\text{inc}} \times \left(\mathbf{H}_1^{\text{inc}}\right)^* \right\}. \tag{3.11}$$

Thus, for a spherical scatterer where C is the hemisphere facing the incident plane wave, we have

$$\begin{aligned} P_{\text{in}} &= -\int_C \mathrm{d}S \left\langle \mathbf{S}_{\text{inc}} \right\rangle_t \cdot \hat{\mathbf{n}} \\ &= \frac{1}{2}\int_C \mathrm{d}S \ \mathrm{Re}\left\{ \mathbf{E}_1^{\text{inc}} \times \left(\mathbf{H}_1^{\text{inc}}\right)^* \right\} \cdot \hat{\mathbf{n}} \\ &= \frac{1}{2}\int_C \mathrm{d}S \ \mathrm{Re}\left\{ \mathbf{E}_1^{\text{inc}} \times \frac{1}{Z}\left(\hat{\mathbf{k}} \times \mathbf{E}_1^{\text{inc}}\right)^* \right\} \cdot \hat{\mathbf{n}}, \end{aligned} \tag{3.12}$$

where the impedance $Z = \sqrt{\mu/\epsilon}$, and $\hat{\mathbf{n}}$ is the unit normal of C pointing out of the hemisphere. Using vector identity (2) in the back cover of Griffiths [24] yields

$$P_{\text{in}} = \frac{1}{2Z}|\mathbf{E}_1^{\text{inc}}|^2 \int_C \mathrm{d}S \hat{\mathbf{k}} \cdot \hat{\mathbf{n}} = -\frac{1}{2Z}|\mathbf{E}_1^{\text{inc}}|^2 \pi a^2, \tag{3.13}$$

where a is the radius of the scattering sphere. Since energy conservation dictates that $P_{\text{in}} = P_{\text{out}}$, we may use the following equation as a verification of the numerical results from a nonabsorbing, scatterer

$$\frac{1}{Z}|\mathbf{E}_1^{\text{inc}}|^2\pi a^2 = \int_A \mathrm{d}S \ \text{Re}\left\{\mathbf{E}_1^{\text{sca}} \times (\mathbf{H}_1^{\text{sca}})^*\right\} \cdot \hat{\mathbf{n}}. \tag{3.14}$$

The right hand side of Eq. (3.14) may be numerically evaluated using a Lebedev quadrature rule (see Section 2.4.3).

Symmetry

A plane electromagnetic wave, travelling in the $\hat{\mathbf{z}}$-direction, and arbitrary polarised, will have symmetry across the z-axis. Consequently, if the scattering surface has similar symmetry (e.g. a sphere centred in origo), the scattered electric field will have symmetry across the z-axis too, yielding a rudimentary test of the simulation results.

Interference pattern

Because of the linearity of Maxwell's equations, the scattered electric field $\mathbf{E}_1^{\text{sca}}$ will superpose with the incident electric field $\mathbf{E}_1^{\text{inc}}$ and produce interference patterns in the total field distribution $\mathbf{E}_1 = \mathbf{E}^{\text{sca}} + \mathbf{E}_1^{\text{inc}}$. The peak-to-peak distance of this interference pattern has a lower bound equal to the lowest wavelength among its components. When evaluating the field intensity, $|\mathbf{E}|^2$, the wavelength of the interference pattern is cut in half, such that when considering the two region electromagnetic scattering problem, the interference pattern of the field intensity has a lower limit $d_{\text{min}} = \lambda/2$, where λ is the wavelength of the incident plane wave.

Comparison to Mie theory

In the scenario where the scattering volume, region 2 in Fig. 2.1, is a homogeneous sphere with arbitrary radius and refractive index, we have an analytical solution to the Maxwell's equations, the Mie theory (see Section 2.5). The results from the numerical simulation should then be comparable to this solution, and this comparison is a powerful way of validating the simulation results.

The numerical implementation of the Mie solution was assessed from Appendix A in Ref. [8]. A Fortran 77 routine called `bhmie` returns the angle dependent Mie scattering coefficients S_1 and S_2, which may be used to calculate the elements of the Mueller matrix and the following Stokes parameters (see section 4.4.4 in Ref. [8]). In addition, the routine returns the *efficiency factors* for scattering and extinction, Q_{sca} and Q_{ext}, respectively, and the scattering angles θ, which is the angle between the scattering direction $\hat{\mathbf{r}}$ and the direction of propagation of the incident wave $\hat{\mathbf{z}}$. The unit vectors $\hat{\mathbf{r}}$ and $\hat{\mathbf{z}}$ define the *scattering plane*, or the *plane of incidence*, which is determined by the angle ϕ, illustrated in Fig. 3.7. As arguments, the routine takes the scaling parameter X, the refraction index of the sphere n_2, and the number

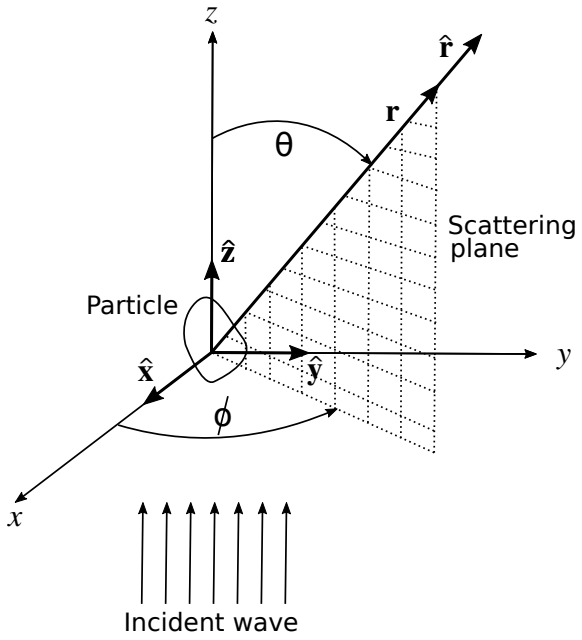

Figure 3.7: Scattering by an arbitrary particle. The incident wave is propagation in the positive $\hat{\mathbf{z}}$-direction, and the scattering plane is defined by $\hat{\mathbf{r}}$ and $\hat{\mathbf{z}}$.

of angles to evaluate the Mie scattering coefficients on. The dimensionless scaling parameter is defined as

$$X = 2\pi n_1 \frac{a}{\lambda}, \tag{3.15}$$

where λ is the wavelength of the incident wave, a is the radius of the sphere, and n_1 is the refraction index of the medium in which the sphere is embedded.

The scattering and extinction *cross sections*, C_{sca} and C_{ext}, may be computed from the efficiency factors as

$$C_{\text{sca}} = Q_{\text{sca}}\pi a^2, \qquad C_{\text{ext}} = Q_{\text{ext}}\pi a^2, \tag{3.16}$$

from which we may calculate the absorption cross section

$$C_{\text{abs}} = C_{\text{ext}} - C_{\text{sca}}. \tag{3.17}$$

The scattered and absorbed *power* is computed by multiplying their respective cross sections with the flux of the incident beam, i.e. $1/(2Z_0)|\mathbf{E}^{\text{inc}}|^2$ for a plane wave in vacuum. This way, we may compare the scattered and absorbed power from the Mie solution, with the scattered and absorbed power evaluated by the numerical simulation, W_{sca} and W_a.

In addition, it is enlightening to compare the angle dependent *bistatic scattering cross section*

$$\sigma_{\varphi}^{\text{sim}}(\theta) = 4\pi R^2 \frac{|\mathbf{E}^{\text{sca}}(\mathbf{r}_{\varphi}(\theta))|^2}{|\mathbf{E}^{\text{inc}}|^2}, \qquad \text{for } \varphi = \|, \perp, \tag{3.18}$$

to see how well the simulation results matches with the Mie solution in particular scattering planes and at particular scattering angles. Here $\mathbf{r}_{\varphi}(\theta)$ defines an circular

arc of radius R in a scattering plane parallel or perpendicular to the polarisation, corresponding to $\varphi = \parallel$ and $\varphi = \perp$, respectively. In other words, $\mathbf{r}_\parallel(\theta)$ is the spherical vector $\mathbf{r}(R, \theta, 0)$, and $\mathbf{r}_\perp(\theta)$ is the spherical vector $\mathbf{r}(R, \theta, \pi/2)$ for $\theta \in [0, \pi]$. Of course, R has a lower bound of a, the radius of the sphere. Using the scattering coefficients returned by the Mie-routine, `bhmie`, we may compute the scattered irradiance per incidence irradiance a scattering plane both parallel (plane of incidence), and perpendicular to the incident polarisation, defined as

$$i_\parallel = |S_2|^2, \tag{3.19}$$

and

$$i_\perp = |S_1|^2, \tag{3.20}$$

respectively. From these relationships, the comparable bistatic scattering cross sections of the Mie soluttion is

$$\sigma_\varphi^{\text{Mie}} = \frac{4\pi}{k_1^2} i_\varphi, \qquad \text{for } \varphi = \parallel, \perp, \tag{3.21}$$

where k_1 is the wavenumber of the surrounding medium.

3.3 Building the Simulation Program

By following the algorithm presented in Fig. 3.3, one may produce a program file, in Fortran, that contains instructions for simulating a specific scattering scenario. However, to be able to run the simulation one must create an executable from the programme file, which requires proper compilation of all source files, and proper linking of all object files and modules. Because of the many dependencies across the modules and the numerous source-files, it is very convenient to use the utility *make* [38] and construct a Makefile that accumulates the various dependencies and ensures a successful order of compilation. The tool *make* keeps track of which files that need recreation as well, avoiding compilation of files which the recent modifications do not affect. In a Makefile, rules for the compilation of source-files and the creation of object-files, module-files, and executables are defined. For example, in one of the Makefiles used in this project, it is explicitly stated that the object-files *working_precision_mod.o*, *maths_funcs_mod.o*, and *io_mod.o* must exist prior to creating *mesh_mod.o*. This is because `mesh_mod` depends on `working_precision_mod`, `math_funcs_mod`, and `io_mod`, and so the source file *mesh_mod.f90* will not compile without the corresponding module- and object-files present. Furthermore, general rules for creating object- and module-files (files with suffix *.o* and *.mod*, respectively) from source files (files with suffix *.f90*) are defined, which uses a user-specified compiler and compiler-flags.

In this project, a hierarchical Makefile system was used, consisting of a recursive parent Makefile in the top directory, and two additional Makefiles located in the subdirectories `src` and `testing`. The Makefiles assume the directory tree illustrated in Fig. 3.8.

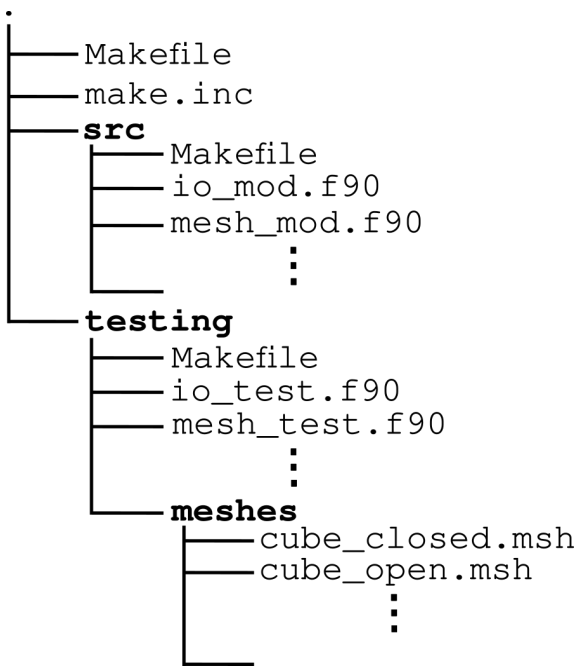

Figure 3.8: Directory tree/structure assumed by the recursive Makefile setup used in the project. The dot at the top represents the top directory. Bold font indicates directories and normal font indicates normal files. The lines and indentations illustrates the hierarchy of the files and directories.

The source directory, `src`, contains the source file to all modules presented in Section 3.1.2, in addition to a Makefile. The directory `testing` contains the source file of every testing program presented in Section 3.2, in addition to another Makefile and the directory `meshes`, which includes the Gmsh2 mesh-files used by the tests. Finally, we have the file `make.inc` which is imported by the Makefile in the top directory. This is where the user specifies the desired Fortran compiler, compilation and linking flags, the computer architecture, and the path to the required external libraries. The Linear Algebra PACKage (LAPACK) [39] and Basic Linear Algebra Subprograms (BLAS) [40] is the only required external library, from which routines for solving the matrix equation (2.99) is used. The libraries LAPACK and BLAS are both provided when using OpenBlas [37], as OpenBlas is a library that aims at optimising BLAS and LAPACK for specific processor types. In particular, the routine *GESV* is used to solve the equation using LU-factorisation in any precision (See Section 3.1.2). All Makefiles and `make.inc` are found in Appendix C.2.

When using the Makefile setup, the compilation and linking process is easy. After having updated `make.inc`, simply type the following command while in the top directory:

```
make
```

This will run the contents of the top Makefile, which recursively calls the Makefile in the directory `src`. The latter Makefile contains all the rules for compiling the source files, and will move the resulting module files into a new directory called `modules`.

The object-files will remain in `src`. If the compilation process is successful, the top Makefile will then call the Makefile in the directory `testing`. This Makefile contains the rules for compiling and linking the testing programs located in the same directory. The resulting executables are stored in a new directory called `bin`, and they are all run sequentially after creation. If a test fails, then the whole make process will fail, signalling that the built modules are not reliable for use in a simulation. The source files may be compiled without subsequently running tests by issuing the command

```
make lib
```

in the top directory, or `make` in the source directory `src`. After building all files the directory tree should look like the one in Fig. 3.9.

All modules-files, their corresponding object-files, and the folder `modules` are removed by running

```
make clean
```

in `src`. All test executables, their object-files and the folder `bin` are removed by calling the same command in the `testing` directory. Both cleaning procedures above will be executed by running `make clean` in the top directory.

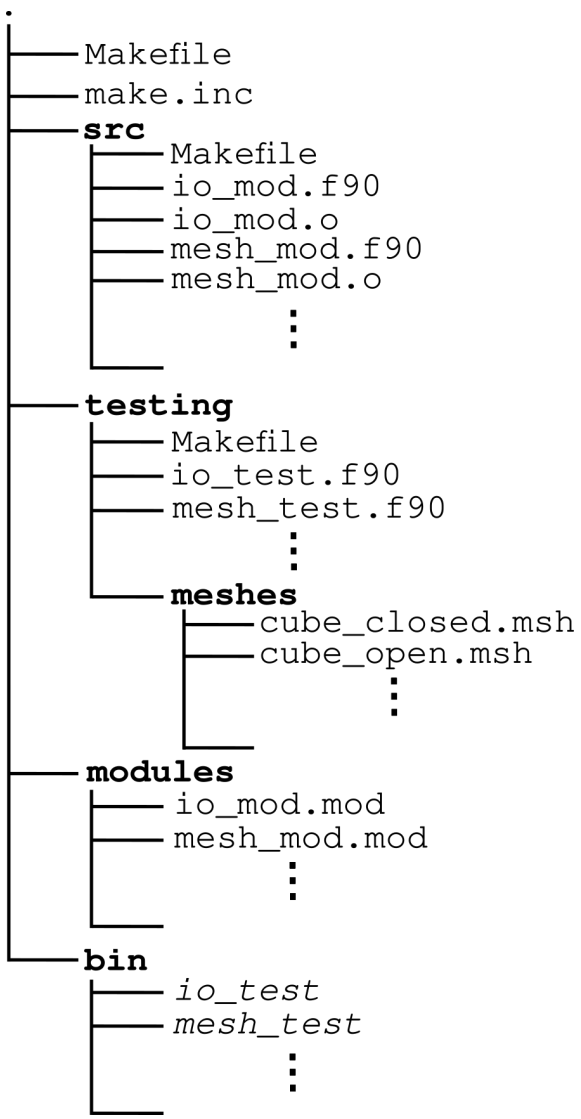

Figure 3.9: Directory tree/structure after building the modules and testing executables. The dot at the top represents the top directory. Bold font indicates directories, italic font indicates executables, and normal font indicates normal files. The lines and indentations illustrates the hierarchy of the files and directories.

Chapter 4

Results and Discussion

In this chapter we present and discuss the results from running the simulation program described in Chapter 3, that is, we look at the scattering of electromagnetic waves by arbitrary surfaces, numerically simulated by using the method of moments (MoM) and Galerkin's method for the weighted residuals. In all of the various scattering scenarios considered, the incident wave is planar, directed along the z-axis, and x-polarised. Additionally, both regions of the scattering geometry are considered nonmagnetic, i.e. the permeability equals the vacuum permeability, $\mu_i = \mu_0$. Furthermore, the surrounding medium is assumed to be vacuum, such that its refractive index and relative permitivity, n_1 and $\epsilon_{r,1}$, equals unity. The refractive index and relative permitivity of region 2 is from here on referred to as n and ϵ_r, respectively.

We begin by looking at the scattering of a dielectric sphere, whose permitivity is homogeneous and real, and assert the quality of the numerical simulation by comparing them the results obtained from the Mie solution (see Section 2.5), in addition to looking at symmetry properties, interference patterns and energy conservation.

Next, we look at a nonspherical scattering surface, the rectangular prism, and try a complex permitivity. The simulation program is then tested with multiple scattering surfaces, in particular a rectangular prism dipole. Using the permitivity of gold and the resonance wavelength found by Kern and Martin in Ref. [1], we demonstrate the ability to simulate localised surface plasmon resonance (LSPR).

Finally, results, computational efficiency, and memory use the face-by-face (FBF) and basis-by-basis (BBB) approach in calculating the matrix elements $\mathcal{D}_{mn}^{(i)}$ and $\mathcal{K}_{mn}^{(i)}$ (see Eq. (2.95)) are compared.

4.1 The Sphere

Interference, symmetry and energy conservation

Fig. 4.1 shows a contour plot of the electric field intensity $|\mathbf{E}|^2$ around and inside a dielectric sphere of refractive index $n = 4$, corresponding to a relative permitivity $\epsilon_r = 16$. The sphere is centred at origo, and its radius is $a = \lambda/2$, where λ is the wavelength of the incident, x-polarised, planar wave that is incident from the far left

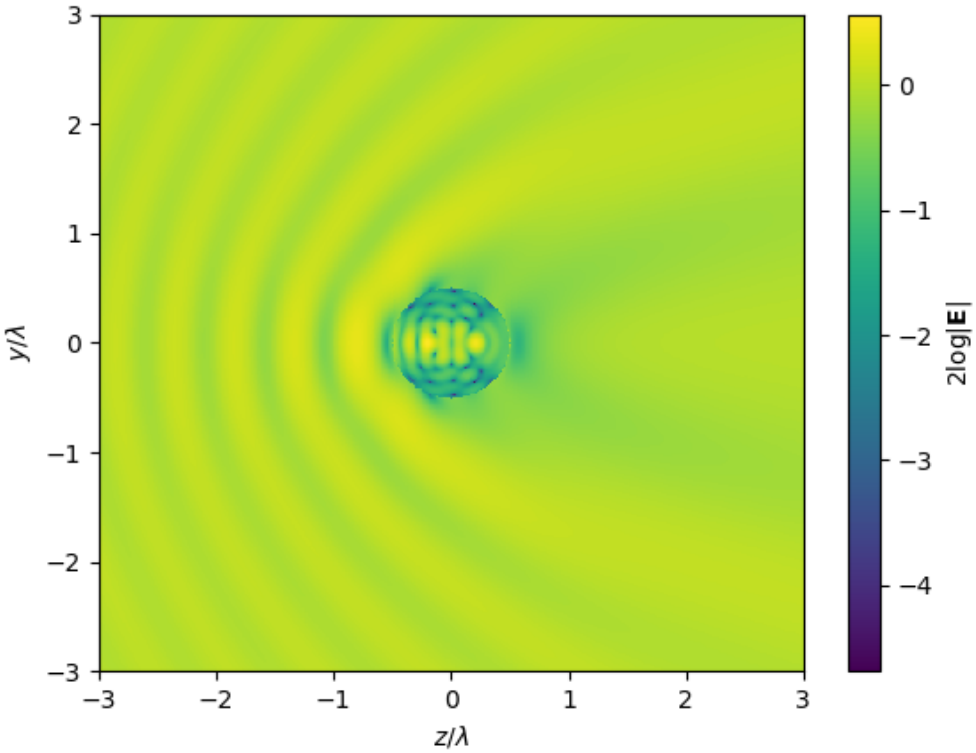

Figure 4.1: The the electric field intensity $|\mathbf{E}|^2$ in the yz-plane from scattering by a sphere with $n = 4$. The incident wave is x-polarised, has intensity $|\mathbf{E}^{\text{inc}}|^2 = 1$ and wavelength λ. The radius of the sphere is $\lambda/2$, and the DOF of the MoM is 10110.

($z = -\infty$). The amplitude of the incident wave is unity, such that $|\mathbf{E}^{\text{inc}}|^2 = 1$. In this specific simulation, the surface mesh contains 3370 faces, corresponding to 10110 degrees of freedom (DOF). The DOF is equal to the number of unknown coefficients α_n and β_n, i.e. DOF is twice the number of edges in the mesh, or equivalently, three times the number of faces. The relationship $ka = 2\pi a/\lambda$ is from here on referred to as the *scaling parameter*. The intensity in Fig. 4.1 is plotted in the yz-plane, i.e. a plane parallel to the propagation of the incident wave, but perpendicular to its polarisation. We clearly see an interference pattern in region 1 on both sides of $z = 0$, although the pattern from the forward scattering is less prominent and have a larger peak-to-peak distance. The interference pattern arising from the back scattering has an average peak-to-peak distance of $d_1^{\text{avg}} = 0.55 \pm 0.02\lambda$, which is slightly larger than the minimum peak-to-peak distance $d_1^{\text{min}} = 0.5\lambda$, expected from theory. Similar interference patterns are visible when plotting the electric field intensity in the xy-plane, as shown in Fig 4.2. Interference patterns with much shorter peak-to-peak distance are visible inside the scattering sphere (in region 2) as well. Measurement of a few peaks yields the distance $d_2^{\text{avg}} = 0.144 \pm 0.015\lambda$. Inside region 2, there are no incident wave present, but the transmitted wave scatters on the inside of the

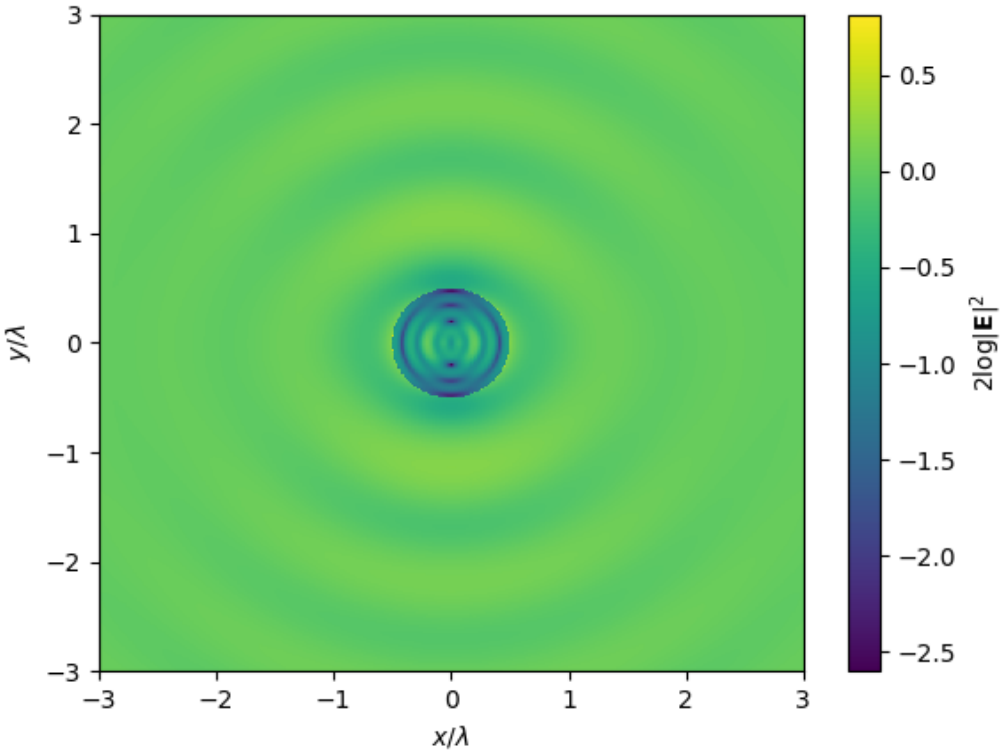

Figure 4.2: The the electric field intensity $|\mathbf{E}|^2$ in the xy-plane from scattering by a sphere with $n = 4$. The incident wave is x-polarised, has intensity $|\mathbf{E}^{\text{inc}}|^2 = 1$ and wavelength λ. The radius of the sphere is $\lambda/2$, and the DOF of the MoM is 10110.

boundary and interfere with itself. The wavenumber of the electromagnetic wave inside the sphere is $k_2 = \omega\sqrt{\epsilon_r\epsilon_0\mu_0} = 4k_1$, where ω is the angular frequency of the field in both regions, and $k_1 = 2\pi/\lambda$ is the wavenumber of the field in region 1. Thus, the expected minimum peak-to-peak distance of the interference pattern in region 2 is $d_2^{\min} = d_1^{\min}/4 = 0.125\lambda$, serving well as a lower bound of the observed pattern.

From looking at Figs. 4.1–4.2 it is apparent that the scattered electric field in the yz-plane, is symmetric across the xz-plane. Similarly, the scattered field in the xy-plane is symmetric across the z-axis. A proper test of the symmetric properties is presented in Fig 4.3, where the bistatic cross section (BSCS, see Eq. (3.18)) on the interval $\theta \in [0, \pi]$ is compared with the BSCS on the interval $\theta \in [\pi, 2\pi]$ for both scattering planes $\varphi = \|, \perp$. Here θ is the angle of scattering with respect to the positive z-axis (the incident direction of propagation). The figure shows that the symmetric properties are very well respected, which is a consequence of the spherical symmetries of the scatterer. Thus, this i a verification of the spherical symmetries the surface mesh representation should contain.

The energy for the particular scattering scenario shown in Figs. 4.1–4.3 was con-

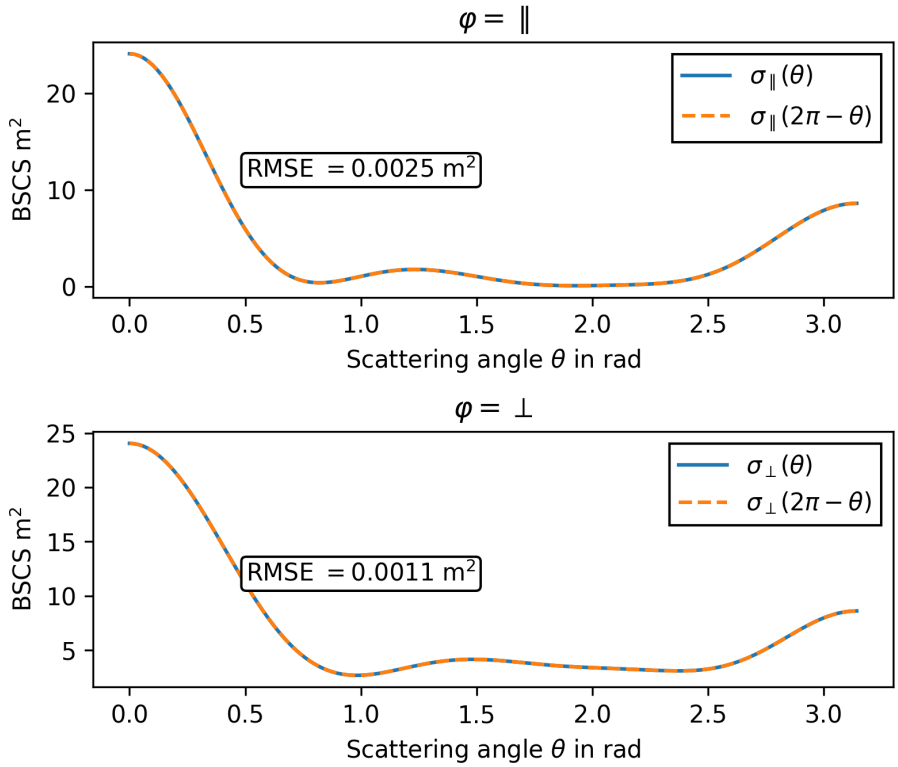

Figure 4.3: The bistatic scattering cross section of the sphere in Fig. 4.1. The BSCS across the symmetry planes are also plotted, along with the root mean square error (RMSE) between them.

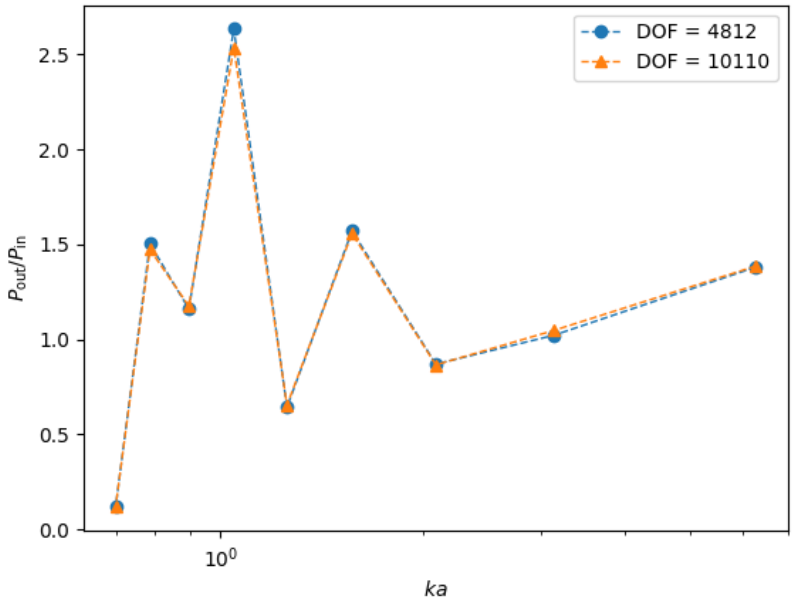

Figure 4.4: The ratio between incoming P_{in} and outgoing P_{out} average energy per unit time for scattering by a sphere of refractive index $n = 4$ and varying scaling parameter ka.

served with a precision of 4.7%. Thus, by simulating realistic interference patterns, keeping symmetry properties, and conserving energy, the numerical implementation seems to be working and giving fairly accurate results.

Fig. 4.4 shows how well energy is conserved in the scattering simulations when varying scaling parameter. The conservation of energy is evaluated by looking at the ratio

$$\frac{P_{\text{out}}}{P_{\text{in}}},$$

where P_{in} is the average energy per unit time flowing into the cross section of the scattering sphere, as in Eq. (3.13), and P_{out} is the average energy per unit time scattered out of an imaginary sphere of radius R, surrounding the sphere, as in Eq. (3.9). The energy conservation is unfortunately not very consistent when varying ka, which may indicate erroneous scattering amplitudes, caused by e.g. a scaling error in the implementation. The possibility of errors is discussed further in the next section.

Comparison with Mie theory

The BSCS shown in Fig. 4.3 is plotted with polar projection in Fig. 4.5, together with the BSCS evaluated by the Mie theory for a similar scattering scenario (i.e. sphere with $n = 4$ and $ka = \pi$). The Mie solution is calculated using the Fortran routine `bhmie` from Bohren and Huffman [8] (see Section 3.2.2). The BSCS in a

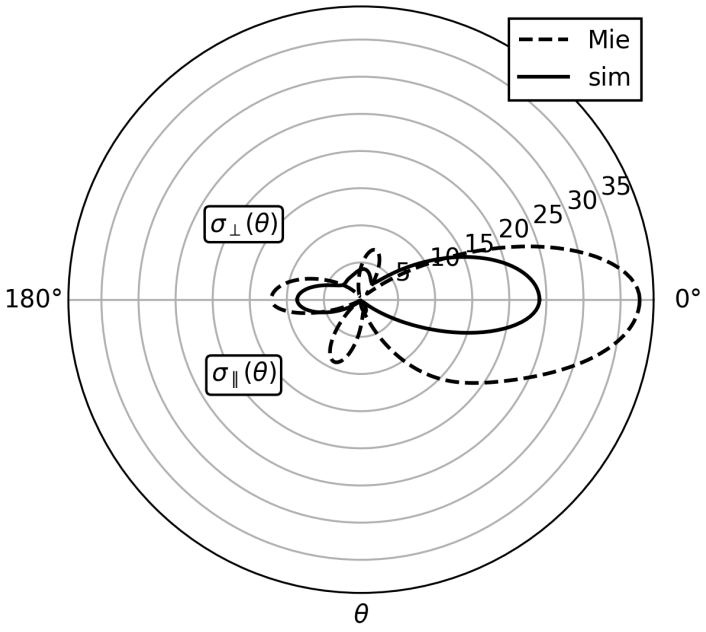

Figure 4.5: A polar plot of the bistatic scattering cross section of a sphere with refractive index $n = 4$. Results from both numerical simulation and the Mie solution.

scattering plane perpendicular to the polarisation of the incident plane wave and parallel to its propagation, i.e. $\sigma_\perp(\theta)$ for $\theta \in [0, \pi]$, is plotted in the upper half of the polar plot. In the lower half, the BSCS in a plane parallel to both the polarisation and propagation (the plane of incidence), i.e. $\sigma_\parallel(\theta)$ for $\theta \in [0, \pi]$, is plotted. For reference, the polar angle $\theta = 0$ is forward scattering.

Although the BSCS of the numerical simulation and the Mie solution are not perfectly equal, we may point out several similarities. The amplitudes are in general of the same order of magnitude, across all angles. Both have less backward scattering than forward, and the parallel and perpendicular scattering are of approximately the same magnitude for the same angle, except at a range around $\theta = \pi/2$, where both exhibit enhancement in amplitude.

Figure 4.6 compares the numerical simulation result to the Mie solution for a few different scaling parameters ka and refractive index n.

Figure 4.7 shows the integrated error

$$\Sigma_\varphi = \sqrt{\frac{1}{\pi}\int_0^\pi \mathrm{d}\theta \frac{\left[\sigma_\varphi^{\mathrm{sim}}(\theta) - \sigma_\varphi^{\mathrm{Mie}}(\theta)\right]^2}{\sigma_\varphi^{\mathrm{Mie}}(\theta)^2}}, \qquad \text{for } \varphi = \parallel, \perp, \tag{4.1}$$

and the total error

$$\Sigma = \Sigma_\parallel + \Sigma_\perp, \tag{4.2}$$

where the superscripts *sim* and *Mie* denote the BSCS from the numerical simulation and Mie solution, respectively, as a function of DOF. Compared to the equivalent figure in Kern and Martin [1], the errors arising from the numerical implementation

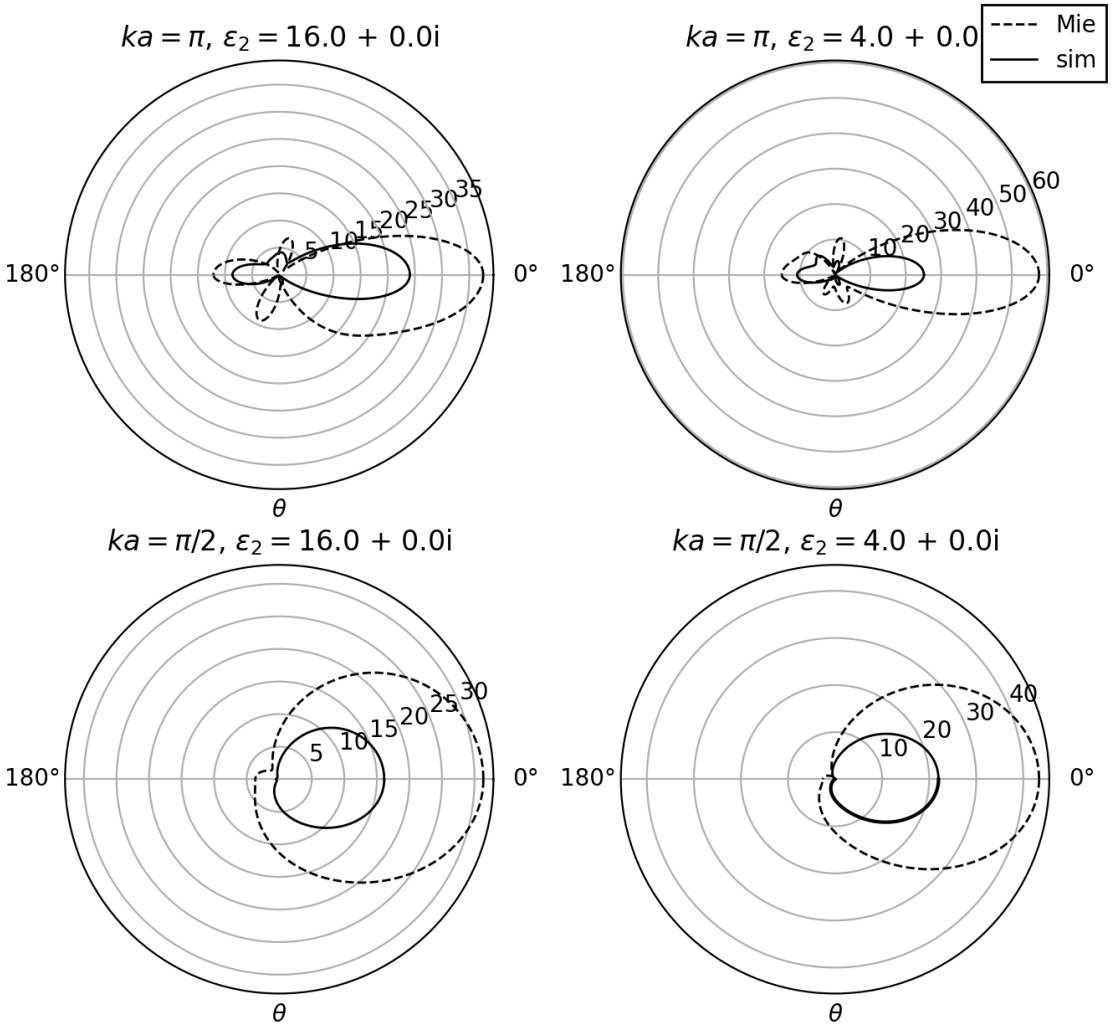

Figure 4.6: The BSCS of a sphere for a few different scaling parameters ka and permitivities ϵ_r. The upper half of the polar plots constitutes BSCS in a plane perpendicular to the polarisation and propagation of the incident wave, while the lower half constitutes BSCS in the plane of incidence.

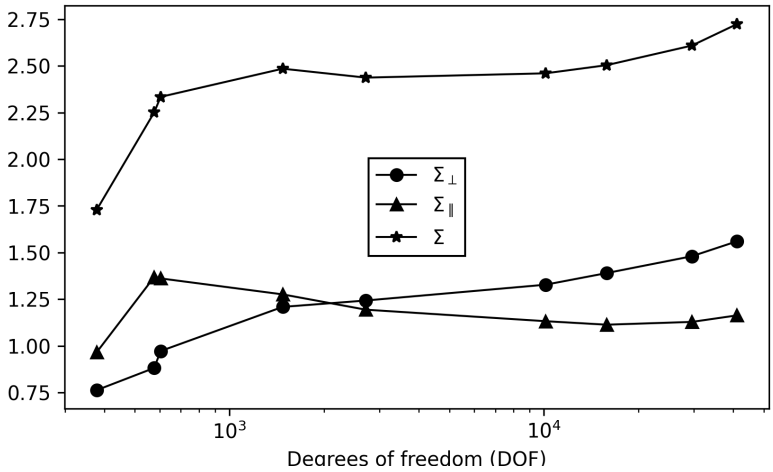

Figure 4.7: Integrated error (see Eq. (4.1)) of the numerical simulation compared to the Mie solution.

in this thesis are relatively large. In addition, the trend when increasing DOF is not falling, which indicates that the implementation may have a bug. For an increasing DOF, the surface mesh should be an increasingly better approximation of the perfect sphere assumed by the Mie theory. In addition, a higher DOF should result in a better MoM discretisation, i.e. a more accurate approximation of the equivalent currents defined by Eqs. (2.84) and (2.85). The reason for this behaviour is currently not known, but as touched upon above when discussing the conservation of energy, there might be a scaling error in the implementation, leading to erroneous scattering amplitudes. This makes sense since the BSCS result follows the trend of the Mie solution when varying θ, but has a consistently lesser amplitude. However, the scaling error cannot lie in the field distribution evaluation (Eq. (2.158)), which would scale the amplitude equally for all scattering angles. On the other hand, there could be a possible scaling error in the matrix elements of Eq. (2.100) or the elements of the vector $\mathbf{q}$ (defined by Eq. (2.101)), since this would alter the matrix equation (2.99) and affect the expansion coefficients α_n and β_n.

Alternatively, the large errors and instability could be caused by having too few quadrature points when evaluating the integrals using Gaussian quadrature, especially when the integrands are close to being singular. Convergence tests were conducted for some integrals, and the results seemed to converge when having 7 or more quadrature points within each triangle. The maximum number of quadrature points tested was 13, so that the results could have had a different convergence if a much larger range of quadrature points were tested. In the available literature on singularity treatment, it is common to test the methods with quadrature points up to several hundred [12, 15, 16]. However, the results from the numerical implementation in this thesis, showed little or no improvement when increasing the number of quadrature points from 3 to 13.

There is also the possibility of more trivial errors, such as a wrong sign or index

in any of the numerous calculations. Such an error could produce correct symmetric properties of the result, while still having a significant effect on the energy conservation. This type of error would most likely occur in the `PMCHW_RWG_mod`-module (see Section 3.1.2) because of two, main reasons; it is where the majority of the calculations are performed, including all of the integrals, and it comprises the numerical evaluations which are most difficult to test. The latter reason arises from the fact that there are, in most of the isolated calculations, no analytical solution to compare the numerical result to. The implementation of the singularity subtraction is particularly vulnerable to sign and index errors, as well as typos, because of its many transformations and minor calculations. Similar reasoning applies to the implementation of the FBF approach to the integral evaluations, although in this case we had the BBB implementation to compare the results to. The BBB approach is much more straight forward to implement, and therefore less prone to errors. Since the FBF and BBB approaches consistently gave equal results, we may with reasonably confidence conclude that the FBF approach implementation is, to certain extent, bug-free.

Furthermore, the current numerical implementation does not treat the system parameters dimensionless. Since the magnitudes of the permitivities and permeabilities remain constant while scaling the magnitude of λ and the mesh edges, there will be a risk of loss in precision. If the scaling parameter and dielectric properties of the media involved are kept constant and independent of wavelength, the results should be the same independent of the order of magnitude of λ and sphere radius a. Naturally, a test was conducted, showing that the results are approximately equal in the range $\lambda \in [2 \cdot 10^{-10}\text{m}, 2 \cdot 10^{3}\text{m}]$ for $ka = 1/2$. Consequently, the numerical implementation should be adapted to dimensionless parameters for simulating even larger or smaller systems. When the scale of the sphere is too small, the quantity $1/R$ will be less accurately computed, and the singularities of the integrals become more prominent.

Of course, we cannot ignore that the plots of the Mie solution may be wrong. They do however, stem from a very well established and reviewed implementation, and the BSCS plots are consistent with the one presented by Kern and Martin [1] with equal problem parameters, even though the amplitudes of the BSCS are not verified by this reference. Nevertheless, independent of the Mie solution, the results from the numerical simulation should conserve energy.

All the same, as it is the main motivation for implementing MoM with the surface integral equation (SIE) formulation, it is interesting to see how the numerical implementation tackles scattering of surface shapes other than the sphere. This issue we will now address.

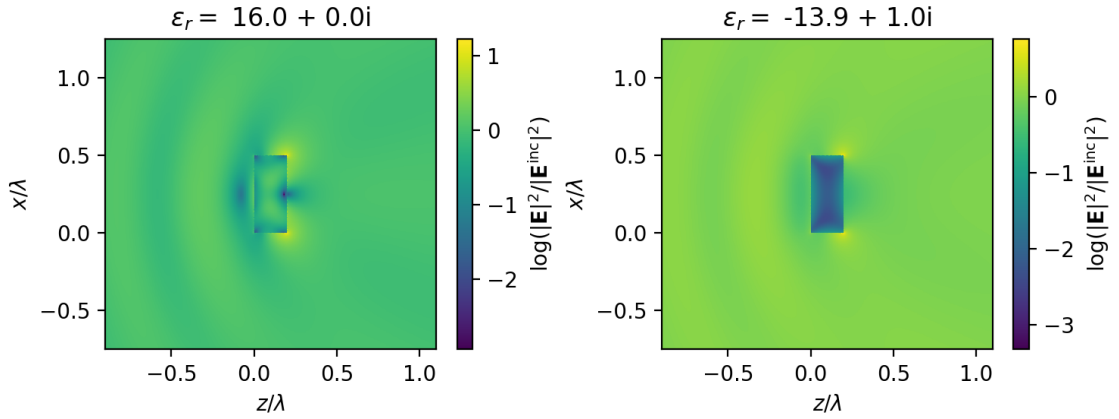

Figure 4.8: Left: The electric field intensity resulting from scattering by a nonabsorbing rectangular prism. Right: The electric field intensity resulting from scattering by an absorbing rectangular prism. Incident is directed along the positive z-axis and its intensity is $|\mathbf{E}^{\text{inc}}|^2 = 1$.

4.2 Nonspherical shapes

The rectangular prism

In this section, we exploit the advantages of using a numerical method applicable to scattering surfaces of arbitrary shape. Figure 4.8 shows the the electric field intensity around and inside of a rectangular prism with length $\lambda/2$ and a square cross section of side length $\lambda/5$. The surface mesh approximating the prism is illustrated in Fig. 4.9, having DOF = 5796. The rectangular prism in the left subplot of Fig. 4.8 is nonabsorbing with relative permitivity $\epsilon_r = 16$ ($n_2 = 4$), while the rectangular prism in the right subplot of Fig. 4.8 is absorbing with relative permitivity $\epsilon_r = -13.86 + 1.028i$. The scattered energy in the nonabsorbing case is far larger than the scattered energy in the absorbing case, which means a fraction of the energy is absorbed in the prism, as expected. It is also clearly visible that the absorbing prism acts like a sink because of the almost vanishing intensity.

As with the scattering from a sphere, there are visible interference patterns with peak-to-peak distance approximately equal to half the incident wavelength, or larger. In addition, the symmetric properties of the scattering surface are well respected.

The dipole antenna

The gold dipole antenna scenario described by Kern and Martin in Ref. [1] (section 3.B) is an excellent example for testing the numerical program's ability to simulate scattering from multiple scattering surfaces. It is also an appropriate test to whether the program is able to replicate LSPR effects. The rectangular prism in the section above is scaled to nm size, giving a length of 100 nm and sides of 40 nm, and is mirrored across the yz-plane, such that there is a gap of 30 nm between the square cross sections of the duplicates (see Fig. 4.10). The mesh is coarser at DOF=1527

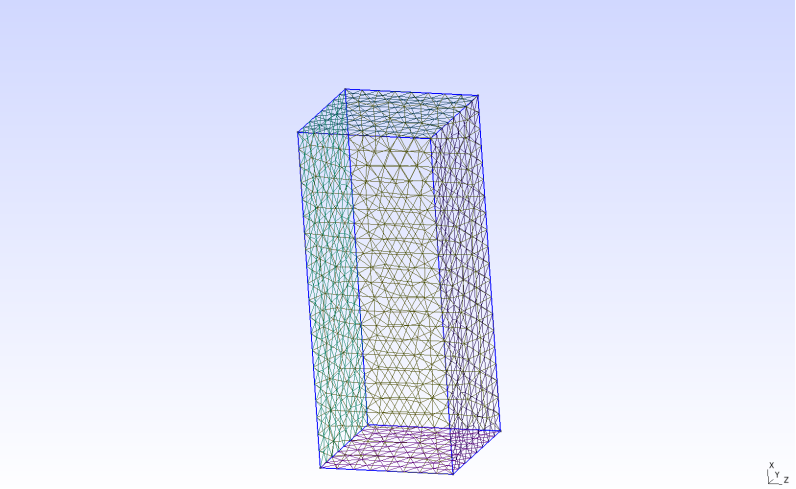

Figure 4.9: Mesh of rectangular prism generated by Gmsh [2]. DOF = 5796.

per prism. Figure 4.11 shows the electric field intensity $|\mathbf{E}|^2$ emerging when an incident plane wave of $\lambda = 662$ nm (the resonance wavelength found by Kern and Martin), meets the dipole with relative permitivity $\epsilon_r = -13.86 + 1.028i$. The permitivity, corresponding to a gold dipole at wavelength $\lambda = 662$ nm, is accessed from Ref. [41]. The contour plot clearly reveals a resonance in the gap between the prisms and at their corners. With, $|\mathbf{E}^{\text{inc}}|^2 = 1$ the field is magnified with a factor of approximately 250 in the gap, and 500 at the inner corners.

The fact that the simulation replicates the contour plot by Kern and Martin reasonable well, in spite of the different order of intensity, verifies the implementations ability to successfully represent multiple surface meshes.

4.2.1 Performance comparison off FBF and BBB approach

A considerable amount of work was put into implementing an FBF approach in evaluating the matrix elements of $\boldsymbol{\mathcal{H}}$ (Eq. (2.100)). As derived in Section 2.6.6, an FBF approach will reduce the amount of integral evaluations by a factor of 9, which will lead to considerable reduction of the overall simulation time. Using an FBF approach when evaluating the vector $\mathbf{q}$ (Eq. (2.101)) and when calculating the field distributions (Eq. (2.158)), will also reduce the number of integrals to evaluate by a factor of 9. However, the majority of the integrals, and thus the major computational effort, lies in evaluating $\boldsymbol{\mathcal{H}}$, whose size scales as $2N^2$, where N is the number of basis functions in the MoM.

On the other hand, the efficiency of the algorithm for solving the matrix equation (2.99) may scale faster than $2N^2$, such that for large enough N, the dominant computing effort lies in solving Eq. (2.99).

Nevertheless, Fig. 4.12 compares the time spent when computing the matrix $\boldsymbol{\mathcal{H}}$ by the use of the FBF and BBB approach, as a function of DOF. The computing times are relative to the shortest one among them, that is, relative to the time used by the FBF approach on a mesh of DOF = 378. From the figure, we see that the FBF

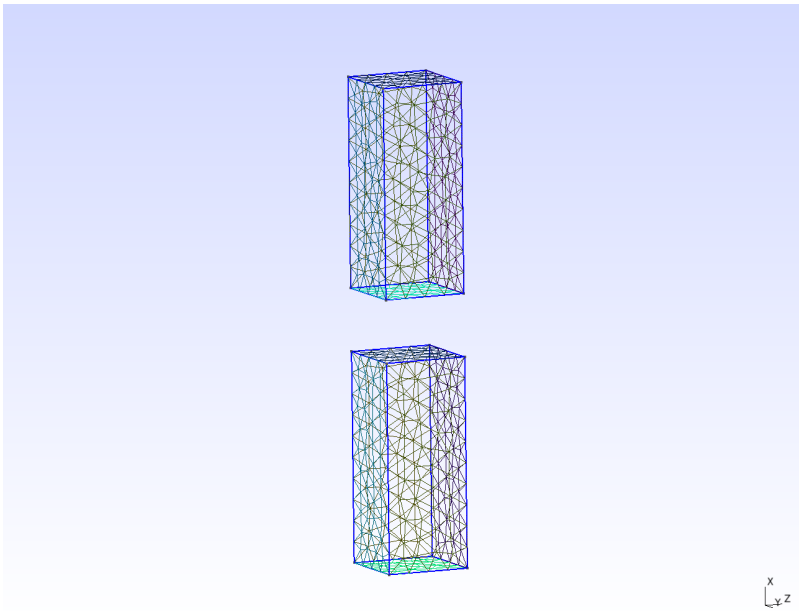

Figure 4.10: Mesh of dipole antenna generated by Gmsh [2]. There are 1527 DOF per prism.

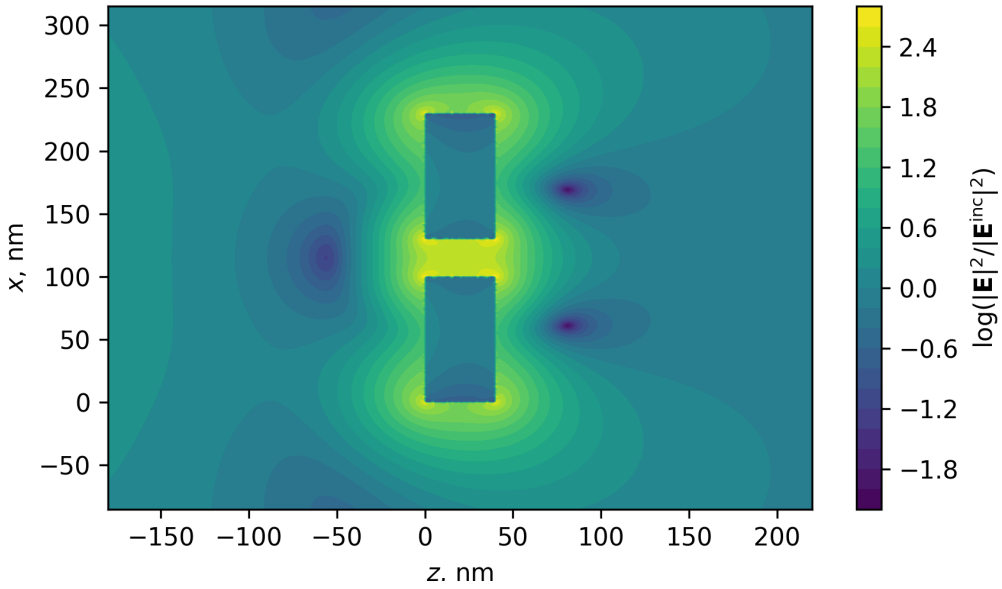

Figure 4.11: A gold dipole antenna showing LSPR. The incident resonance wavelength is $\lambda = 662$ nm. The figure is a reproduction of Fig. 6 in [1].

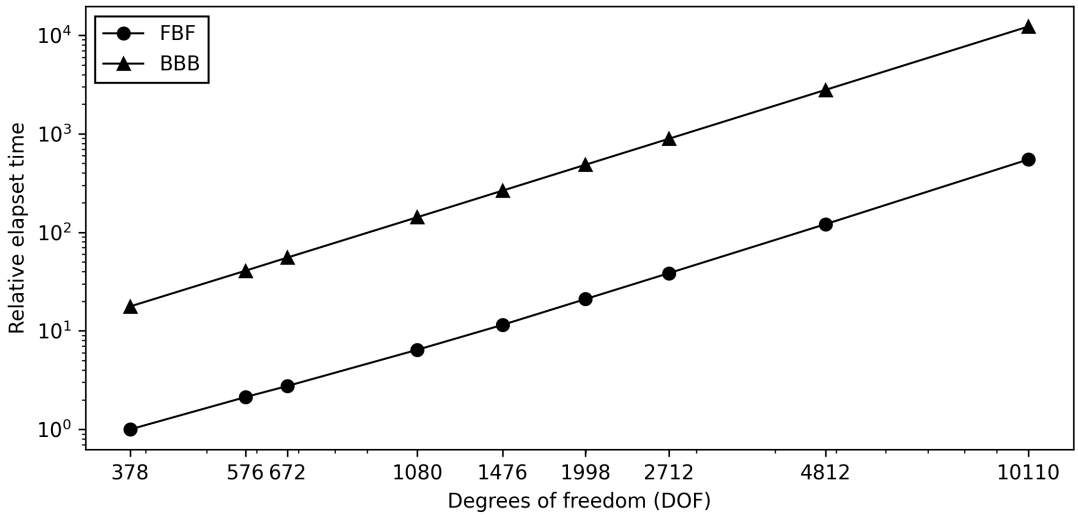

Figure 4.12: Computing time of the FBF and BBB approach as a function of DOF.

approach seems to be reducing the computing time by factor of about 20, which is a significant improvement. On the workstation used to produce this figure, 3.20GHz CPU and 32 GiB of RAM, the unitary computing time equalled approximately 1.19 seconds.

As mentioned in Section 2.6.6, the price we pay for increased computational efficiency is increased memory usage. Thus, to review the downside of an FBF approach, we include a plot of peak memory use as a function of DOF, shown in Fig. 4.13. Both approaches have exponentially increasing memery usage, reaching the order of gigabytes when crossing 10^3 DOF. The FBF approach uses considerable more memory than the BBB approach, although the BBB approach seems to converge towards the slope of the FBF approach.

In addition to the increased memory usage, the requirement of temporarily storing variables makes an FBF approach more difficult to parallelise on systems with distributed memory. The BBB approach is straight forward, where the calculation of the individual matrix elements may be more easily distributed.

Reducing memory usage

It is apparent from Fig. 4.13 that finite memory capacity may be a limiting factor when increasing the DOF in order to achieve desired accuracy. However, in the current code, there are several possible modifications that would decrease memory use. Although they will increase the computational effort, the major advantages of an FBF approach are kept. Among these modifications are:

1. Cease to temporary store the quadrature points of all faces, and instead calculate them progressively. In this case, all quadrature points will be calculated 9 times each.

2. Optimise the number of temporary stored complex variables per face-pair.

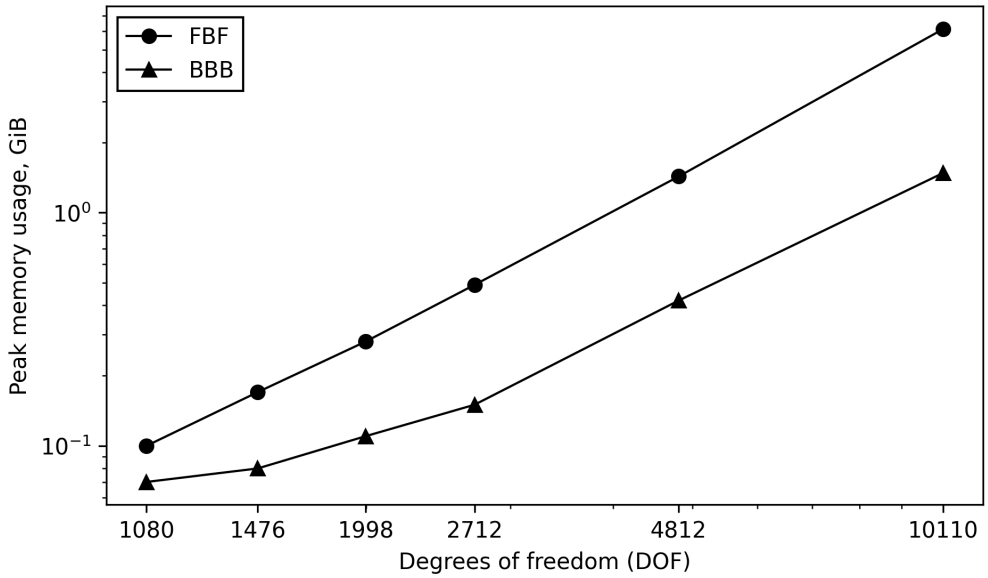

Figure 4.13: Peak memory use of the FBF and BBB approach as a function of DOF.

Reduces the memory usage with a factor proportional to $4/9N^2$.

3. Reduce the DOF of the surface mesh while increasing the number of quadrature points to maintain accuracy.

4. Avoid the temporary storing of the matrices $\boldsymbol{\mathcal{D}}^{(i)}$ and $\boldsymbol{\mathcal{K}}^{(i)}$.

Moreover, using an FBF approach only on the inner integrals of the matrix elements would decrease the memory use significantly, but also decrease the computational speed. As a result, this approach may be an attractive compromise.

If the memory use is too extensive even with the BBB approach, which is probable for large DOF, one may resort to reevaluation of the data structure representing the surface mesh and the RWG-basis functions.

Firstly, in the case where the edges of the mesh are linear, which is consistent with MoM using RWG basis functions, the array `vertices` in the derived type `mesh_mod_type` is superfluous because the number of nodes equals the number of vertices. Thus, we may avoid storing N_v instances of the derived type `vertex_type` by accessing the nodes directly as if they were the vertices.

Secondly, the derived type storing the RWG basis function representation, `RWG_basis_mod_type`, contains the array `basis_edges`, which stores N indices representing the edge of which the basis function is associated with. If the scattering surface is closed, which is consistent with the PMCHW-formulation, then `basis_edges` contain no new information, as all edges have an associated basis function, and the allocation of this array may be omitted.

Finally, `RWG_basis_mod_type` stores the adjacency relation $M_i^1\{M^2\}$ and the length of the basis edge L_n, which both may be calculated progressively. This increases the computing time, but reduces the memory usage by $2N$ integers and

N floating numbers. In contrast to the two suggestions above, this modification would increase the computational effort, and since the adjacency relation $M_i^1\{M^2\}$ and length L_n is used in all matrix elements, the increased computing time would be proportional to N^2. Since the gain in memory capacity is about the same as the two former suggestions, this modification should not be prioritised ahead of them.

However, the above modifications only reduces memory usage proportional to N, insignificant compared to improvements proportional to N^2, such as item 4. on the list above. This modification, along with item 1., would also affect the BBB approach.

Chapter 5

Conclusion and Outlook

In this thesis, we have used modular programming in Fortran to design and develop a framework for numerically simulating the two region electromagnetic scattering problem using the surface integral formulation (SIE). The framework uses an object-oriented approach to represent the discretised surface as a mesh through derived types. The main type uses a topology-based and hierarchical data structure in order to reduce memory usage while still enabling efficient fetching of adjacency relations. In addition, it includes functionality that, via an I/O module, imports pre-generated surface meshes in the *Gmsh2 ASCII* format. The modular and object-oriented approach makes it easy to extend the framework's compatibility to other file formats.

Furthermore, modules for solving the Maxwell equations with an SIE formulation of the Galerkin Method of Moments (MoM) was implemented. The triangulated surface was mapped onto the Rao-Wilton-Glisson (RWG) basis functions through inheritance of the derived type storing the triangulation. The electric field integral equation (EFIE) and the magnetic field integral equation (MFIE) were combined using the PMCHW-formulation, and the resulting matrix equation was stored in a derived type inheriting the RWG basis function mapping. Using singularity subtraction methods, the singular integrals of the matrix elements were separated (where relevant) into nonsingular, regular parts, which where numerically evaluated using Gaussian quadrature, and singular parts, which were analytically evaluated.

Results from the scattering from a sphere centred at the origo showed that the numerical implementation was able to reproduce the expected interference patterns, having a lower limit in the peak-to-peak distance at half of the incident wavelength. The scattered field was symmetric across the xz- and yz-planes, as expected from the spherical symmetry of the scatterer and the symmetric properties of the incident plane wave. The angular trend of the bistatic scattering cross section (BSCS) followed that of the Mie solution, and the amplitudes had equal order of magnitude. Be that as it may, the energy of the simulated system was not consistently conserved, and the BSCS did not converge towards a better approximation of the Mie solution as the degrees of freedom (DOF) of the simulation increased. These results indicate that the numerical implementation probably has one or more significant errors. Based on the complexity of the numerical and analytical integration, and on

the simulation results consistency with the expected symmetry, it is assumed that the error lies in the evaluation of the matrix elements, likely in the singularity subtraction, manifesting itself as a scaling error or a more trivial faulty sign or index. On the contrary, the error may also be caused by having too few quadrature points when numerically evaluating the regularised integrals.

Nevertheless, by simulating the scattering of a rectangular prism, the numerical implementation was found to successfully adapt to nonspherical scattering surfaces and absorbing regions. In addition, a reproduction of the gold dipole antenna resonance scattering scenario described by Kern and Martin in Ref. [1], verified the ability to represent and evaluate multiple scattering surfaces, as well as reproduce similar resonance patterns.

On the topic of optimising the computational efficiency of the code, the integrals of the matrix equation were evaluated face-by-face, instead of base-by-base. As a result, the number of integral evaluations was reduced by a factor of 9, as compared to a base-by-base implementation. The resulting computation time was reduced by a factor of about 20. On the other hand, the face-by-face approach requires significantly more memory usage, becoming the limiting factor for simulations of large DOF. Moreover, the face-by-face approach will prove more challenging to parallelise on systems of distributed memory, because the face-by-face integrals needs to be stored before calculating each matrix element.

Several modifications of the numerical implementation aiming to reduce the memory usage was suggested. For both the face-by-face and the basis-by-basis approach, it was concluded that the most effective measure was to omit storing the complex type matrices $\boldsymbol{\mathcal{D}}^{(i)}$ and $\boldsymbol{\mathcal{K}}^{(i)}$ (see Eq. (2.95)), which would reduce the memory usage by an amount equivalent to storing $4N^2$ variables of complex type, where N is the number of basis functions comprising the surface mesh. The most effective modification to reduce the memory use of the face-by-face approach even further, was the optimisation of the number of necessary additional variables.

There are various interesting extensions and improvements of the numerical implementation to pursue in the future. Although, given more time, the primary goal would have been to ensure energy conservation by further testing and search for the source of the error, e.g. by extensive convergence tests of the integral evaluations.

In addition to comparison with the Mie solution, it would be interesting to investigate the consistency of the simulation results with the simulation toolkits SCUFF-EM [42, 43] and COMSOL [44]. They would also provide a check of the Mie solution implemented in this thesis.

Furthermore, the memory usage of the implementation would have been optimised, such that applications to systems large DOF would be practical. This way, the numerical implementation would be suitable for scattering scenarios where the wavelength of the incident plane wave is much smaller than the scattering particle, as the spatially more frequent oscillations of the incident wave provides the requirement of a finer surface discretisation.

Ideally, the parameters of the implementation would have been dimensionless to maintain numerical precision. Hence, converting to a dimensionless implementation

is natural further work.

Likewise, the addition of adaptive quadrature will reduce the number of quadrature points in the integral evaluations to a minimum, and ensure the convergence of ill-behaved integrands. As follows, adaptive quadrature will improve the computational efficiency and numerical accuracy of the implementation. The computing efficiency could be further improved by the exploitation of multi-cored, modern computer systems through parallel programming. For shared-memory systems, it could be achieved through the straight forward use of *OpenMP* [45]. On distributed-memory systems, the task is naturally more complex due to the extensive memory use. However, it could be accomplished by careful design and implementation of e.g. *Message Passing Interface* (MPI) or a master/slave-model.

Although the numerical implementation in this thesis was found to have some issues leading to deviation from energy conservation, we have successfully designed and implemented a modular Fortran code, capable of simulating electromagnetic scattering by a single, or multiple, arbitrary shaped surfaces using MoM. We strongly believe that the shortcomings of the results presented are due to minor issues with our implementation, which the lack of time has made it difficult to locate, and not the design of the software or the methodology that it is based on.

Bibliography

[1] 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, Apr 2009.

[2] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[3] A. T. Young. Rayleigh scattering. *Physics Today*, 35(1):42–48, 1982.

[4] E. Petryayeva and U. J. Krull. Localized surface plasmon resonance: Nanostructures, bioassays and biosensing—a review. *Analytica Chimica Acta*, 706(1):8–24, 2011.

[5] P. Singh. *LSPR Biosensing: Recent Advances and Approaches*, pages 211–238. Springer International Publishing, Cham, 2017.

[6] M. Sui, S. Kunwar, P. Pandey, and J. Lee. Strongly confined localized surface plasmon resonance (LSPR) bands of Pt, AgPt, AgAuPt nanoparticles. *Scientific Reports*, 9(16582), Nov 2019.

[7] G. Mie. Beiträge zur optik trüber medien, speziell kolloidaler metallösungen. *Annalen der Physik*, 330(3):377–445, 1908.

[8] C. F. Bohren and D. R. Huffman. *Absorption and Scattering by a Sphere*, chapter 4, pages 82–129. John Wiley & Sons, Ltd, 1998.

[9] A. Bondeson, T. Rylander, and P. Ingelström. *Computational Electromagnetics*. Springer, New York, N.Y. London, 2005.

[10] B. T. Draine and P. J. Flatau. Discrete-dipole approximation for scattering calculations. *J. Opt. Soc. Am. A*, 11(4):1491–1499, Apr 1994.

[11] S. Rao, D. Wilton, and A. Glisson. Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Transactions on Antennas and Propagation*, 30(3):409–418, 1982.

[12] P. Yla-Oijala and M. Taskinen. Calculation of CFIE impedance matrix elements with RWG and $\mathbf{n} \times$ RWG functions. *IEEE Transactions on Antennas and Propagation*, 51(8):1837–1846, 2003.

[13] I. Hänninen, M. Taskinen, and J. Sarvas. Singularity Subtraction Integral Formulae for Surface Integral Equations with RWG, Rooftop and Hybrid Basis Functions. *Progress In Electromagnetics Research*, 63:243–278, 2006.

[14] R. D. Graglia. On the numerical integration of the linear shape functions times the 3-D Green's function or its gradient on a plane triangle. *IEEE Transactions on Antennas and Propagation*, 41(10):1448–1455, 1993.

[15] M. T. H. Reid, J. K. White, and S. G. Johnson. Generalized taylor–duffy method for efficient evaluation of galerkin integrals in boundary-element method computations. *IEEE Transactions on Antennas and Propagation*, 63(1):195–209, 2015.

[16] A. G. Polimeridis, J. M. Tamayo, J. M. Rius, and J. R. Mosig. Fast and accurate computation of hypersingular integrals in galerkin surface integral equation formulations via the direct evaluation method. *IEEE Transactions on Antennas and Propagation*, 59(6):2329–2340, jun 2011.

[17] Ismatullah and Eibert. Adaptive singularity cancellation for efficient treatment of near-singular and near-hypersingular integrals in surface integral equation formulations. *IEEE Transactions on Antennas and Propagation*, 56(1):274–278, 2008.

[18] M. G. Duffy. Quadrature Over a Pyramid or Cube of Integrands with a Singularity at a Vertex. *SIAM Journal on Numerical Analysis*, 19(6):1260–1262, 1982.

[19] A. G. Polimeridis and J. R. Mosig. Complete semi-analytical treatment of weakly singular integrals on planar triangles via the direct evaluation method. *International Journal for Numerical Methods in Engineering*, 83(12):1625–1650, 2010.

[20] A. G. Polimeridis, F. Vipiana, J. R. Mosig, and D. R. Wilton. Directfn: Fully numerical algorithms for high precision computation of singular integrals in galerkin sie methods. *IEEE Transactions on Antennas and Propagation*, 61(6):3112–3122, 2013.

[21] A. W. Glisson. *On the development of numerical techniques for treating arbitrarily-shaped surfaces*. PhD thesis, University of Mississippi, 1978.

[22] X. Q. Sheng, J. . Jin, J. Song, W. C. Chew, and C. . Lu. Solution of combined-field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies. *IEEE Transactions on Antennas and Propagation*, 46(11):1718–1726, 1998.

[23] C. Tai. *Dyadic Green Functions in Electromagnetic Theory*. IEEE Press Publication Series. IEEE Press, 1994.

[24] D. J. Griffiths. *Introduction to Electrodynamics*. Cambridge University Press, 4 edition, 2017.

[25] R. Harrington. *Field computation by moment methods*. IEEE Press, Piscataway, NJ, 1993.

[26] S. Rao, D. Wilton, and A. Glisson. Electromagnetic scattering by arbitrary surfaces. Rome Air Development Center, Griffiss AFB, NY, Tech. Rep. RADC-TR-79-325, March 1980.

[27] P. J. Frey and P. G. *Mesh generation : Application to finite elements*. ISTE John Wiley & Sons, London Hoboken, NJ, 2008.

[28] C. Bär. *Elementary differential geometry*. Cambridge University Press, Cambridge New York, 2010.

[29] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007.

[30] T. Sauer. *Numerical Analysis Second Edition*. Pearson Education Limited, 2014.

[31] P. C. Hammer, O. J. Marlowe, and A. H. Stroud. Numerical integration over simplexes and cones. *Mathematics of Computation*, 10:130–137, 1956.

[32] G. R. Cowper. Gaussian quadrature formulas for triangles. *International Journal for Numerical Methods in Engineering*, 7(3):405–408, 1973.

[33] J. Burkardt. FORTRAN90 Source Codes. Accessed November 2020.

[34] K. Rottmann. *Matematisk Formelsamling*. Spektrum forlag, 2014.

[35] J. Schöberl. NETGEN an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, July 1997.

[36] M. W. Beall and M. S. Shepard. A General Topology-Based Mesh Data Structure. *International Journal for Numerical Methods in Engineering*, 40(9):1573–1596, 1997.

[37] Z. Xianyi, W. Qian, and Z. Yunquan. Model-driven Level 3 BLAS Performance Optimization on Loongson 3A Processor. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 684–691, 2012.

[38] GNU Software. *GNU make Manual*.

[39] 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*, third edition, August 1999.

[40] L. S. Blackford, A.e Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[41] P. B. Johnson and R. W. Christy. Optical constants of the noble metals. *Phys. Rev. B*, 6:4370–4379, Dec 1972.

[42] M. T. H. Reid and S. G. Johnson. Efficient Computation of Power, Force, and Torque in BEM Scattering Calculations. *ArXiv e-prints*, July 2013.

[43] `http://github.com/homerreid/scuff-EM`.

[44] S. Yushanov, J. S. Crompton, and K. C. Koppenhoefer. Mie scattering of electromagnetic waves. AltaSim Technologies, 2013.

[45] D. Clark. Openmp: a parallel standard for the masses. *IEEE Concurrency*, 6(1):10–12, 1998.

Appendix A

Expansion of $\mathcal{K}_{mn}^{(i)}$

As discussed in Section 2.6.6, the double integral in $\mathcal{K}_{mn}^{(i)}$ has the form

$$\frac{1}{A_m^p A_n^q} \int_{T^p} \mathrm{d}S(\mathbf{r} - \mathbf{p}_m^p) \cdot \int_{T^p} \mathrm{d}S'[\nabla' G_i(\mathbf{r}, \mathbf{r}')] \times (\mathbf{r}' - \mathbf{p}_n^q), \tag{A.1}$$

which may be approximated by the double sum

$$\begin{aligned} \sum_{j=1}^{N_j} w_j(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p - \mathbf{p}_m^p) \cdot \Big\{ & \mathbb{I}_\xi^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_1'^q + \mathbf{r}_1'^q \times \mathbf{p}_n^q \right] \\ & + \mathbb{I}_\eta^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_2'^q + \mathbf{r}_2'^q \times \mathbf{p}_n^q \right] \\ & + \mathbb{I}_\zeta^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{r}_3'^q + \mathbf{r}_3'^q \times \mathbf{p}_n^q \right] \\ & - \mathbb{I}^{\mathbf{r}q} \left[(\alpha_j \mathbf{r}_1^p + \beta_j \mathbf{r}_2^p + \gamma_j \mathbf{r}_3^p) \times \mathbf{p}_n^q \right] \Big\}. \end{aligned} \tag{A.2}$$

To enable a face-by-face approach in evaluating the double integrals of $\mathcal{K}_{mn}^{(i)}$, the sums in Eq. (A.2) must be independent of the free vertices $\mathbf{p}_m^p$ and $\mathbf{p}_n^q$. Further expansion of Eq. (A.2) yields

$$
\begin{aligned}
&(\mathbf{r}_1'^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\alpha\xi}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\xi}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\xi}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\xi}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_2'^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\alpha\eta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\eta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\eta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\eta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_3'^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\alpha\zeta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\zeta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\zeta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\zeta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_1^q \times \mathbf{r}_1'^q) \cdot \left[\mathbb{Q}_{\alpha\alpha\xi}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\alpha\xi}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\alpha\xi}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\alpha\xi}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_2^q \times \mathbf{r}_1'^q) \cdot \left[\mathbb{Q}_{\alpha\beta\xi}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\beta\xi}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\beta\xi}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\beta\xi}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_3^q \times \mathbf{r}_1'^q) \cdot \left[\mathbb{Q}_{\alpha\gamma\xi}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\gamma\xi}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\gamma\xi}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\gamma\xi}^{pq}\mathbf{p}_m^p\right] \\
-&(\mathbf{r}_1^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\alpha\alpha}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\alpha\beta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\alpha\gamma}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\alpha}^{pq}\mathbf{p}_m^p\right] \\
-&(\mathbf{r}_2^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\beta\alpha}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\beta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\beta\gamma}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\beta}^{pq}\mathbf{p}_m^p\right] \\
-&(\mathbf{r}_3^q \times \mathbf{p}_n^q) \cdot \left[\mathbb{Q}_{\gamma\alpha}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\gamma\beta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\gamma}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\gamma}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_1^q \times \mathbf{r}_2'^q) \cdot \left[\mathbb{Q}_{\alpha\alpha\eta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\alpha\eta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\alpha\eta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\alpha\eta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_2^q \times \mathbf{r}_2'^q) \cdot \left[\mathbb{Q}_{\alpha\beta\eta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\beta\eta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\beta\eta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\beta\eta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_3^q \times \mathbf{r}_2'^q) \cdot \left[\mathbb{Q}_{\alpha\gamma\eta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\gamma\eta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\gamma\eta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\gamma\eta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_2^q \times \mathbf{r}_3'^q) \cdot \left[\mathbb{Q}_{\alpha\alpha\zeta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\alpha\zeta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\alpha\zeta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\alpha\zeta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_2^q \times \mathbf{r}_3'^q) \cdot \left[\mathbb{Q}_{\alpha\beta\zeta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\beta\zeta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\beta\zeta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\beta\zeta}^{pq}\mathbf{p}_m^p\right] \\
+&(\mathbf{r}_3^q \times \mathbf{r}_3'^q) \cdot \left[\mathbb{Q}_{\alpha\gamma\zeta}^{pq}\mathbf{r}_1^p + \mathbb{Q}_{\beta\gamma\zeta}^{pq}\mathbf{r}_2^p + \mathbb{Q}_{\gamma\gamma\zeta}^{pq}\mathbf{r}_3^p - \mathbb{Q}_{\gamma\zeta}^{pq}\mathbf{p}_m^p\right],
\end{aligned}
\tag{A.3}
$$

where

$$\mathbb{Q}_{\alpha\xi}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \mathbb{I}_{\xi}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\alpha\xi}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k^2 \mathbb{I}_{\xi}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\beta\xi}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k \mathbb{I}_{\xi}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\alpha\eta}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k^2 \mathbb{I}_{\eta}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\xi}^{pq} = \sum_{k=1}^{N_k} w_k \mathbb{I}_{\xi}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\alpha}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k^2 \mathbb{I}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\alpha\eta}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \mathbb{I}_{\eta}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\beta\beta\xi}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k^2 \mathbb{I}_{\eta}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\beta\eta}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k \mathbb{I}_{\eta}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\beta\beta\eta}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k^2 \mathbb{I}_{\eta}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\eta}^{pq} = \sum_{k=1}^{N_k} w_k \mathbb{I}_{\eta}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\beta\beta}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k^2 \mathbb{I}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\alpha}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \mathbb{I}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\beta\xi}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \beta_k \mathbb{I}_{\xi}^{\mathbf{r}q},$$

$$\mathbb{Q}_{\beta}^{pq} = \sum_{k=1}^{N_k} w_k \beta_k \mathbb{I}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\beta\eta}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \beta_k \mathbb{I}_{\eta}^{\mathbf{r}q},$$

$$\mathbb{Q}^{pq} = \sum_{k=1}^{N_k} w_k \mathbb{I}^{\mathbf{r}q}, \qquad \mathbb{Q}_{\alpha\beta}^{pq} = \sum_{k=1}^{N_k} w_k \alpha_k \beta_k \mathbb{I}^{\mathbf{r}q}, \tag{A.4}$$

and

$$
\begin{aligned}
\mathbb{Q}^{pq}_{\alpha\zeta} &= \mathbb{Q}^{pq}_{\alpha} - \mathbb{Q}^{pq}_{\alpha\xi} - \mathbb{Q}^{pq}_{\alpha\eta},\\
\mathbb{Q}^{pq}_{\beta\zeta} &= \mathbb{Q}^{pq}_{\beta} - \mathbb{Q}^{pq}_{\beta\xi} - \mathbb{Q}^{pq}_{\beta\eta},\\
\mathbb{Q}^{pq}_{\zeta} &= \mathbb{Q}^{pq} - \mathbb{Q}^{pq}_{\xi} - \mathbb{Q}^{pq}_{\eta},\\
\mathbb{Q}^{pq}_{\gamma} &= \mathbb{Q}^{pq}_{\zeta} - \mathbb{Q}^{pq}_{\alpha} - \mathbb{Q}^{pq}_{\beta},\\
\mathbb{Q}^{pq}_{\gamma\xi} &= \mathbb{Q}^{pq}_{\xi} - \mathbb{Q}^{pq}_{\alpha\xi} - \mathbb{Q}^{pq}_{\beta\xi},\\
\mathbb{Q}^{pq}_{\gamma\eta} &= \mathbb{Q}^{pq}_{\eta} - \mathbb{Q}^{pq}_{\alpha\eta} - \mathbb{Q}^{pq}_{\beta\eta},\\
\mathbb{Q}^{pq}_{\gamma\zeta} &= \mathbb{Q}^{pq}_{\zeta} - \mathbb{Q}^{pq}_{\alpha\zeta} - \mathbb{Q}^{pq}_{\beta\zeta},
\end{aligned}
\qquad
\begin{aligned}
\mathbb{Q}^{pq}_{\alpha\alpha\zeta} &= \mathbb{Q}^{pq}_{\alpha\alpha} - \mathbb{Q}^{pq}_{\alpha\alpha\xi} - \mathbb{Q}^{pq}_{\alpha\alpha\eta},\\
\mathbb{Q}^{pq}_{\beta\beta\zeta} &= \mathbb{Q}^{pq}_{\beta\beta} - \mathbb{Q}^{pq}_{\beta\beta\xi} - \mathbb{Q}^{pq}_{\beta\beta\eta},\\
\mathbb{Q}^{pq}_{\alpha\beta\zeta} &= \mathbb{Q}^{pq}_{\alpha\beta} - \mathbb{Q}^{pq}_{\alpha\beta\xi} - \mathbb{Q}^{pq}_{\alpha\beta\eta},\\
\mathbb{Q}^{pq}_{\gamma} &= \mathbb{Q}^{pq}_{\zeta} - \mathbb{Q}^{pq}_{\alpha} - \mathbb{Q}^{pq}_{\beta},\\
\mathbb{Q}^{pq}_{\alpha\gamma\xi} &= \mathbb{Q}^{pq}_{\alpha\xi} - \mathbb{Q}^{pq}_{\alpha\alpha\xi} - \mathbb{Q}^{pq}_{\alpha\beta\xi},\\
\mathbb{Q}^{pq}_{\alpha\gamma\eta} &= \mathbb{Q}^{pq}_{\alpha\eta} - \mathbb{Q}^{pq}_{\alpha\alpha\eta} - \mathbb{Q}^{pq}_{\alpha\beta\eta},\\
\mathbb{Q}^{pq}_{\alpha\gamma\zeta} &= \mathbb{Q}^{pq}_{\alpha\gamma} - \mathbb{Q}^{pq}_{\alpha\gamma\xi} - \mathbb{Q}^{pq}_{\alpha\gamma\eta},\\
\mathbb{Q}^{pq}_{\alpha\gamma} &= \mathbb{Q}^{pq}_{\alpha} - \mathbb{Q}^{pq}_{\alpha\alpha} - \mathbb{Q}^{pq}_{\alpha\beta},\\
\mathbb{Q}^{pq}_{\beta\gamma} &= \mathbb{Q}^{pq}_{\beta} - \mathbb{Q}^{pq}_{\alpha\beta} - \mathbb{Q}^{pq}_{\beta\beta},\\
\mathbb{Q}^{pq}_{\gamma\gamma\xi} &= \mathbb{Q}^{pq}_{\gamma\xi} - \mathbb{Q}^{pq}_{\alpha\gamma\xi} - \mathbb{Q}^{pq}_{\beta\gamma\xi},\\
\mathbb{Q}^{pq}_{\gamma\gamma\eta} &= \mathbb{Q}^{pq}_{\gamma\eta} - \mathbb{Q}^{pq}_{\alpha\gamma\eta} - \mathbb{Q}^{pq}_{\beta\gamma\eta},\\
\mathbb{Q}^{pq}_{\gamma\gamma} &= \mathbb{Q}^{pq}_{\gamma} - \mathbb{Q}^{pq}_{\alpha\gamma} - \mathbb{Q}^{pq}_{\beta\gamma},\\
\mathbb{Q}^{pq}_{\gamma\gamma\zeta} &= \mathbb{Q}^{pq}_{\gamma\gamma} - \mathbb{Q}^{pq}_{\gamma\gamma\xi} - \mathbb{Q}^{pq}_{\gamma\gamma\eta},\\
\mathbb{Q}^{pq}_{\beta\gamma\xi} &= \mathbb{Q}^{pq}_{\beta\xi} - \mathbb{Q}^{pq}_{\alpha\beta\xi} - \mathbb{Q}^{pq}_{\beta\beta\xi},\\
\mathbb{Q}^{pq}_{\beta\gamma\eta} &= \mathbb{Q}^{pq}_{\beta\eta} - \mathbb{Q}^{pq}_{\alpha\beta\eta} - \mathbb{Q}^{pq}_{\beta\beta\eta},\\
\mathbb{Q}^{pq}_{\beta\gamma\zeta} &= \mathbb{Q}^{pq}_{\beta\gamma} - \mathbb{Q}^{pq}_{\beta\gamma\xi} - \mathbb{Q}^{pq}_{\beta\gamma\eta},
\end{aligned}
\tag{A.5}
$$

as consequence of the constraints $\gamma = 1 - \alpha - \beta$ and $\zeta = 1 - \xi - \eta$.

Appendix B

Module Interfaces

In this appendix we list the Fortran code defining the interfaces of selected modules that are discussed in Section 3.1.2. The reader is referred to the code attached to the report for the Fortran modules in their entirety.

B.1 `mesh_mod`

The listing below contains the interface of the module `mesh_mod`. The complete module is defined in the Fortran file *mesh_mod.f90*.

```
   ,
1  module mesh_mod
2  !!=================================================================
3  ! This module represents a mesh in the hierarchical structure:
4  !    face --> edge --> vertex --> point
5  !
6  ! A mesh_type is initialised by reading a .msh-formated file.
7  !
8  ! Abbreviations:
9  !  CS - Closed Surface
10 !  OS - Open Surface
11 !‘
12 ! Last edited: March 7th 2021.
13 !!=================================================================
14
15   !!==============!!
16   ! Use statements !
17   !================!================================================
18   use working_precision, only: wp
19   use math_funcs_mod   , only: cross_prod_3D
20   use io_mod           , only: open_read_gmsh2
21   use io_mod           , only: r8mat_write
22   use iso_fortran_env  , only: real64
23   use is_close_mod     , only: is_close
24   use constants_mod    , only: PI
25   use constants_mod    , only: ZERO
26
27   implicit none
```

```
28
29    !!==================!!
30    ! External procedures !
31    !====================!=========================================
32    external :: dnrm2 ! BLAS level 1: Euclidean norm (double)
33
34
35    !!==================================!!
36    ! Public types/procedures/constants !
37    !==================================!============================
38    public :: mesh_type  ! Main type
39    public :: face_type
40    public :: edge_type
41    public :: vertex_type
42    public :: node_type
43
44    !!==================================!!
45    ! Private types/procedures/constants !
46    !==================================!============================
47    private :: eval_Euler_characteristic_CS
48    private :: check_input_triangulated_surface
49    private :: eval_topology_on_triangulated_surface
50
51    ! From here on everything is by defualt decleared private
52    private
53
54    !!------------------------!!
55    ! Derived type definitions !
56    !------------------------!-------------------------------------
57    type face_type
58       ! Type to store indices related to the edges
59       !forming a face on the mesh.
60       integer, dimension(:), allocatable :: edges
61    end type face_type
62
63    type edge_type
64       ! Type to store the indicies related to the vertices
65       ! forming an edge.
66       integer, dimension(2)              :: vertices
67       ! Length of node_idx depends on edge order:
68       !  linear:    len = 0
69       !  quadratic: len = 1
70       !  cubic:     len = 2
71       integer, dimension(:), allocatable :: node_idx
72     contains
73       procedure, pass(this), public :: initialise_edge
74    end type edge_type
75
76    type vertex_type
77       ! Type to store the index of the node at which
78       ! the vertex is located.
79       integer :: node_idx
80    end type vertex_type
81
```

```
82    type node_type
83       ! Type to store the 3D coordinates of a node.
84       real(wp), dimension(3) :: coords
85    end type node_type
86
87    !!---------!!
88    ! Main type !
89    !-----------!-----------------------------------------------------
90    type mesh_type
91       type (face_type)  , dimension(:), allocatable :: faces
92       type (edge_type)  , dimension(:), allocatable :: edges
93       type (vertex_type), dimension(:), &
94            allocatable :: vertices
95       type (node_type)  , dimension(:), allocatable :: nodes
96       integer            :: edge_order
97       integer            :: spatial_dim
98       integer            :: face_order
99       integer            :: num_faces
100      integer            :: num_edges
101      integer            :: num_vertices
102      integer            :: num_nodes
103      integer            :: num_handles
104      integer            :: num_apertures
105      integer            :: num_boundary_edges
106      logical            :: closed_surface
107    contains
108      ! Initialisers
109      procedure, pass(this), public  :: initialise
110      procedure, pass(this), public :: initialise_tetrahedron
111      ! Writing procedures
112      procedure, pass(this), public  :: write_mesh
113      ! Deallocation of attributes
114      procedure, pass(this), public  :: deallocate_attributes
115      ! Get-procedures
116      procedure, pass(this), public  :: get_closed_surface
117      procedure, pass(this), public  :: get_edge_order
118      procedure, pass(this), public  :: get_spatial_dim
119      procedure, pass(this), public  :: get_face_order
120      procedure, pass(this), public  :: get_num_handles
121      procedure, pass(this), public  :: get_num_faces
122      procedure, pass(this), public  :: get_num_edges
123      procedure, pass(this), public  :: get_num_vertices
124      procedure, pass(this), public  :: get_num_nodes
125      procedure, pass(this), public  :: get_topology
126      procedure, pass(this), public  :: get_edges_on_face
127      procedure, pass(this), public  :: get_vertices_of_edge
128      procedure, pass(this), public  :: get_vertices_of_face
129      procedure, pass(this), public  :: get_vertex_coords
130      procedure, pass(this), public  :: get_edge_coords
131      procedure, pass(this), public  :: get_face_coords
132      ! Calculations
133      procedure, pass(this), public  :: face_normal
134      procedure, pass(this), public  :: face_unit_normal
135      procedure, pass(this), public  :: face_area
```

```
136       procedure, pass(this), public  :: face_centroid
137       procedure, pass(this), public  :: edge_length
138       procedure, pass(this), public  :: surface_area
139       procedure, pass(this), public  :: volume
140       ! Other routines
141       procedure, pass(this), public  :: print
142       procedure, pass(this), public  :: scale_nodes
143       ! For determining whether a point is inside or outside
144       ! of meshe_nodes
145       procedure, pass(this), public  :: solid_angle_spanned_by_face
146       procedure, pass(this), public  :: solid_angle_spanned_by_mesh
147       procedure, pass(this), public  :: is_obs_pnt_inside_mesh
148       ! Private procedures for internal use
149       procedure, pass(this), private :: create_member_types_linear
150
151    end type mesh_type
152
153
154    !!================================!!
155    ! Overloaded operator interfaces !
156    !================================!================================
157    ! Overloaded operator interfaces
158    interface operator (==)
159       module procedure is_edges_equal
160    end interface operator (==)
```

B.2 RWG_basis_mod

The listing below contains the interface of the module `RWG_basis_mod`. The complete module is defined in the Fortran file *RWG_basis_mod.f90*.

```
    ,
1   module RWG_basis_mod
2   !!====================================================================
3   ! This module defines the RWG basis type, which inherits the
4   ! mesh_mod_type
5   ! from mesh_mod.f90 and represents an RWG basis function mapping of
6   ! a surface mesh.
7   !
8   ! Abbreviations:
9   !  CS - Closed Surface
10  !  OS - Open Surface
11  !  GQ - Gaussian Quadrature
12  !
13  ! Last edited: March 7th 2021.
14  !!====================================================================
15
16    !!==============!!
17    ! Use statements !
18    !================!================================================
19    use working_precision, only: wp
20    use iso_fortran_env  , only: real64
```

```
21    use mesh_mod          , only: mesh_type
22    use math_funcs_mod    , only: cross_prod_3D
23    use is_close_mod      , only: is_close
24    use constants_mod     , only: PI
25    use io_mod            , only: r8mat_write
26
27    implicit none
28
29    !!==================!!
30    ! External procedures !
31    !====================!=========================================
32
33
34    !!==================================!!
35    ! Public types/procedures/constants !
36    !==================================!============================
37    public :: RWG_basis_type ! Main type
38
39    ! Constants
40    integer, parameter, public :: SPATIAL_DIM = 3
41    integer, parameter, public :: NUM_FACES_IN_BASIS = 2
42    integer, parameter, public :: NUM_FACE_VERTICES = 3
43    character(*), parameter, public :: MODULE_NAME = 'RWG_basis_mod'
44
45
46    !!==================================!!
47    ! Private types/procedures/constants !
48    !==================================!============================
49    private
50
51    !!------------------------!!
52    ! Derived type definitions !
53    !------------------------!-------------------------------------
54
55
56    !!---------!!
57    ! Main type !
58    !-----------!-----------------------------------------------------
59    type RWG_basis_type
60       type (mesh_type)                          :: mesh
61       integer                                   :: num_bases
62       integer , dimension(:)      , allocatable :: basis_edges
63       integer , dimension(:, :)   , allocatable :: adjacent_faces
64       real(wp), dimension(:)      , allocatable :: basis_edge_length
65     contains
66       ! Initialisers
67       procedure, pass(this), public :: initialise
68       ! Deallocation
69       procedure, pass(this), public :: deallocate_attributes
70       ! Get-functions
71       procedure, pass(this), public :: get_num_bases
72       procedure, pass(this), public :: get_free_vertices
73       procedure, pass(this), public :: get_basis_edge_coords
74       procedure, pass(this), public :: get_basis_edge_length
```

```
75        procedure, pass(this), public :: get_adjacent_faces
76        ! Calculations
77        procedure, pass(this), public :: integrate_tested_func
78        ! Validations
79        procedure, pass(this), public :: validate_current_direction
80        ! Write procedures
81        procedure, pass(this), public :: write_RWG_basis
82
83    end type RWG_basis_type
```

B.3 PMCHW_RWG_mod

The listing below contains the interface of the module `PMCHW_RWG_mod`. The complete module is defined in the Fortran file *PMCHW_RWG_mod.f90*.

```
    ,
1   module PMCHW_RWG_mod
2   !!========================================================================
3   ! This module uses PMCHW (Poggio, Miller, Chang, Harrington,
4   ! and Wu) formulation for combining EFIE and MFIE to simulate
5   ! electromagnetic scattering on an arbitrary surface. The
6   ! scattering problem is solved using method of moments (MoM)
7   ! (often called boundary element method) with RWG basis (Rao,
8   ! Wilson, and Glisson) functions using Gelerkin's method. Integrals
9   ! are solved numerically using Gaussian quadrature formulas.
10  !
11  ! The problem consists of two regions, inside and outside of the
12  ! closed surface. The regions have different permeability and
13  ! permitivity.
14  !
15  ! The main type of the module inherits the discretisation of the
16  ! scattering surface and the RWG basis function mapping through an
17  ! instance of the RWG_basi_mod_type defined in RWG_basis_mod.f90.
18  !
19  ! Abbreviations:
20  !  CS - Closed Surface
21  !  OS - Open Surface
22  !  GQ - Gaussian Quadrature
23  !  GQF - Gaussian Quadrature Formula
24  !  GLQF - Gauss-Legendre Quadrature Formula
25  !  EFIE - Electric Field Integral Formulation
26  !  MFIE - Magnetic Field Integral Formulation
27  !
28  ! Last edited: March 7th 2021.
29  !!========================================================================
30
31    !!==============!!
32    ! Use statements !
33    !================!================================================
34    use iso_fortran_env   , only: real32, real64, real128
35    use ieee_arithmetic   , only: ieee_is_finite
36    use working_precision, only: wp
```

```
37    use RWG_basis_mod     , only: RWG_basis_type
38    use math_funcs_mod    , only: cross_prod_3D
39    use math_funcs_mod    , only: plane_wave
40    use constants_mod     , only: PI
41    use constants_mod     , only: I_IMAG
42    use constants_mod     , only: ZERO_CMPLX
43    use constants_mod     , only: ZERO
44    use constants_mod     , only: UNITY
45    use constants_mod     , only: PI4_INV
46    use is_close_mod      , only: is_close
47    use io_mod            , only: r8mat_write
48    use gauss_quad_formulas_mod, only: GQF_triangle_3pnt
49    use gauss_quad_formulas_mod, only: GQF_Legendre_3pnt
50    use gauss_quad_formulas_mod, only: GQF_Legendre_5pnt
51
52    implicit none
53
54    !!==================!!
55    ! External procedures !
56    !=====================!=========================================
57    external :: CGESV
58    external :: ZGESV
59
60    !!==================================!!
61    ! Public types/procedures/constants !
62    !==================================!============================
63    public :: PMCHW_RWG_type ! Main type
64
65    integer     , parameter, public :: NUM_REGIONS = 2
66    integer     , parameter, public :: INC_FIELD_TYPE_PLANE_WAVE = 1
67    integer     , parameter, &
68                  public :: INC_FIELD_TYPE_SPHERICAL_WAVE = 2
69    integer     , parameter, public :: OUTER_REGION_IDX = 1
70    integer     , parameter, public :: INNER_REGION_IDX = 2
71    integer     , parameter, public :: X_IDX = 1
72    integer     , parameter, public :: Y_IDX = 2
73    integer     , parameter, public :: Z_IDX = 3
74    integer     , parameter, public :: SPATIAL_DIM = 3
75    integer     , parameter, public :: NUM_FACE_VERTICES = 3
76    integer     , parameter, public :: NUM_FACES_IN_BASIS = 2
77    integer     , parameter, public :: GQF_WEIGHT_IDX = 1
78    integer     , parameter, public :: GQF_XI_IDX = 2
79    integer     , parameter, public :: GQF_ETA_IDX = 3
80    integer     , parameter, public :: GQF_ZETA_IDX = 4
81    integer     , parameter, public :: GQF_LEGENDRE_POINT_IDX = 2
82    real(wp)    , parameter, &
83                  public :: PROP_CONST_OBS_PNT_SRC_CLOSE = -1._wp!e-11
84    logical     , parameter, public :: CAUCHY = .false.
85
86    public :: eval_green_func_integrals
87    public :: eval_outer_integrals
88    public :: face_pair_integral_EFIE
89    public :: face_pair_integral_MFIE
90    public :: surface_intgr_solution
```

```
91    public :: line_intgr_solution
92    public :: inner_intgr_of_subtr_terms
93    public :: green_func_smoothened
94    public :: calc_edge_unit_normals
95    public :: map_GLQF_pnt_to_triangle_edge
96    public :: dbl_singularity_intgr
97    public :: eval_subtracted_terms
98    public :: calc_green_func
99    public :: calc_grad_of_green_func
100   public :: Cauchy_principal_value
101
102   !====================================!
103   ! Private types/procedures/constants !
104   !====================================!==========================
105
106   private
107   !!------------------------!!
108   ! Derived type definitions !
109   !------------------------!-----------------------------------
110
111   !!---------!!
112   ! Main type !
113   !-----------!-----------------------------------------------------
114   type PMCHW_RWG_type
115      type (RWG_basis_type)                        :: RWG_basis
116      complex(wp), dimension(NUM_REGIONS)          :: permeabilities
117      complex(wp), dimension(NUM_REGIONS)          :: permitivities
118      real(wp)                                     :: angular_frequency
119      complex(wp), dimension(:, :), allocatable :: PMCHW_matrix
120      complex(wp), dimension(:, :), allocatable :: q_vectors
121      complex(wp), dimension(:, :), &
122          allocatable :: expansion_coeff_alpha
123      complex(wp), dimension(:, :), &
124          allocatable :: expansion_coeff_beta
125      complex(wp), dimension(:, :), allocatable :: inc_E_field_ampl
126      complex(wp), dimension(:, :), allocatable :: inc_H_field_ampl
127      real(wp)   , dimension(:, :), &
128          allocatable :: inc_wave_direction ! unit-
129      integer    , dimension(:)   , allocatable :: inc_field_type
130      integer                                   :: num_q_vectors
131    contains
132      ! Initialisers
133      procedure, pass(this), public :: initialise
134      ! Deallocation
135      procedure, pass(this), public :: deallocate_attributes
136      ! Get-functions
137      procedure, pass(this), public :: get_permeability
138      procedure, pass(this), public :: get_permitivity
139      procedure, pass(this), public :: get_angular_frequency
140      procedure, pass(this), public :: get_num_q_vectors
141      procedure, pass(this), public :: get_PMCHW_matrix_size
142      procedure, pass(this), public :: get_q_vectors_size
143      procedure, pass(this), public :: get_q_vectors
144      procedure, pass(this), public :: get_PMCHW_matrix
```

```
145       procedure, pass(this), public :: get_solutions
146       ! Calculations
147       procedure, pass(this), public :: calc_q_vectors
148       procedure, pass(this), public :: calc_q_vectors_direct
149       procedure, pass(this), public :: calc_PMCHW_matrix
150       procedure, pass(this), public :: D_and_K_matrix_element_mn
151       procedure, pass(this), public :: inc_E_and_H_field_at_obs_pnt
152       procedure, pass(this), public :: solve_matrix_equation
153       procedure, pass(this), &
154           public :: E_and_H_field_at_obs_pnt
155       procedure, pass(this), &
156           public :: E_and_H_field_at_obs_pnt_direct
157       procedure, pass(this), &
158           public :: bistatic_scattering_cross_section
159       procedure, pass(this), public :: face_centroid
160       procedure, pass(this), public :: are_obs_pnt_and_src_close
161       procedure, pass(this), public :: get_edge_lengths
162       procedure, pass(this), &
163           public :: D_and_K_matrix_element_mn_direct
164       ! Writing data
165       procedure, pass(this), public :: write_solutions
166
167   end type PMCHW_RWG_type
```

B.4 io_mod

The listing below contains the interface of the module `io_mod`. The complete module is defined in the Fortran file *io_mod.f90*.

```
    ,
1   module io_mod
2   !!========================================================================
3   ! This module contains I/O procedures, and has a procedure
4   ! specifically designed to read Gmsh2 files. To be used together
5   ! with mesh_mod.
6   !
7   ! Last edited: March 7th 2021.
8   !!========================================================================
9
10    !!==============!!
11    ! Use statements !
12    !================!================================================
13    use working_precision, only: wp
14    use iso_fortran_env  , only: IOSTAT_END
15    use iso_fortran_env  , only: ERROR_UNIT
16    use ieee_arithmetic  , only: ieee_is_finite
17    use ieee_arithmetic  , only: ieee_is_nan
18
19    implicit none
20
21
22    !!================================!!
```

```
23    ! Public types/procedures/constants !
24    !=================================!==========================
25    public  :: open_read_gmsh2
26    public  :: string_to_int4
27    public  :: string_to_real_wp
28    public  :: read_nth_int4
29    public  :: read_n_last_int4
30    public  :: read_n_last_real_wp
31    public  :: count_int4_on_string
32    public  :: count_real_wp_on_string
33    public  :: capitalise_char
34    ! Procedures by John Burkardt for writing tables to file
35    public  :: r8mat_write
36    public  :: get_unit
37
38    !!===============================!!
39    ! Private types/procedures/constants !
40    !=================================!==========================
41    private :: check_ioerr_opening
42    private :: check_ioerr_reading
43
44
45    !=======!=========================!==============================
46  contains  ! /\/\/\/\/\/\/\/\/\/\/\!/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
47    !=======!=========================!==============================
48
49
50    !!================!!
51    ! Public procedures !
52    !==================!===========================================
53
54    !!-----------------------------------------!!
55    ! Specific file format open and read routines !
56    !-----------------------------------------!-----------------
57    subroutine open_read_gmsh2(FILENAME, spatial_dim, &
58    element_order, nodes, elements)
59      character(len=*)              , intent(in)  :: FILENAME
60      integer                       , intent(in)  :: spatial_dim
61      integer                        , intent(in)  :: element_order
62      real(wp), dimension(:, :), allocatable, intent(out) :: nodes
63      integer , dimension(:, :), allocatable, intent(out) :: elements
64      ! Variables for internal use --------------------------------
65      integer , parameter       :: BUFFER_LEN = 255
66      integer,  parameter       :: IOMSG_LEN = 255
67      character(len=BUFFER_LEN) :: line
68      character(len=IOMSG_LEN)  :: iotxt
69      integer                   :: line_nr
70      integer                   :: num_nodes
71      integer                   :: num_elements
72      integer                   :: num_elements_tot
73      integer                   :: element_line_start
74      integer                   :: element_line_end
75      integer                   :: unit_nr
76      integer                   :: level
```

```
77      integer                     :: int4_value
78      real(wp)                    :: real_wp_value
79      integer                     :: length
80      integer                     :: ioerr
81      integer                     :: error_state
82      integer                     :: i, j, k
83      !-------------------------------------------------------------
84      !/\_/\_/\_/\_/\_/\_/\_/\_/\__DOCSTRING__/\_/\_/\_/\_/\_/\_/\
85      ! This routine loads a Gmsh2 ASCII file given by a file name,
86      ! the spatial order of the mesh, and the element order. It
87      ! reads the file line by line and successively progresses
88      ! through levels, which are activated by
89      ! keywords in the .msh-file.
90      !
91      ! Arguments:
92      !     FILENAME - The path to the .msh
93      !     spatial dim - The spatial order of the mesh
94      !     element order - The order of the elements in the mesh.
95      ! Result:
96      !     nodes - A matrix contain the nodes of the mesh and their
97      ! Cartesian
98      !             coordinates.
99      !     elements - A matrix containing the elements of the mesh,
100     ! defined by
101     !                the indices of the nodes it comprises.
102     !-------------------------------------------------------------
103
104     open (newunit=unit_nr, file=FILENAME, status='old', &
105     action='read', iostat=ioerr, iomsg=iotxt)
106     call check_ioerr_opening(ioerr, iotxt, IOMSG_LEN, FILENAME, &
107     1, 'reading')
108
109     ! Read file and interpret line by line
110     level = 0
111     num_elements = 0
112     line_nr = 0
113     do
114        length = 1
115        line_nr = line_nr + 1
116
117        read (unit_nr, '(a)', iostat=ioerr, iomsg=iotxt) line
118        if (ioerr /= 0) then
119           call check_ioerr_reading(ioerr, iotxt, IOMSG_LEN,&
120           FILENAME, 3)
121           exit
122        end if
123        ! Read nodes
124        if (level == 0) then
125           if (line(1:6) == '$Nodes') then
126              level = 1
127           end if
128        else if (level == 1) then
129           call string_to_int4(line, length, int4_value, &
130           error_state)
```

```
131            num_nodes = int4_value
132            allocate(nodes(num_nodes, spatial_dim))
133            j = 0
134            level = 2
135         else if (level == 2) then
136            if (line(1:9) == '$EndNodes') then
137               level = 3
138            else
139               j = j + 1
140               call read_n_last_real_wp(line, spatial_dim, &
141               nodes(j, :), &
142                    num_real_wp_in=(spatial_dim + 1))
143            end if
144
145         ! Read elements
146         else if (level == 3) then
147            if (line(1:9) == '$Elements') then
148               level = 4
149            end if
150         else if (level == 4) then
151            call string_to_int4(line, length, int4_value, &
152            error_state)
153            num_elements_tot = int4_value
154            level = 5
155         else if (level == 5) then
156            int4_value = read_nth_int4(line, 2)
157            if (int4_value == 2) then
158               element_line_start = line_nr
159               num_elements = 1
160               level = 6
161            end if
162         else if (level == 6) then
163            if (line(1:12) == '$EndElements') then
164               level = 7
165               rewind(unit_nr)
166            else
167               int4_value = read_nth_int4(line, 2)
168               if (int4_value /= 2) then
169                  level = 7
170                  rewind(unit_nr)
171               end if
172            end if
173            if (level == 6) then
174               num_elements = num_elements + 1
175            end if
176         else if (level == 7) then
177            allocate(elements(num_elements, element_order))
178            level = 8
179            j = 0
180            element_line_end = line_nr - 1
181            line_nr = 1
182         else if (level == 8) then
183            if (line(1:12) == '$EndElements') then
184               exit
```

```
185             else if (j == num_elements) then
186                exit
187             else if (line_nr == element_line_end) then
188                exit
189             else if (line_nr >= element_line_start) then
190                j = j + 1
191                call read_n_last_int4(line, element_order, &
192                elements(j, :))
193             end if
194          end if
195       end do
196
197       close (unit_nr)
198
199       call validate_nodes_and_elements(spatial_dim, element_order, &
200       num_nodes, num_elements, nodes, elements)
201
202    end subroutine open_read_gmsh2
```

Appendix C

Gmsh2 format and Makefiles

C.1 A Gmsh2 ASCII file

The Gmsh2 file for the tetrahedron in Fig. 3.4 is listed below. See Section 3.1.2 for a description of the format.

```
,
$MeshFormat
2.000000 0 8
$EndMeshFormat
$Nodes
4
1 1. -1. 1.
2 -1. 1. 1.
3 1.0 1.0 -1.0
4 -1. -1. -1.
$EndNodes
$Elements
4
1 2 0 1 3 2
2 2 0 1 2 4
3 2 0 2 3 4
4 2 1 0 1 4 3
$EndElements
```

C.2 Makefiles

This section lists two of the *Makefiles* used to compile and organise the numerical implementation of Chapter 3, in addition to the file *make.inc*. See Section 3.3 for description of *Make* and how the Makefiles below are used.

C.2.1 Top directory Makefile

```
#=============================================================
# Top level Makefile for pre-processing, compiling and linking all
```

```
# files
#
#   Last edited: Marhch 7th 2021.
#=====================================================================
.SUFFIXES:
TOPDIR = .
include $(TOPDIR)/make.inc

PROGRAMS    = lib testing

all: $(PROGRAMS)
@echo "Build successfull.."

lib:
$(MAKE) -C src

testing: lib
$(MAKE) -C testing

sims: lib
$(MAKE) -C programs

clean_sims:
$(MAKE) -C programs clean
clean:
$(MAKE) -C src clean
$(MAKE) -C testing clean
$(MAKE) -C programs clean
@echo "Clean successfull.."
```

C.2.2 Makefile in the src directory

```
#=====================================================================
# Makefile for compiling fortran modules in src.
#   First level of Makefile recursion.
#
#   Last edited: March 7th 2020.
#=====================================================================
.SUFFIXES:
TOPDIR   = ..
include $(TOPDIR)/make.inc

MODPATH  = $(TOPDIR)/$(MOD_DIR)
LDFLAGS += -I$(MODPATH) -J$(MODPATH)
```

```
# working_precision needs to be listed first as it is used by all
# modules.
# Dependency on working_precision may be explicitly stated instead
SOURCES = \
working_precision.f90 \
mesh_mod.f90 \
io_mod.f90 \
test_utilities.f90 \
RWG_basis_mod.f90 \
gauss_quad_formulas_mod.f90 \
math_funcs_mod.f90 \
is_close_mod.f90 \
constants_mod.f90 \
PMCHW_RWG_mod.f90

OBJECTS = $(subst .f90,.o, $(SOURCES))
MODULES = $(subst .f90,.mod, $(SOURCES))


all:  $(MODPATH) $(OBJECTS)

$(MODPATH):
@mkdir -p $(MODPATH)

%.o $(MODPATH)/%.mod: %.f90
$(FC) $(LDFLAGS) -c $< $(LDLIBS)
@touch $@

io_mod.o: \
working_precision.o

mesh_mod.o: \
working_precision.o \
math_funcs_mod.o   \
is_close_mod.o   \
constants_mod.o   \
io_mod.o

RWG_basis_mod.o: \
working_precision.o \
mesh_mod.o \
math_funcs_mod.o  \
is_close_mod.o
```

```
test_utilities.o:  \
working_precision.o \
mesh_mod.o \
RWG_basis_mod.o

PMCHW_RWG_mod.0: \
working_precision.o \
RWG_basis_mod.o \
math_funcs_mod.o  \
constants_mod.o \
is_close_mod.o \
gauss_quad_formulas_mod.o

math_funcs_mod.o: \
working_precision.o \
is_close_mod.o  \
constants_mod.o

constants_mod.o: \
working_precision.o

gauss_quad_formulas_mod.o: \
working_precision.o

is_close_mod.o: \
working_precision.o

clean:
$(RM) $(OBJECTS)
$(RM) $(MODPATH)/*.mod
$(RM) -d $(MODPATH)
```

C.2.3 *make.inc*

```
#======================================================================
# make include file
#
#   Last edited: March 7th 2021.
#======================================================================

SHELL = /bin/sh
```

```
# Modify the variables to desired compiler, flags and machine
# architecture.
#
FC          = gfortran
FFLAGS      = --pedantic
LDFLAGS     = -O3 #-g -fcheck=all -ffpe-trap=invalid
TARGET_ARCH = -march=x86-64

# Choose libraries. LAPACK and BLAS is required.
LAPACK      = -llapack
BLAS        = -lblas
OPENBLAS    = -lopenblas
LAPACK95    = -llapack95
#
#LDLIBS      = $(OPENBLAS)
LDLIBS      = $(LAPACK) $(BLAS)

# Path to directories where module files are stored.
MOD_DIR     = modules
# Path to where executables are stored
BIN_DIR     = bin
```

Eilif Sommer Øyre

Electromagnetic Scattering Calculations for Arbitrarily Shaped Closed Surfaces using the Method of Moments

NTNU
Norwegian University of
Science and Technology